Tariq El-Jumaily 2024 A-Level Computer Science Coursework

Table of contents

Analysis

Problem Description

For my project, I will create an augmented reality interactive mobile application that will be composed of an attractive user interface. The main screen will show the mobile phone reverse camera that displays the user's surroundings. The aim is to create an “all-in-one” tool that can assist people with visual impairments. The user will point the camera at a particular person and the program would be able to identify and output a guess of the surroundings and mood, based on facial expressions. There will also be an option to read this out so that the user can perhaps get a better mental visualisation for if their sight is not very capable of seeing the screen. Furthermore, the program can prompt the user to take a picture and create a basic caption of the image’s description which can also be read out via a text-to-speech interface for the user.

For the user interface and style, I will use an open source library of pre-designed buttons and text fonts which will create an attractive display, similar perhaps, to the style of minimalist apps and operating systems such as, Uber, and iOS. Since I lack previous expertise in UI design, this allows me to spend less time designing assets, which will be beneficial in terms of time costs.

Many current, similar systems often make use of a pre-trained or custom-trained neural network to identify patterns and recognition between various states of emotion and character. Some algorithms identify facial features such as eye positions and wrinkle patterns, alongside the mouth and eyebrow shape to almost ‘piece together’ a particular mood. E.g. an image of a human with squinted eyes, raised eyebrows, and upwards curved mouth, with side eye wrinkles would indicate a common combination of a possible ‘happy’ mood. Another example would be the combination of a human with thinner lips, higher cheekbones, and smoother skin complexion would indicate a possible female. While developing my system, accuracy and responsiveness would be a high priority in order to provide a highly interactive and usable experience.

One popular option that can be used to identify a face, facial expressions, age and gender, is OpenCV's DNN module or DeepFace, which support various pre-trained models. For mood estimation, I can use a pre-trained model like FER2013 (Facial Expression Recognition). And to caption the image I will also need to use another pre-trained AI model, this can be one such as the show-and-tell model, which is based on a neural network architecture that combines a CNN (Convolutional Neural Network) for image feature extraction and an LSTM (Long Short-Term Memory) for generating captions.

Overall, the user may not have much experience with technology, consequently, the interface should be overall simple and straightforward.

Stakeholders

This project targets a broad audience of mobile app users, particularly those who have an interest in augmented reality (AR) and artificial intelligence (AI). The application will feature a user-friendly interface that allows users to interact with AI technology in a simple, fun, and engaging way. This makes it suitable for all, irrespective of their knowledge level in AI and AR.

My primary target demographic is individuals who are tech-savvy and curious about innovations in AI and AR while also considering those who might find practical use-cases for the application, such as educators teaching AI, those who may benefit from the application (perhaps those with visual impairments), or researchers studying human interaction with technology.

My app aims to bridge the gap between sophisticated AI technologies and everyday smartphone users. It leverages AR technology to create an interactive experience, making use of AI to analyse and interpret facial features and expressions, and subsequently, display information about the person's environment and mood.

The application will use intuitive touchscreen controls and simple user interface elements, making it accessible to a wide range of users, including those who may not be familiar with advanced technologies, such as perhaps elderly people. Moreover, given that the app is developed for mobile platforms, it will cater to a substantial number of smartphone users around the world.

I have identified three specific stakeholders to gain valuable feedback throughout the development process. They include a 48-year-old software developer, Aiad Tarik, who regularly interacts with cutting-edge technology and could provide insights on the app's performance and usability. A 17-year-old student, Mario Prifti, who has a curious insight into developing technologies, along with a heavily creative mind, which would help me tackle the issue of making the project engaging and interesting to people of all ages. Finally, a 75-year-old retired farmer, Mike Parish, who could provide valuable insights on the functionality of the technology, since he is not as tech-confident and his insight would help me develop my app to be usable for those of every background.

Why can the problem be solved by computational methods

Abstraction and visualisation

For my project, I'm going to use abstraction in a variety of ways to shave off redundant information and make my problem simpler. To make the user's experience with the app better, the use of abstraction will also conceal potentially confusing details from them.

Ways I will use abstraction and visualisation:

- **Clear and simplified graphics:**
    - A bounding box around the user's face will clearly define their position and identify their name just outside of the box, making it easier and clearer for the user to see and visually process

- **Minimalistic and direct UI**
    - Simple, and minimalistic UI is going to be used to prevent confusion for users, especially those who are not confident in using technology

- **Accessibility options:**
    - Text-To-Speech capability will be implemented in order to reduce the amount of information on screen, which can be overwhelming for some, and useful for those who have visual problems.

Thinking Ahead

The prerequisites, inputs, outputs, and reusable components of my project will all be established before I begin to apply the principle of forward computation to it. This is important because, before I begin writing scripts, I need to have a complete understanding of all the anticipated inputs and outputs. Recognising the prerequisites for any operations that will be carried out is also crucial.

I've listed all the app's necessary inputs, outputs, and additional prerequisites in the sections below:

- **Inputs:**
    - Laptop webcam
    - Phone Camera (front/back)
    - Target image (for recognition)
    - Pixel size of image/video feed
    - Colour format of inputted image/video feed
    - UI button for text-to-speech functionality
    - UI button for image captioning functionality

- **Outputs:**
    - Bounding box around face
    - Name of person (text)
    - Emotion of person (text)
    - Spoken (audio) text-to-speech
    - Image caption (text)

- **Additional Prerequisites (for development)**
    - DeepFace library
    - OpenCV library
    - Unity library
    - FER2013 training data

- **Additional Prerequisites (for user)**
    - Device with internet connection (needed to download model weights files locally)
    - Device with working camera
    - Device with working speaker
    - Higher performance devices will produce more accurate results faster
    - 4-8GB memory in order to keep up with heavy duty of facial recognition

Thinking Procedurally

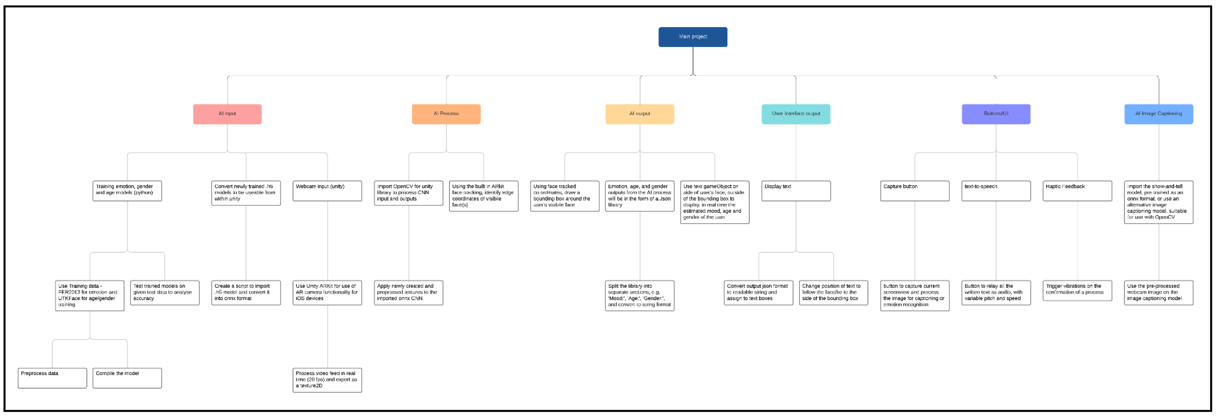
I should deconstruct my problem into more manageable subproblems that can be coded independently in order to implement a procedural approach to my development. This would enable me to modularize my code, which will aid in testing and debugging and cut down on the amount of time it takes to develop new components

The program can be divided into numerous subsections, such as image input, which can be further divided into image processing, which is further divided into resizing, colouring, and normalisation. This will lead to the creation of unique functions that, provided they have the right parameters for the particular application, only need to be coded once and can be used repeatedly throughout the entire system.

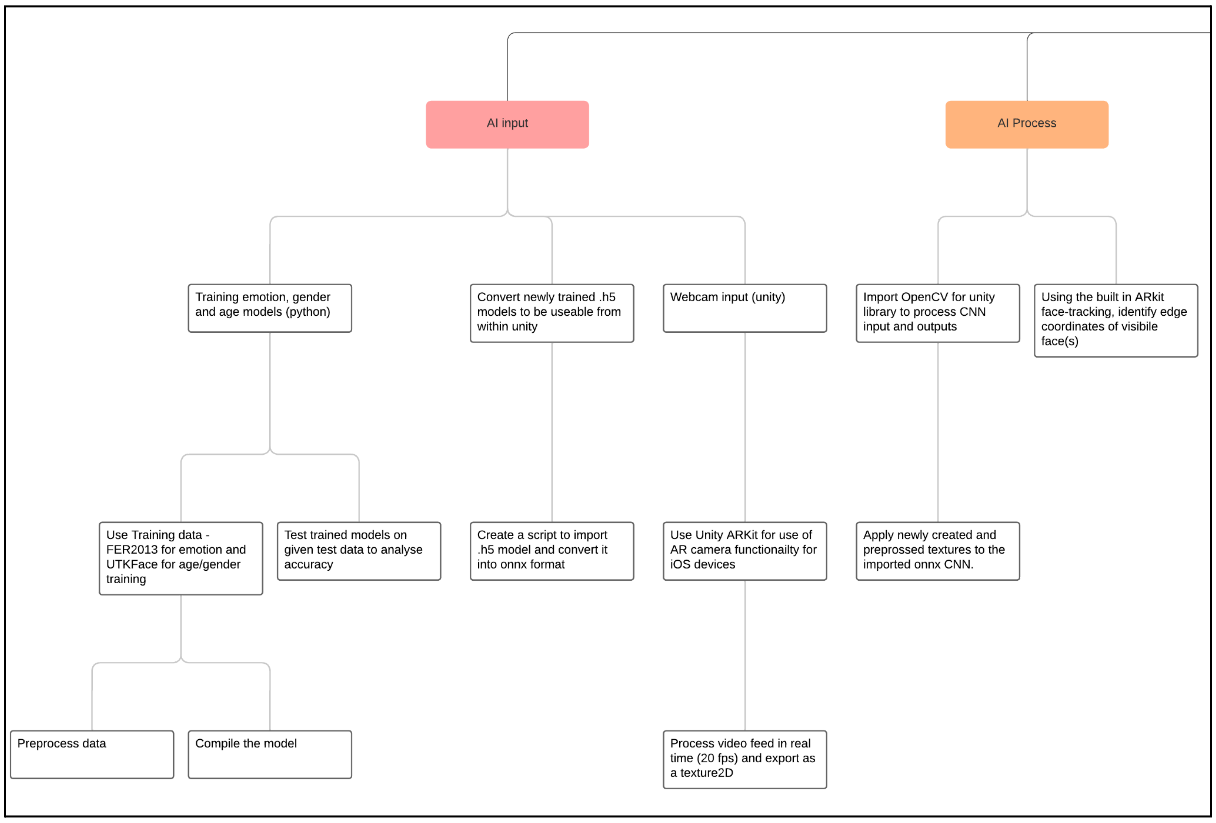
These functions, characterised by their reusable nature, serve as the bedrock of our procedural approach. By ensuring that they are well-parameterized and suited to the specific task at hand, we can significantly decrease the necessity of repetitive code, enhancing the overall efficiency of the system. Code reuse not only reduces the time spent on development, but it also diminishes the chances of errors creeping into the system.

Modular breakdown of the program:

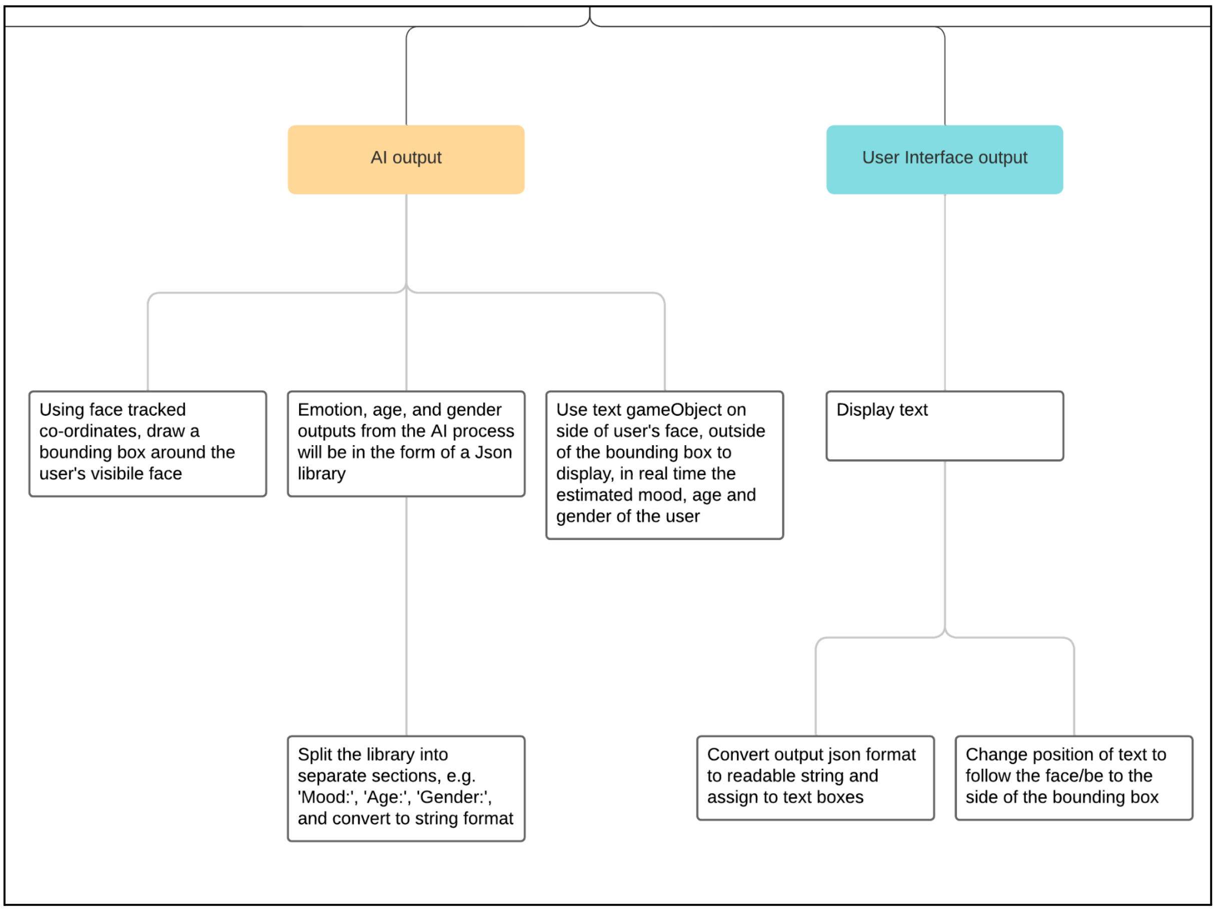
Overview:


Main project
AI input
AI Process
AI output
User Interface output
Buttons/UI
AI Image Captioning
Training emotion, gender and age models (python)
Convert newly trained .h5 models to be useable from within unity
Webcam input (unity)
Import OpenCV for unity library to process CNN input and outputs
Using the built in ARkit face-tracking, identify edge coordinates of visibile face(s)
Using face tracked co-ordinates, draw a bounding box around the user's visibile face
Emotion, age, and gender outputs from the AI process will be in the form of a Json library
Use text gameObject on side of user's face, outside of the bounding box to display, in real time the estimated mood, age and gender of the user
Display text
Capture button
text-to-speech
Haptic Feedback
Import the show-and-tell model, pre-trained as an onnx format, or use an alternative image captioning model, suitable for use with OpenCV
Use Training data - FER2013 for emotion and UTKFace for age/gender training
Test trained models on given test data to analyse accuracy
Create a script to import .h5 model and convert it into onnx format
Use Unity ARKit for use of AR camera functionailty for iOS devices
Apply newly created and preprossed textures to the imported onnx CNN.
Split the library into separate sections, e.g. 'Mood:', 'Age:', 'Gender:', and convert to string format
Convert output json format to readable string and assign to text boxes
Change position of text to follow the face/be to the side of the bounding box
button to capture current screenview and process the image for captioning or emotion recognition
Button to relay all the written text as audio, with variable pitch and speed
Trigger vibrations on the confirmation of a process
Use the pre-processed webcam image on the image captioning model
Preprocess data
Compile the model
Process video feed in real time (20 fps) and export as a texture2D

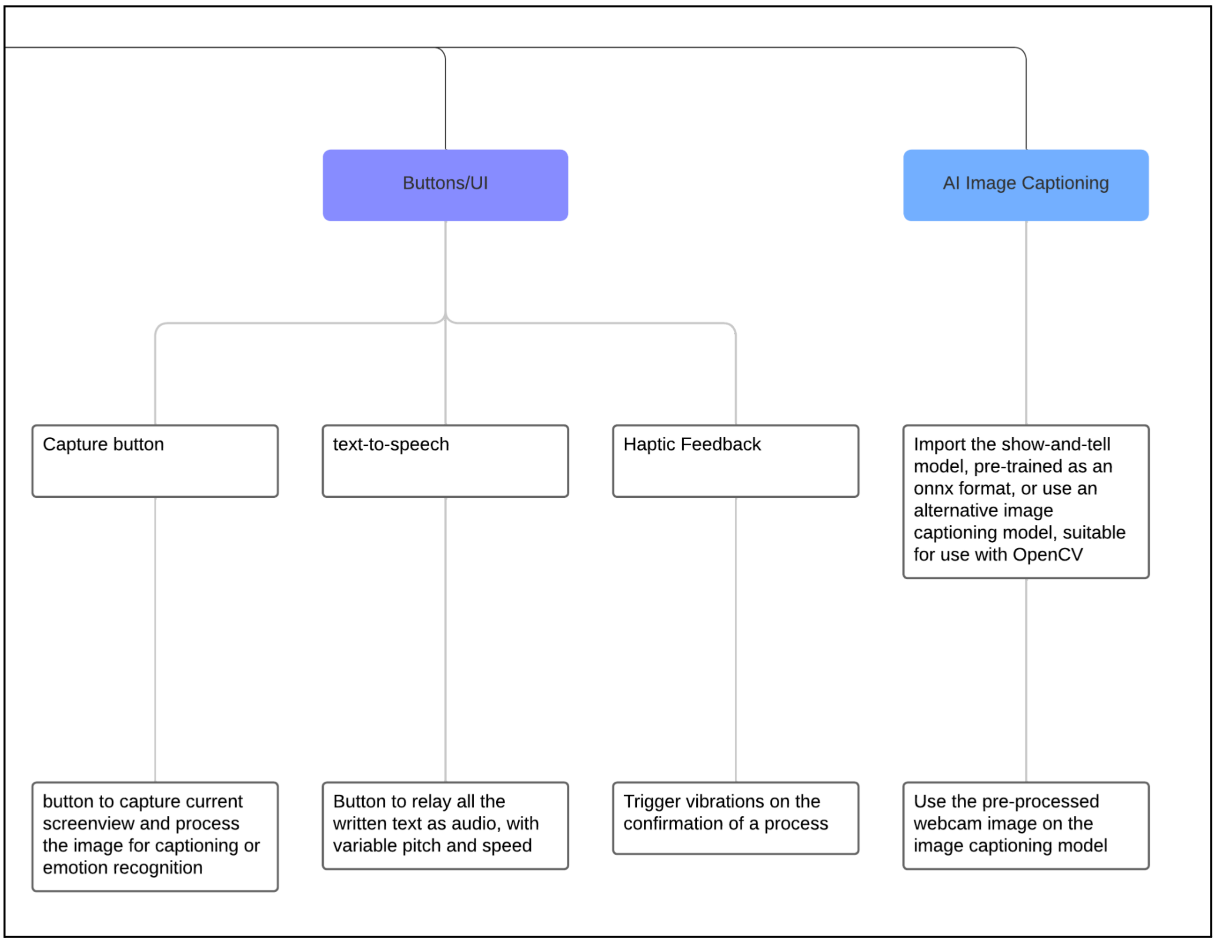
Section 1 & 2:


AI input
AI Process
Training emotion, gender and age models (python)
Convert newly trained .h5 models to be useable from within unity
Webcam input (unity)
Import OpenCV for unity library to process CNN input and outputs
Using the built in ARkit face-tracking, identify edge coordinates of visibile face(s)
Use Training data - FER2013 for emotion and UTKFace for age/gender training
Test trained models on given test data to analyse accuracy
Create a script to import .h5 model and convert it into onnx format
Use Unity ARKit for use of AR camera functionailty for iOS devices
Apply newly created and preprossed textures to the imported onnx CNN.
Preprocess data
Compile the model
Process video feed in real time (20 fps) and export as a texture2D

Section 3 & 4:


AI output
User Interface output
Using face tracked co-ordinates, draw a bounding box around the user's visibile face
Emotion, age, and gender outputs from the AI process will be in the form of a Json library
Use text gameObject on side of user's face, outside of the bounding box to display, in real time the estimated mood, age and gender of the user
Display text
Split the library into separate sections, e.g. 'Mood:', 'Age:', 'Gender:', and convert to string format
Convert output json format to readable string and assign to text boxes
Change position of text to follow the face/be to the side of the bounding box

Section 5 & 6:


Buttons/UI
AI Image Captioning
Capture button
text-to-speech
Haptic Feedback
Import the show-and-tell model, pre-trained as an onnx format, or use an alternative image captioning model, suitable for use with OpenCV
button to capture current screenview and process the image for captioning or emotion recognition
Button to relay all the written text as audio, with variable pitch and speed
Trigger vibrations on the confirmation of a process
Use the pre-processed webcam image on the image captioning model

Thinking Logically

It is essential to approach problem-solving from a logical perspective. This logical thinking, in a programming context, refers to the identification of decision points within the program that can cause it to branch off or repeat certain processes based on specific conditions. These decision points essentially steer the flow of the program, allowing it to adapt and respond to different scenarios.

- **Logical Flow in Facial Recognition:**
  - For instance, the facial recognition feature in the proposed application serves as an excellent case study of logical thinking in action. When the user points the camera at a person, the program needs to analyze the image and make decisions based on what it detects. The first decision point might involve determining whether a face is present in the image or not. If a face is detected, the program branches off to identify the person's mood based on their facial features. Each of these attributes requires its own set of decision points. For example, the determination of mood could branch off into various emotions like happiness, sadness, anger, or surprise, each identified by distinctive facial cues.

- **Incorporating User-Interactive Decision Points:**
  - Incorporating decision points that involve user interaction is another significant aspect of logical thinking. In the proposed application, users can opt for an audio description of the identified facial features. This introduces another decision point where the program must decide whether to output the results through text or convert the text to speech based on the user's preference.

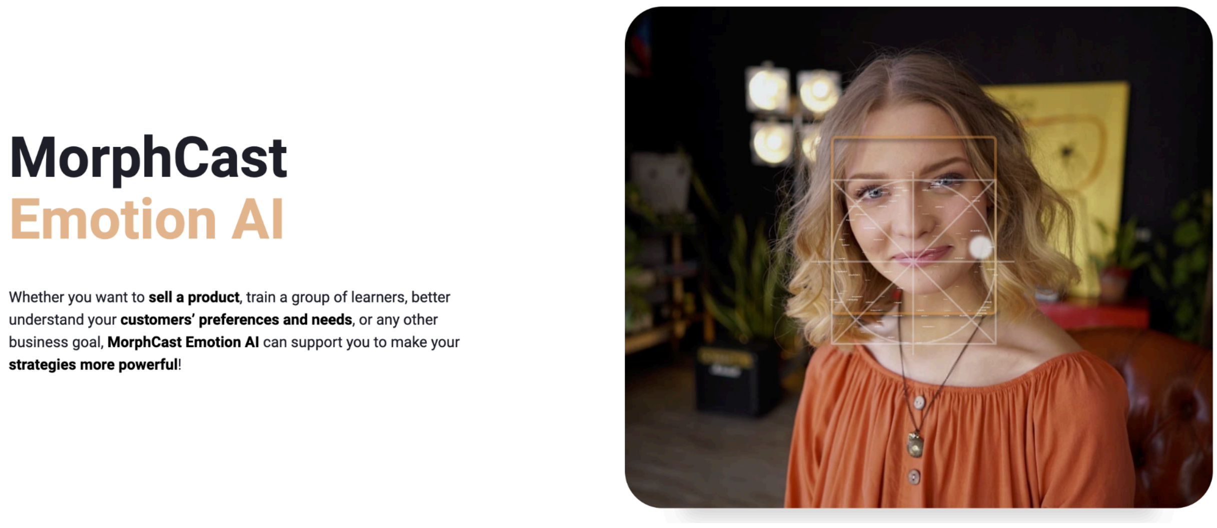
- **Caption Generation and Logical Decisions:**
  - The caption generation feature introduces yet another realm of decision-making. Upon taking a picture, the program must logically dissect the image contents and compose a suitable caption. The decision points here may involve determining the number of objects in the image, identifying the objects, and generating a grammatically correct and contextually relevant caption.

- **Error Handling and Logic:**
  - Finally, logical thinking is vital in error handling. Decision points must be included to detect potential issues such as poor lighting or obstructed views that could interfere with image processing. Upon encountering such issues, the program can decide to request the user to retake the image or adjust their position, thereby ensuring optimal operation.

Research

MorphCast Emotion AI

MorphCast is a tech company offering advanced solutions in the field of facial recognition and facial feature detection technologies. Their primary product is client side emotion AI software, which allows developers to integrate facial recognition capabilities into their applications.



MorphCast
Emotion AI

Whether you want to **sell a product**, train a group of learners, better understand your **customers' preferences and needs**, or any other business goal, **MorphCast Emotion AI** can support you to make your **strategies more powerful**!

MorphCast Emotion AI Description
https://www.morphcast.com/

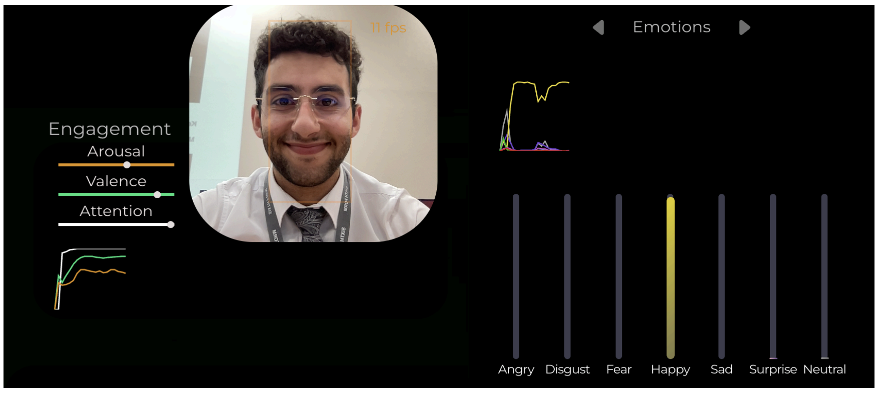
The company also offers an array of APIs for web and mobile platforms that allow computers to recognise, understand, and imitate human emotions. A subset of emotion AI called facial emotion recognition (FER) aims to identify emotions from facial expressions. Based on the recognition of users' moods, sentiments, or emotions as expressed in their facial expressions, MorphCast Emotion AI can be used to create software applications and services that engage with users in a more human and natural way.

Like most other similar solutions, MorphCast uses convolutional neural networks (CNN) to extract and identify particular facial expressions and features by learning and identifying patterns in large, labelled datasets. This training can take a massive amount of computational power and time but the end result is a working ‘model’ that can be lightweight and used from a variety of different devices and operating systems to make predictions on new and unseen data

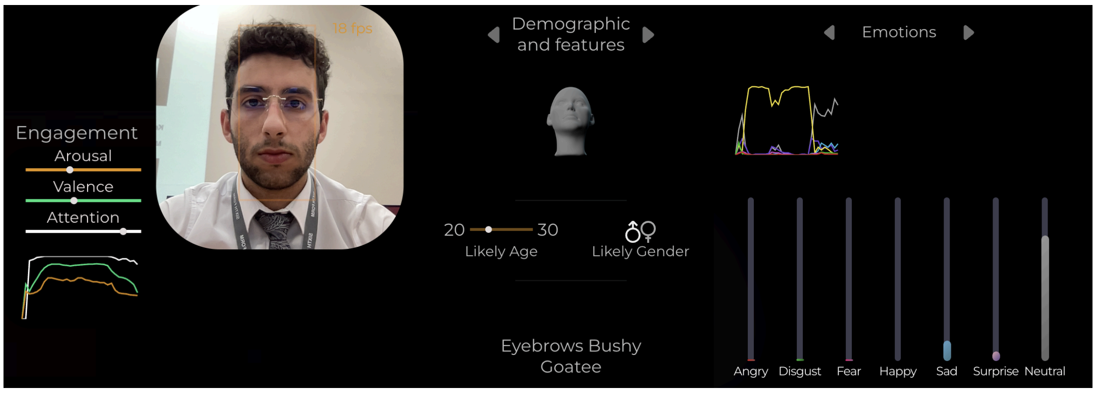
Personal review:

The way the objects detected are displayed to the user is a crucial component of MorphCast, which I will implement in my facial recognition system. Due to the stark contrast between the coloured facial bound-boxes and the rest of the image's colours, the face is fairly obvious. This would be helpful because users who need to be able to clearly identify a person in the footage, such as those with visual impairment or poor eyesight, will need to be able to do so.

Additionally, the status bars at the bottom make it easy to determine a person's status and to visually combine various emotions. However, since it is extremely complicated and mathematical and not something I could realistically produce, I will not be using the same approach to identify multiple emotions.



A demonstration of the MorphCast emotion AI Tool recognising visible 'Happy' emotions



Another demonstration of MorphCast, detecting a visible 'neutral' emotion, along with age and gender estimation

One major limitation of MorphCast is its web-based functionality. I am to make my project application-based, with all the processing happening locally, on-system, although I understand that this may be difficult and could be outside the time scope of the project, especially with limitations such as device performance, which means I may have to process the AI images over a server/network such as python flask.

Visage Technologies

Visage Technologies is a leading tech firm specialising in face tracking, analysis, and recognition solutions. They deliver innovative software for developers, focusing on their primary product - an advanced AI-driven facial recognition system. This cutting-edge technology provides developers with the capacity to seamlessly incorporate face tracking and analysis functionalities into their applications.

About us

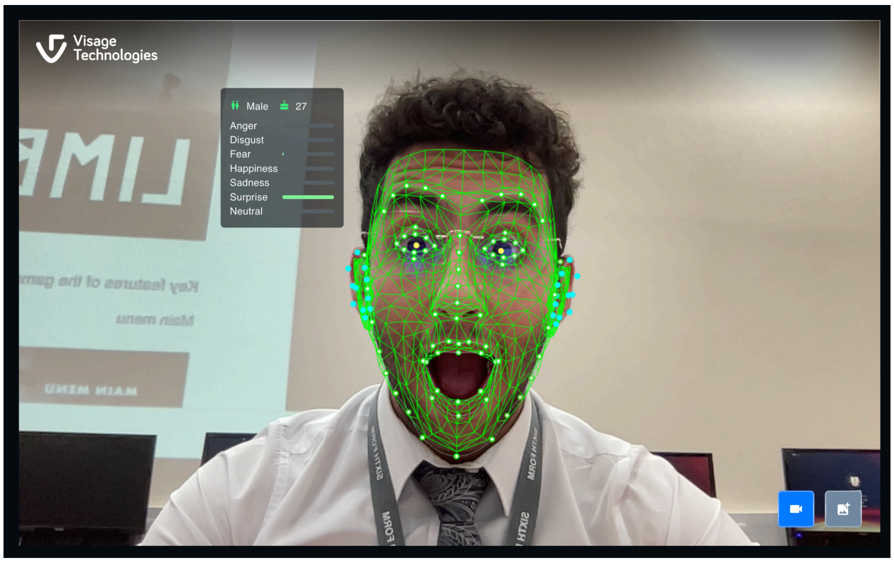
Innovating in the field of computer vision and machine learning since 2002.

Our vision is to **enable the unimaginable with computer vision technology created in a harmonic workplace**. To do that, we keep developing our knowledge and skills in the fields of computer vision, machine learning, computer graphics and general computing, and using this critical mass of knowledge and expertise to deliver world-class products and services to our clients.

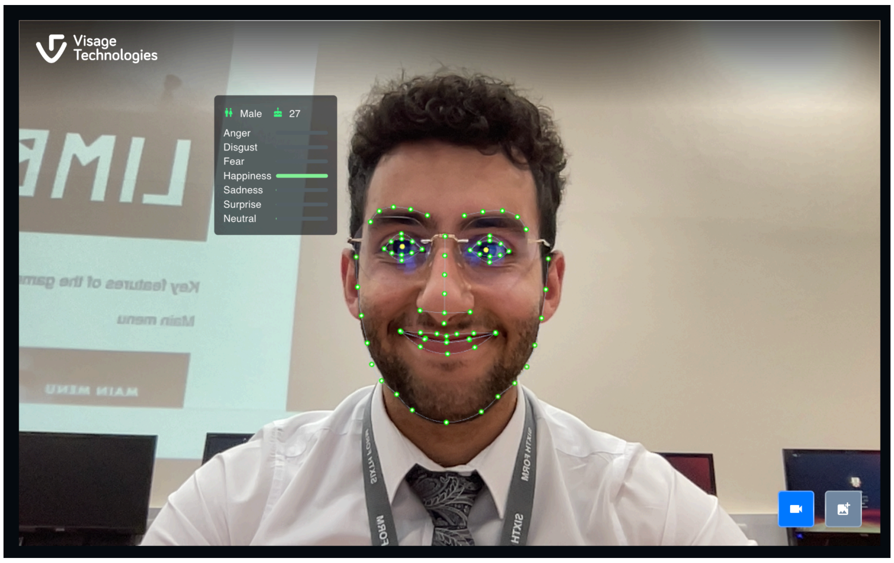
Visage technologies company description
https://visagetechnologies.com/about/

Personal review:
The way emotion and age are detected and displayed to the user is a key defining factor for Visage Technologies, which I will implement in my facial recognition system - this includes the text box for the age and gender close to their face. The face tracking nodes and wireframe allows the face to be obviously seen. This would be helpful because users who need to be able to clearly identify a person in the footage, such as those with visual impairment or poor eyesight, will need to be able to do so.

In regards to the wireframe and tracking points, since it is extremely complicated and mathematical it would not be something I could realistically produce, I will not be using the same approach to track the face, but rather utilise a basic bounding box around the face.



Detecting my face as “Surprised” and a 27-year-old male

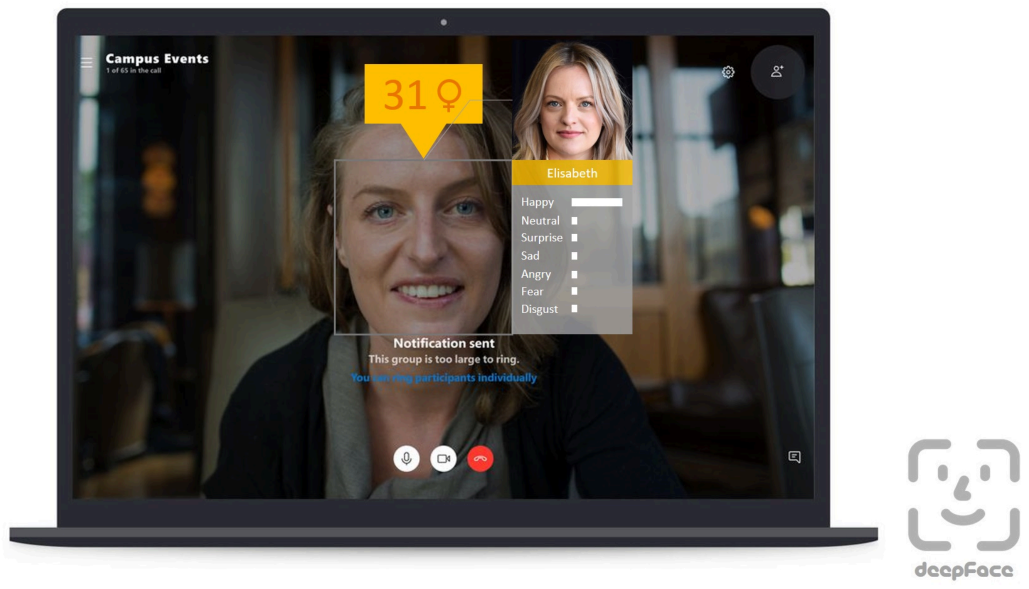


Detecting my face as a “happy” 27-year-old male (wireframe off)

DeepFace

Facebook's research team has developed an advanced facial recognition library called DeepFace, capable of accurately identifying human faces in digital photos. This powerful library incorporates several external face recognition models, including VGG-Face, Facenet, OpenFace, DeepID, ArcFace, Dlib, and SFace. Unlike most other facial recognition and analysis software, which average accuracies of around 80%, DeepFace boasts an impressive accuracy rate of 97.25%.

DeepFace's success can be attributed to its utilisation of a sophisticated nine-layer neural network comprising over 120 million link weights. This neural network is designed as a siamese network, enabling it to effectively recognize faces stored in a database. The immense accuracy of DeepFace highlights its exceptional performance in the field of facial recognition technology.



Deepface demo identifying person as “Elisabeth”, including a bounding box and gender/mood estimation, all processed in real-time through a constant stream of video images
https://github.com/serengil/deepface

A Siamese Neural Network is a form of artificial intelligence model that can learn to distinguish between two inputs, to put it simply. Consider it as a pair of identical twins who have the same upbringing (corresponding to the same weights and architecture in neural network terminology) and have extremely similar worldviews.

They utilise their common knowledge to assess how similar or unlike two distinct objects are when they are provided to consider. They may not always recognise the items, but they are able to determine how similar or unlike they are based on their "learned upbringing."

When doing jobs like face recognition or signature verification, where you must compare a new signature to an old one, this type of network is extremely helpful. It all comes down to determining how similar or different two sets of data are, or in this case, if a person in an image matches a person saved in a database, critical for advanced recognition and differentiation.

Deepface may be used to stream live videos as well. The Stream feature will access your webcam and do both facial attribute analysis and face recognition. If the function can concentrate a face sequentially over five frames, analysis of the frame will begin. Results are displayed five seconds later.

Deepface additionally offers an api, which can be used and implemented with an application - this means that image processing and analysis can be done externally at a relatively low cost and high bandwidth - this eliminates the need of optimising a pre-trained model, which can come with drawbacks, such as reduced accuracy and increased chance of error e.g. not detecting a face in the picture.

Interviews

Plan:

I have chosen to formulate a comprehensive questionnaire to maximise the effectiveness of my interviews. To optimise the interview process and utilise my time efficiently, I categorised the questions into various sections, namely:

- General Controls/Usability
- Accuracy
- Personal Use
- Text-To-Speech

These categories served as a framework for planning the interviews, ensuring that I could extract the most valuable information from each stakeholder. In the upcoming discussions, I will engage with the three stakeholders mentioned earlier, namely Aiad, Mario, and Mike. To maintain clarity and organisation, Aiad's responses will be denoted in red, Mario's responses in green, and Mike's responses in blue.

Open questions will be denoted by an (O) and closed questions will be denoted by a (C).

General Controls/Useability:

1) (O) Can you describe the features of an ideal user interface for an AI and AR mobile app from your perspective?
2) (C) Do you prefer a minimalist design with fewer on-screen controls and more reliance on gestures or a more comprehensive design with on-screen controls for every function?
3) (O) If the app were to include a help or tutorial section, what essential elements should it contain to assist you in understanding the app's functionality?
4) (O) Based on your experience with mobile apps, what common features or controls do you think should be incorporated into this app to enhance its usability?
5) (C) Would you prefer the app to provide haptic feedback (e.g., vibration) as a form of interaction or confirmation of certain actions?

Accuracy:

1) (O) What would your expectations be regarding the accuracy of the app's AI in determining environments and mood from facial expressions?
2) (O) In an ideal scenario, how do you believe the app should handle uncertainties or ambiguities (for example, when the AI can't confidently determine a person's mood)?
3) (C) Do you believe a confidence percentage indicating the AI's certainty of its assessment (e.g., 85% confidence that the person is happy) would be helpful?

Personal Use:

1) (O) Can you describe a scenario where you would find this app most useful in your daily life?
2) (C) Do you anticipate using this app frequently, for example, daily, weekly, or only for special occasions?
3) (O) What specific features or functionalities would you want this app to have to best suit your personal needs and preferences?

Text-to-speech functionality:

1) (C) Would you prefer the option to adjust the speed and pitch of the text-to-speech feature to accommodate your listening comfort?
2) (O) In what situations do you think using a text-to-speech feature could be most beneficial for you or other users of the app?

Interview Summary:

1.1) (O) Can you describe the features of an ideal user interface for an AI and AR mobile app from your perspective?

Aiad:
An ideal user interface for an AI and AR mobile app, from my perspective, should be minimalist yet effective. It should include clear labels for all functions, intuitive touch gestures, and easy navigation. It should also be visually appealing, utilising a harmonious colour scheme and easy-to-read text.

Mario:
From my perspective, the perfect user interface for an AI and AR mobile application should be modern and aesthetically appealing. It should include easily distinguishable icons and a colour scheme that makes the interface enjoyable to use, and also guides the user naturally through the functions of the app.

Mike:
For me, the ideal interface for this kind of application would be simple and straightforward, with large, easy-to-read text and clear, identifiable icons. The functions of different buttons and options should be immediately clear, and there should not be too many options on the screen at once to avoid confusion.

1.2) (C) Do you prefer a minimalist design with fewer on-screen controls and more reliance on gestures or a more comprehensive design with on-screen controls for every function?

Aiad:
I prefer a minimalist design with fewer on-screen controls and more reliance on gestures. This makes for a less cluttered screen, improving the overall user experience. I believe in the power of gestures and think that they can provide a more immersive and engaging user experience.

Mario:
Personally, I appreciate on-screen controls for precise actions, but I'm also comfortable with gesture controls. Gestures can make the interaction smoother and faster, and they seem more futuristic, which I find really cool!

Mike:
At my age, gesture controls might be a bit difficult to master. I prefer on-screen buttons as they give me more certainty in what I'm doing.

1.3) (O) If the app were to include a help or tutorial section, what essential elements should it contain to assist you in understanding the app's functionality?

Aiad:
If the app were to include a help or tutorial section, it should contain step-by-step guides on how to use the app's features. Visual aids, like screenshots or short video clips, would be very beneficial. There should also be a troubleshooting section to help users resolve common problems.

Mario:
I believe a tutorial is essential for any application, especially for those involving complex technologies like AI and AR. A tutorial that walks me through each feature of the app with practical examples would be beneficial. It should be interactive and dynamic, allowing me to actually try the features as I learn about them.

Mike:
Yes, I believe a tutorial would be quite necessary for me. It should explain each feature clearly and provide instructions on how to navigate the app. The tutorial should be interactive, allowing me to practice using the app as I learn.

1.4) (O) Based on your experience with mobile apps, what common features or controls do you think should be incorporated into this app to enhance its usability?

Aiad:
Based on my experience with mobile apps, I think common features that enhance usability include a search function, customizable settings, and feedback options. Also, gesture controls such as pinch to zoom, swipe to navigate, and long press for more options are handy and intuitive.

Mario:
Some features I consider essential for improving usability are a clean and intuitive layout, responsive design, and efficient loading times. I also appreciate when an app allows for personal customization, like changing themes or adjusting settings to suit my preferences.

Mike:
For me, larger buttons and text, a clean and uncluttered interface, and easily identifiable icons are important for usability. In addition, the app should respond quickly and not lag or freeze.

1.5) (C) Would you prefer the app to provide haptic feedback (e.g., vibration) as a form of interaction or confirmation of certain actions?

Aiad:
Yes, I believe haptic feedback can be very useful in providing non-visual cues to users. For example, vibrations could indicate a successful action, like taking a picture, or alert users to an error. This form of interaction can greatly enhance the user experience by providing real-time feedback.

Mario:
Yes, haptic feedback can greatly enhance the user experience. It provides an immediate and intuitive sense of interaction, which can make the app feel more responsive and engaging.

Mike:
Yes, haptic feedback would be very helpful. It would confirm that I've successfully pressed a button or performed an action, which can be reassuring.

2.1) (O) What would your expectations be regarding the accuracy of the app's AI in determining age, gender, and mood from facial expressions?

Aiad:
My expectations regarding the accuracy of the app's AI in determining age, gender, and mood would be fairly high. Although I understand that there may be occasional errors, as is often the case with AI, the app should be correct most of the time to build user trust and provide a meaningful experience.

Mario:
I would expect the AI to be reasonably accurate most of the time. However, I understand that it's not going to be perfect, and I'm okay with that. As long as the errors it makes are minor and infrequent, I'd be content.

Mike:
I think it would be very helpful if the AI could be quite accurate, especially for people who are visually impaired. If it's consistently wrong, it could lead to confusion or misunderstandings. That being said, I also understand that technology isn't perfect and there might be occasional errors.

2.2) (O) In an ideal scenario, how do you believe the app should handle uncertainties or ambiguities (for example, when the AI can't confidently determine a person's mood)?

Aiad:
I believe the app should handle uncertainties or ambiguities by providing a range or confidence level. For example, if the AI can't confidently determine a person's mood, it could offer a range such as "mostly happy with a 20% chance of neutrality". This communicates the uncertainty in a transparent and understandable way.

Mario:
If the app is unsure about something, I'd want it to be honest about it. Rather than making a guess that might be wrong, it could provide a range of possibilities or simply say it's unsure.

Mike:
In case the app is unsure about its prediction, it should probably state so. Maybe it could give a range or approximation, or suggest it's a best guess, just so that the user knows the information might not be completely accurate.

2.3) (C) Do you believe a confidence percentage indicating the AI's certainty of its assessment (e.g., 85% confidence that the person is happy) would be helpful?

Aiad:
Yes, I think a confidence percentage would be very helpful. It would provide transparency about the app's decision-making process and help users understand that AI predictions are probabilistic and not always 100% accurate.

Mario:
Confidence percentages sound like a cool feature. They would provide some context for the app's predictions and help me understand the AI's thought process. Plus, it would be fun to see if the AI's confidence aligns with my own assessments of people's age, gender, and mood.

Mike:
Yes, I think showing confidence percentages could be a good idea. It would help me understand how certain the app is about its predictions. But the app should explain clearly what these percentages mean, so I don't misunderstand.

3.1) (O) Can you describe a scenario where you would find this app most useful in your daily life?

Aiad:
As someone who works extensively with technology, I can see myself using this app in social or networking scenarios. For example, I might use it at a tech conference to gain quick insights about people I'm interacting with.

Mario:
I can see myself using this app in various social settings to make interactions more interesting and engaging. It could be a fun ice-breaker at parties or gatherings. Additionally, it could be helpful in understanding people's emotions better in certain situations, especially if I'm having trouble reading their expressions.

Mike:
To be honest, I might not use this app daily as my interaction with technology is limited. However, I can see it being beneficial in social gatherings where I might find it difficult to recognize faces or judge emotions due to my ageing eyesight.

3.2) (C) Do you anticipate using this app frequently, for example, daily, weekly, or only for special occasions?

Aiad:
Given the nature of my work and my interests, I can see myself using this app on special occasions or when meeting new people. It might not be an everyday tool for me, but it could provide valuable insights in specific scenarios.

Mario:
I'd probably use it quite often, especially when hanging out with my friends or when meeting new people. It could add an interesting dimension to social interactions.

Mike:
I could imagine using this app in family gatherings or when I meet with my old friends. It could also come in handy in situations where I'm interacting with new people and can't see their faces clearly.

3.3) (O) What specific features or functionalities would you want this app to have to best suit your personal needs and preferences?

Aiad:
As a developer, I would appreciate detailed insights from the AI's analysis. Apart from face and mood, the app could also infer an estimation of a person's ethnic heritage, age, and gender, based on their appearance and expressions. Additionally, I would like the app to have a feature that allows me to give feedback or corrections to the AI's predictions, thereby improving its learning process.

Mario:
Apart from the basic features of estimating faces and mood, it would be awesome if the app could also have a feature that allows me to share its predictions with others directly from the app.

Mike:
For me, the most useful features would be face and mood recognition. As my vision isn't what it used to be, these features could help me in social situations by giving me insights about people around me that I might not have been able to recognize myself.

4.1) (C) Would you prefer the option to adjust the speed and pitch of the text-to-speech feature to accommodate your listening comfort?

Aiad:
Yes, being able to adjust the speed and pitch of the text-to-speech feature would be highly beneficial. Everyone has their listening comfort level, and being able to personalise this would greatly improve the user experience. Also, people with hearing impairments might benefit from slower speeds or different pitches.

Mario:
Definitely! I think being able to adjust the speed and pitch of the text-to-speech would be a great feature. People listen and comprehend at different speeds, and some pitches might be more comfortable for some users than others. Having control over these settings would make the feature more user-friendly and personalised.

Mike:
I think the voice output should be slow enough for me to comprehend but not too slow that it feels unnatural. It would be helpful if the app allows me to adjust the speech speed according to my comfort.

4.2) (O) In what situations do you think using a text-to-speech feature could be most beneficial for you or other users of the app?

Aiad:
I think the text-to-speech feature could be most beneficial in situations where visual interaction with the app isn't possible or convenient. For instance, if I'm using the app in a crowded environment or while performing another task, having the app read out its analysis would be more practical than reading it on the screen.

Mario:
I think the text-to-speech feature could be very useful when I'm in a noisy environment and can't look at my screen, or when I'm multitasking and need my hands free. It could read out the app's predictions so I can keep up with what's happening without having to stop what I'm doing to look at the screen.

Mike:
I think using a text-to-speech feature could be very beneficial in situations where I'm trying to understand what's on the screen but have trouble seeing it without glasses. For example, if the app is displaying information about someone's age, gender, or mood, the text-to-speech feature would allow me to listen to this information instead of trying to read small text on the screen.

After the interviews, the following key points were established:

- There is a clear gap in the market for a mobile app that utilises artificial intelligence and augmented reality technologies to assist visually impaired individuals or any user seeking an innovative interaction method. Costly development and potential privacy concerns are often obstacles in creating such applications, particularly in a society where the respect for individual privacy is paramount. However, these challenges can be addressed with careful design and ethical considerations.
- The proposed app, while leveraging artificial intelligence for face analysis, does not aim to replace human perception entirely. Instead, it supplements human understanding and can be seen as a tool that works in tandem with human intuition, particularly for visually impaired users. While AI is growing in capabilities, it is still crucial to acknowledge the irreplaceability of human insight and understanding in interpreting emotional and contextual cues.
- The introduction of an app with these features could significantly enhance efficiency and accessibility for users. By quickly interpreting faces and generating information about the individual's age, gender, and mood, users can gain quick insights without relying heavily on their visual capacity. This 'time-saving' aspect is incredibly valuable for all users, particularly those who might struggle with visual interpretation.
- For visually impaired users, the app could potentially provide a level of independence and confidence in social situations, as they would be able to understand their surroundings better and perhaps anticipate social interactions. This positive impact can extend to a wider user base who might find innovative and practical uses for the app in their daily lives.
- The app's success depends on its ability to cater to a diverse user base with varying technical skills. The user interface should be simple and intuitive to ensure broad accessibility, even for users who are not well-acquainted with complex mobile applications.
- The app will use facial recognition to provide estimates of age, gender, and mood based on facial features. While the emphasis will be on these attributes, the app will not infringe upon personal identities or use names, ensuring that privacy is maintained.
- The application will leverage the live camera feed of the user's mobile device, allowing real-time analysis and providing immediate feedback. This feature enhances the app's usefulness in dynamic and social settings where understanding immediate context and interactions is key.

Features of the proposed solution

Feature to include	**Reasoning**	**Evidence**
A section of the screen dedicated to the live camera feed	This allows the user to see the real-time footage that is currently being analysed by the app, making the process transparent and the application more engaging.	● DeepFace ● Visage Technologies ● MorphCast ● Interview with Aiad Tarik ● Interview with Mario Prifti
A button that activates facial analysis	The use of a button to initiate the facial analysis process will simplify the user interface, making the app more user-friendly. It will also give the user control over when the analysis takes place, preserving battery life and data usage.	● DeepFace ● Interview with Aiad Tarik ● Interview with Mario Prifti
Text-to-speech button	A feature that allows the app to audibly deliver the results of the facial analysis. This is particularly useful for visually impaired users.	● Interview with Aiad Tarik ● Interview with Mike Parish
Dynamic labels showing estimated number of people and mood and sliders.	Labels and sliders that appear on the screen to indicate the estimated number of people and the moods of the people in view. This provides a quick visual summary of the analysis results.	● Visage Technologies ● MorphCast ● Interview with Aiad Tarik ● Interview with Mario Prifti
Integrated tutorials or guide	Integrated tutorials will help new or less tech-confident users to understand how to navigate and utilise the app effectively. This feature ensures the app's usability across a broad range of users.	● Interview with Aiad Tarik ● Interview with Mike Parish
The visual outline of detected faces	To indicate the target of the analysis and confirm that a face has been detected, an outline around the detected face will be displayed. This feature enhances user understanding and allows them to align the face properly with the camera.	● DeepFace ● MorphCast ● Interview with Aiad Tarik ● Interview with Mario Prifti ● Interview with Mike Parish
User feedback system	A system for users to provide feedback about the app's performance, accuracy, or any potential issues will be included. This will help in the app's continuous improvement and evolution.	● MorphCast ● Interview with Aiad Tarik ● Interview with Mario Prifti ● Interview with Mike Parish

Limitations of features from researched solutions

Feature	Evidence	Limitation
Dyanmic Labels	● Visage Technologies ● Morphcast	Visage Technologies and morphcast utilise server-side processing, where they are able to update labels and sliders in real-time. Since my application is on a mobile phone, I will mainly be focusing on local processing. As a result, the inclusion of real-time updating dynamic labels will look laggy and affect performance, due to the little processing power that an mobile phone has.
User feedback system	● Morphcast ● Stakeholder Interviews	In order to implement a user feedback system, I will need to incorporate a lot of back-end code thats sends a message to a server that I have access to, which may be costly and not fit within time-constraints. Furthermore, without proper guidance or structured feedback mechanisms, users may provide vague or unhelpful feedback, making it difficult to extract actionable insights from the data, which results in a feedback system being too unreasonable to implement.
Integrated tutorial/Guide	● Stakeholder Interviews	If the tutorial is only available in one language, it may exclude users who are not proficient in that language, limiting the app's accessibility and usability for a diverse user base. Also, Some users may prefer to explore the app on their own rather than following a guided tutorial, and forcing them to go through the tutorial may lead to resistance or negative perceptions of the app. Furthermore, another limitation of implementing a tutorial, can cause maintenance issues such as having to update and maintain the tutorial to reflect changes in the app or new features. This can be time-consuming and resource-intensive, especially for a small project.

Software and hardware requirements

For development:
These requirements are based on the prerequisites for the primary development tools and platforms, namely Unity, ARKit plugin, OpenCV for Unity package, and C#.

System Requirement	Justification
macOS 10.15 or newer	The operating system required to run the latest version of Xcode and Unity without compatibility issues.
2.4 GHz processor (quad-core)	This is needed because it is the minimum processor speed required to run Unity and Xcode efficiently.
8-16 GB of RAM	This is needed because it is the minimum amount of RAM needed to run Unity, Xcode, and to test the application within the development environment.
Hard disk space up to 100 GB	This is needed because it should be enough storage to account for the code, libraries, and datasets.
Unity 2022.1.1f1	This is the platform on which the app's code will be written and executed.
ARKit Unity Plugin	This plugin is necessary for building AR applications within Unity.
OpenCV for Unity Package	This package allows OpenCV functions to be called in Unity, which will facilitate the implementation of the app's machine learning models.
C#	This is the programming language that will be used to write the app's code within Unity.
Xcode 12 or newer	This is needed to compile and deploy the app to an iPhone.

For the user:

System Requirement	Justification
iPhone with iOS 13.0 or newer	The app will be built to function on iOS devices, utilising ARKit and OpenCV for Unity.
Stable Internet Connection	This is needed because the app may need to access online resources or updates.
Rear Camera	This is essential for the AR capabilities of the application.
Speakers	These are needed in order for the user to hear the text-to-speech output produced by the app.

Success criteria

The development of this mobile app will be guided by the following standards. On a scale of 1 to 3, the relevance of each characteristic will be graded. The number 1 signifies a high priority, whereas the number 3 implies a lower priority. My primary attention will be on the high- and moderate-importance criteria (1 and 2). Based on the time and knowledge restrictions, low priority features will be developed as pleasant extras that are not essential to the system's operation.

ID	Feature	Justification	Priority
1	Camera functionality when user opens app	The camera feed is fundamental for the app to function as it provides the visual input for all processing. Without it, none of the following features can be achieved.	1
2	User interface design	The user interface should be simplistic and intuitive as it can directly affect the user experience. It should be designed with visually impaired users in mind.	2
3	Identification of people in camera's view/facial recognition	This is crucial for the application as it sets the basis for further analysis such as facial recognition and mood detection	1
4	Object detection	Object detection may be added as a separate section of the program where the user can point their camera and view the various objects in their field of vision and surroundings, however this feature is not necessary to the main function of the program.	3
5	Home screen	The home screen for the application should be welcoming and also clear. Because this is the first scene that the user is going to see and as a result, should set off a positive impression on the application.	2
6	Mood recognition based on facial expressions	Identifying the mood of the person in view can help visually impaired users understand non-verbal cues and respond appropriately.	1
7	Text-to-speech output for processed attributes	This feature enables the application to relay the recognized information (gender, age, and mood) to the user orally. It caters to the needs of the visually impaired users who might not be able to see the results displayed on the screen.	3
8	Image capture and captioning	The app should allow the user to capture an image	3

		and generate a basic caption of the image’s content. This aids in creating a mental visualisation of the surroundings for the user.	
9	Responsiveness (20fps consistent feed)	The application should operate with minimal lag between capturing an image and generating results. A quick response time is critical for the user's experience. Due to the time limits and computational bottlenecks, I may result in single frame captures where the inputs are processed slower and outputted afterwards (almost like taking a photo and analysing what’s inside it)	2
10	Accuracy	The output (age, gender, mood, image caption) should be as accurate as possible to provide the user with reliable information about their surroundings. Although due to the time pressure and computational power required, I may not be able to get very accurate results all of the time.	2
11	Ease of use	Considering the user may not have much experience with technology, the application should be simple to navigate. All features should be readily accessible, and instructions should be clear and concise.	1
12	Variable text-to-speech voice pitch/volume/speed	The application should allow users to customise the pitch, volume, and speed of the text-to-speech voice output. This can help users to understand the audio output more comfortably and according to their preference.	1

Design

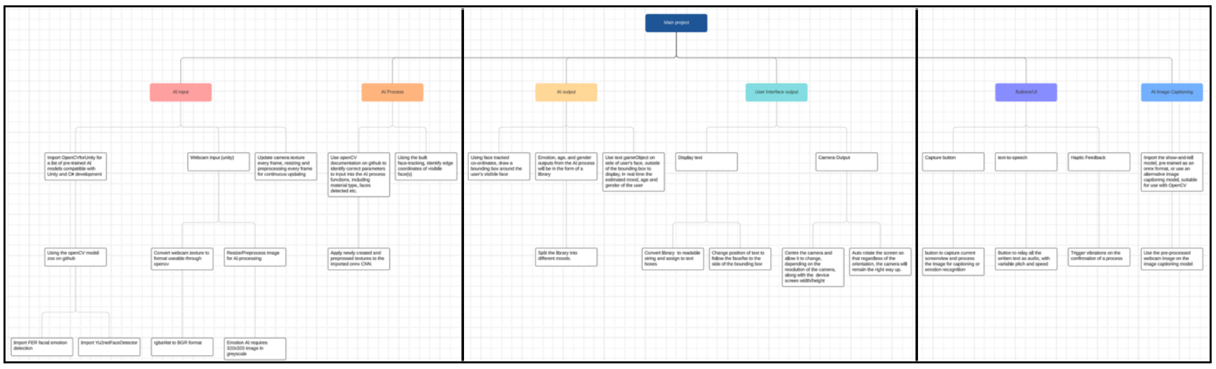
Overview of the system (top-down design/systems diagram)

The diagram below shows a top-down breakdown of my planned project and its implementations, thus allowing me to follow a guide when developing my iterations and visually view which parts will be complete, which will be re-useable, and which are completed/left to do.

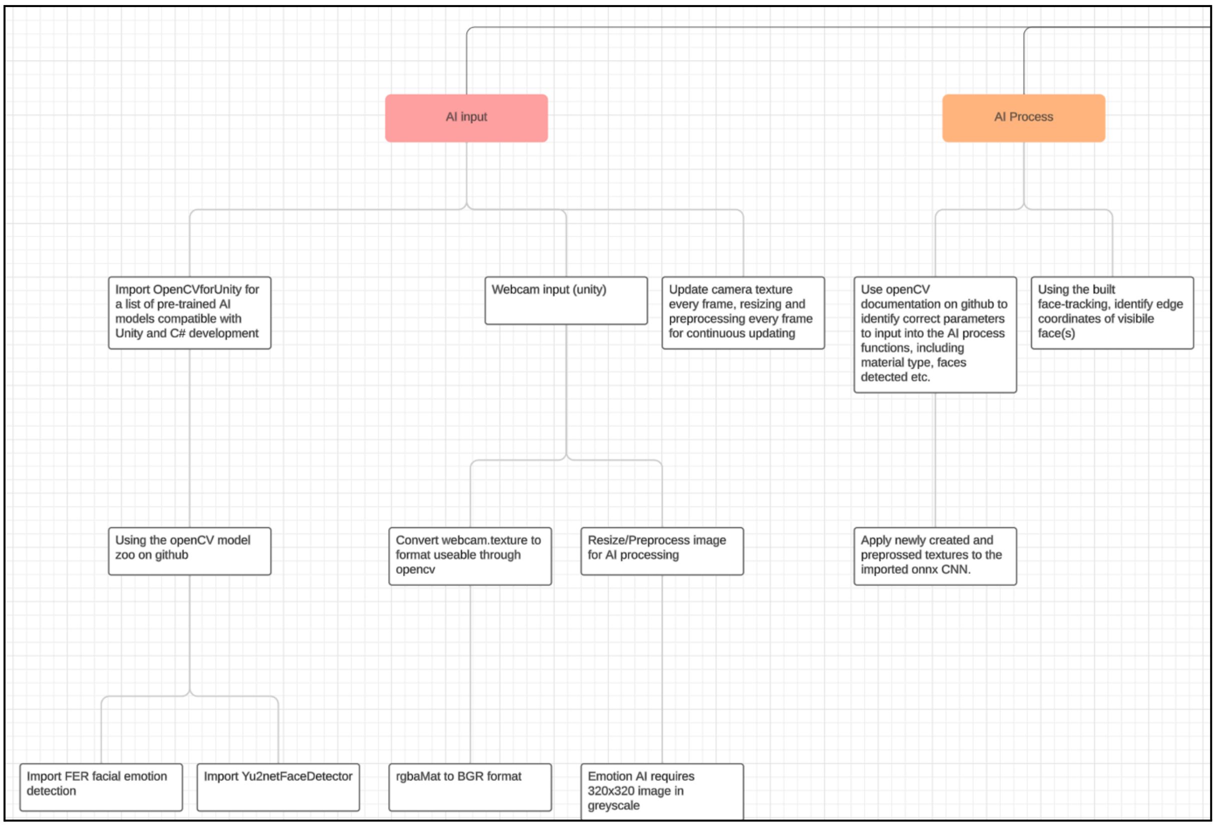
Level 1 of the top-down design contains all of the primary functions of the project such as AI input, AI processing, Output, UI Output, app UI, and Image Captioning. The image captioning and app UI sections of level 1 are going to be developed in the later iterations as they do not serve a critical purpose in my program.

Level 2 breaks down the functions further into separate processes and problem-solving to help tackle any obstacles that may come in the way, such as pre-processing images and outputting results in the correct format. This will allow for more manageable development of the program through the use of sub-functions.

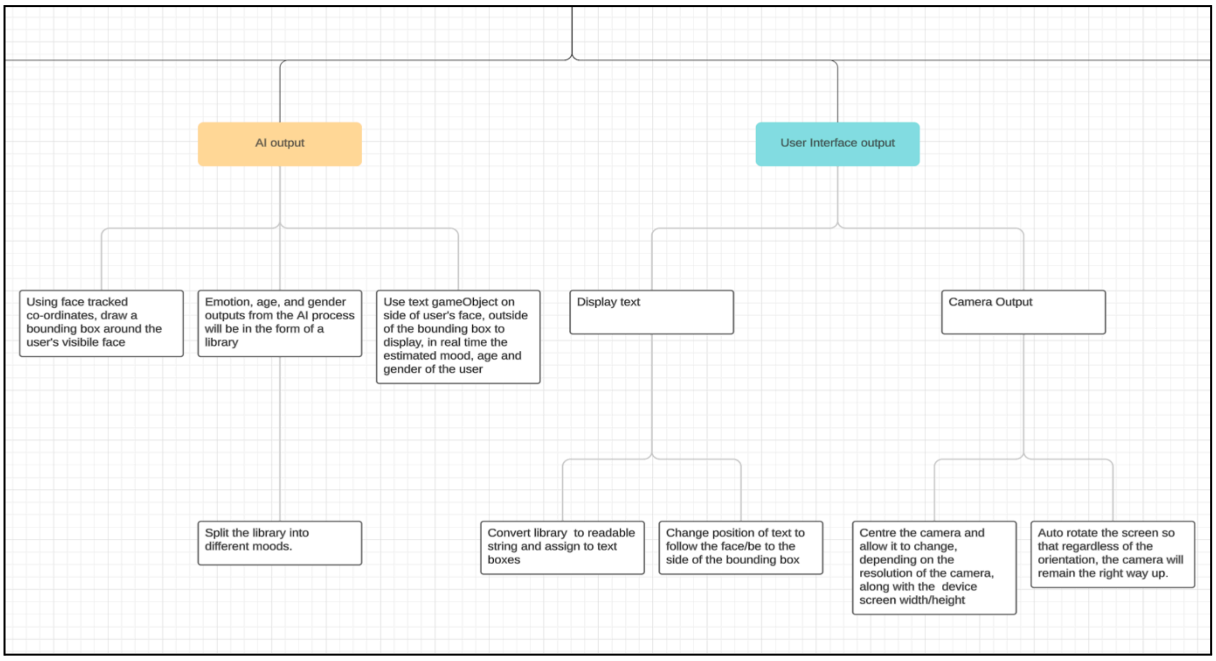
Finally, Level 3 and 4 goes into further detail about the specific processes used to attain the level 2 sub-functions, such as which models to use, where to import from, specific formatting, and splitting data types etc. These end levels will ensure that the level 2 functions are met and can be executed perfectly.


Main project
AI input
AI Process
AI output
User Interface output
Buttons/UI
AI Image Captioning
Import OpenCVforUnity for a list of pre-trained AI models compatible with Unity and C# development
Webcam input (unity)
Update camera texture every frame, resizing and preprocessing every frame for continuous updating
Use openCV documentation on github to identify correct parameters to input into the AI process functions, including material type, faces detected etc.
Using the built face-tracking, identify edge coordinates of visible face(s)
Using face tracked co-ordinates, draw a bounding box around the user's visible face
Emotion, age, and gender outputs from the AI process will be in the form of a library
Use text gameObject on side of user's face, outside of the bounding box to display, in real time the estimated mood, age and gender of the user
Display text
Camera Output
Capture button
text-to-speech
Haptic Feedback
Import the show-and-tell model, pre-trained as an onnx format, or use an alternative image captioning model, suitable for use with OpenCV
Using the openCV model zoo on github
Convert webcam texture to format useable through opencv
Resize/Preprocess image for AI processing
Apply newly created and preprossed textures to the imported onnx CNN.
Split the library into different moods.
Convert library to readable string and assign to text boxes
Change position of text to follow the face/be to the side of the bounding box
Centre the camera and allow it to change, depending on the resolution of the camera, along with the device screen width/height
Auto rotate the screen so that regardless of the orientation, the camera will remain the right way up.
button to capture current screenview and process the image for captioning or emotion recognition
Button to relay all the written text as audio, with variable pitch and speed
Trigger vibrations on the confirmation of a process
Use the pre-processed webcam image on the image captioning model
Import FER facial emotion detection
Import Yu2netFaceDetector
rgbaMat to BGR format
Emotion AI requires 320x320 image in greyscale

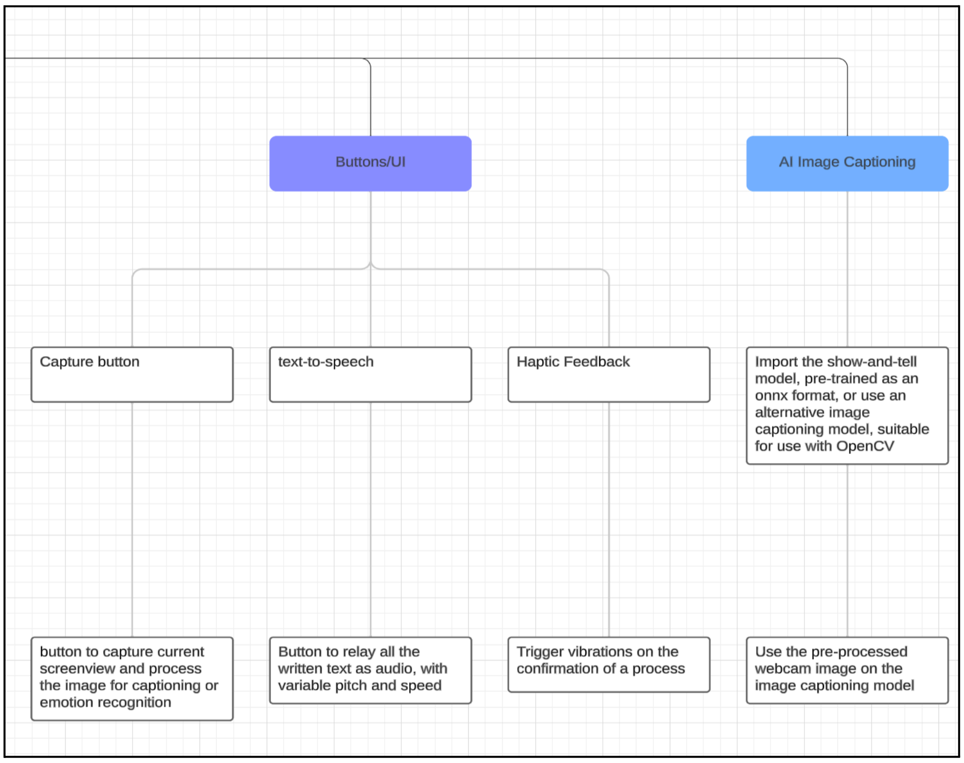
Section 1:


AI input
AI Process
Import OpenCVforUnity for a list of pre-trained AI models compatible with Unity and C# development
Webcam input (unity)
Update camera texture every frame, resizing and preprocessing every frame for continuous updating
Use openCV documentation on github to identify correct parameters to input into the AI process functions, including material type, faces detected etc.
Using the built face-tracking, identify edge coordinates of visibile face(s)
Using the openCV model zoo on github
Convert webcam.texture to format useable through opencv
Resize/Preprocess image for AI processing
Apply newly created and preprossed textures to the imported onnx CNN.
Import FER facial emotion detection
Import Yu2netFaceDetector
rgbaMat to BGR format
Emotion AI requires 320x320 image in greyscale

Section 2:


AI output
User Interface output
Using face tracked co-ordinates, draw a bounding box around the user's visibile face
Emotion, age, and gender outputs from the AI process will be in the form of a library
Use text gameObject on side of user's face, outside of the bounding box to display, in real time the estimated mood, age and gender of the user
Display text
Camera Output
Split the library into different moods.
Convert library to readable string and assign to text boxes
Change position of text to follow the face/be to the side of the bounding box
Centre the camera and allow it to change, depending on the resolution of the camera, along with the device screen width/height
Auto rotate the screen so that regardless of the orientation, the camera will remain the right way up.

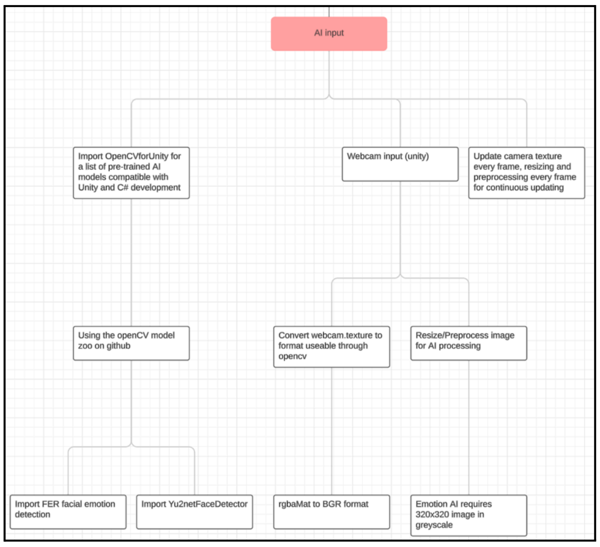
Section 3:


Buttons/UI
AI Image Captioning
Capture button
text-to-speech
Haptic Feedback
Import the show-and-tell model, pre-trained as an onnx format, or use an alternative image captioning model, suitable for use with OpenCV
button to capture current screenview and process the image for captioning or emotion recognition
Button to relay all the written text as audio, with variable pitch and speed
Trigger vibrations on the confirmation of a process
Use the pre-processed webcam image on the image captioning model

Module Descriptions

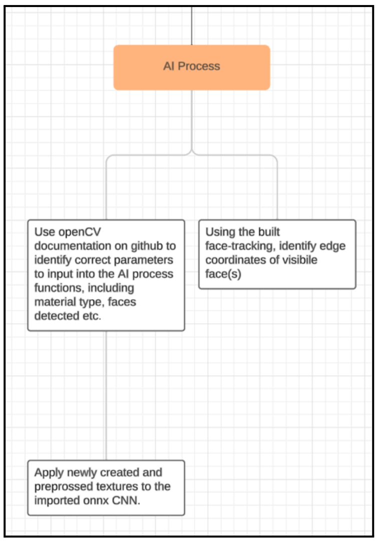
AI input

This section is the most important section in the whole project. The AI input section will manage taking an input and pre-processing the image to work with the computer vision models. This includes importing the pre-trained models, along with their execution scripts, and more importantly, the ability to instantiate a webcam object for the image to be processed in real-time through the update() method.


AI input
Import OpenCVforUnity for a list of pre-trained AI models compatible with Unity and C# development
Webcam input (unity)
Update camera texture every frame, resizing and preprocessing every frame for continuous updating
Using the openCV model zoo on github
Convert webcam.texture to format useable through opencv
Resize/Preprocess image for AI processing
Import FER facial emotion detection
Import Yu2netFaceDetector
rgbaMat to BGR format
Emotion AI requires 320x320 image in greyscale

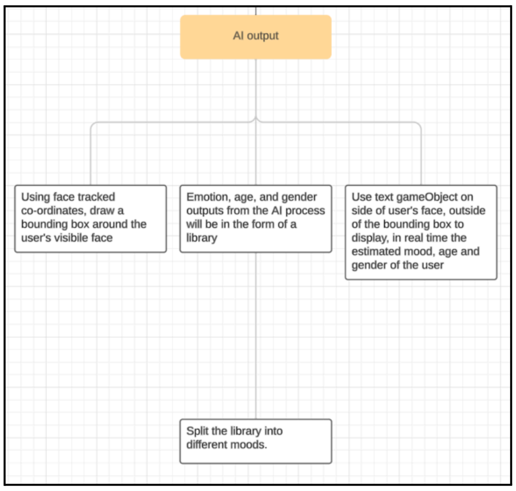
AI Process

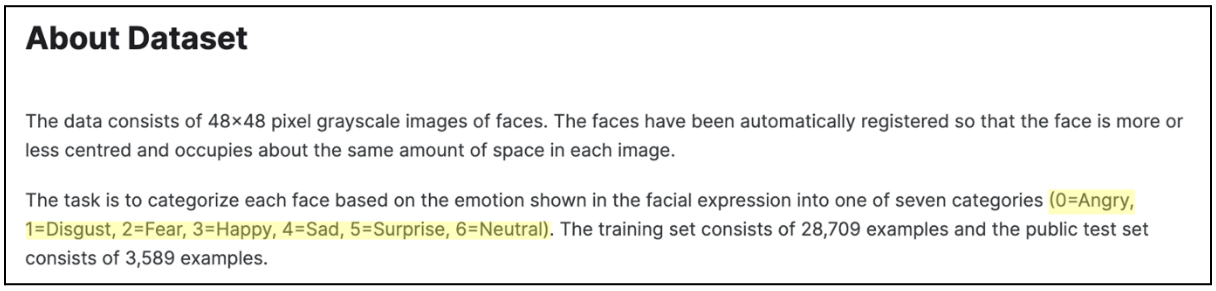
This section details the processing of images for the models, which will include converting the webcam texture to the correct unity material type to allow it to be displayed on screen and then executed. In addition to this, the face-tracking model will be responsible for identifying the co-ordinates of the corners of the detected faces, and draw a bounding box around them, allowing for easy visualisation.


AI Process
Use openCV documentation on github to identify correct parameters to input into the AI process functions, including material type, faces detected etc.
Using the built face-tracking, identify edge coordinates of visibile face(s)
Apply newly created and preprossed textures to the imported onnx CNN.

AI Output

This section manages the output of results from the AI models. After processing the inputted data from the webcam, the program will then convert these results to a type that is recognisable by Unity to be displayed on screen. This includes the drawing of bounding boxes, converting a json output to a string, and splitting outputs (0, 1, 2, 3, 4, 5, 6) into their respective moods (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral)

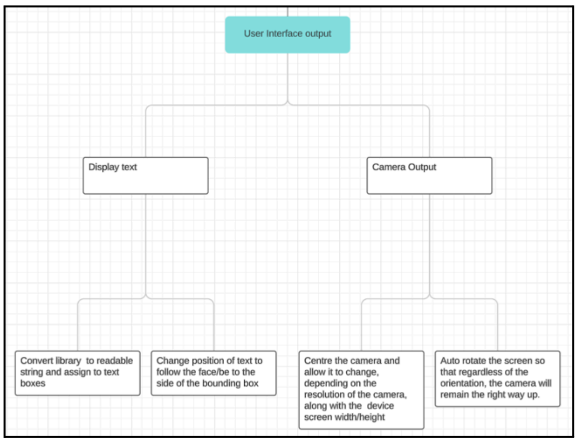

AI output
Using face tracked co-ordinates, draw a bounding box around the user's visibile face
Emotion, age, and gender outputs from the AI process will be in the form of a library
Use text gameObject on side of user's face, outside of the bounding box to display, in real time the estimated mood, age and gender of the user
Split the library into different moods.



About Dataset

The data consists of 48×48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centred and occupies about the same amount of space in each image.

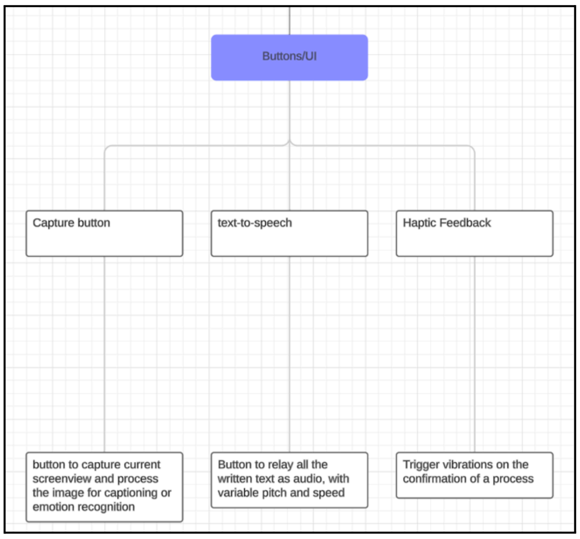
The task is to categorize each face based on the emotion shown in the facial expression into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). The training set consists of 28,709 examples and the public test set consists of 3,589 examples.


User Interface output
Display text
Camera Output
Convert library to readable string and assign to text boxes
Change position of text to follow the face/be to the side of the bounding box
Centre the camera and allow it to change, depending on the resolution of the camera, along with the device screen width/height
Auto rotate the screen so that regardless of the orientation, the camera will remain the right way up.

UI Output

The following section details the output to be set to the user interface. This includes the text, video output, and the text-to-speech audio output. The subroutines include:

- Screen auto-rotation
- Centring the camera
- Text position

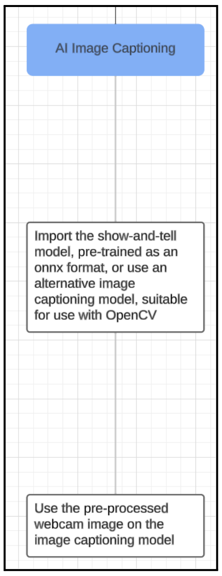

Buttons/UI
Capture button
text-to-speech
Haptic Feedback
button to capture current screenview and process the image for captioning or emotion recognition
Button to relay all the written text as audio, with variable pitch and speed
Trigger vibrations on the confirmation of a process

Buttons/UI

This outlines the general interface that the user will experience when using the app – this includes any buttons, perhaps haptic feedback functionality, landing page and perhaps any other scene's/settings button

AI Image Captioning

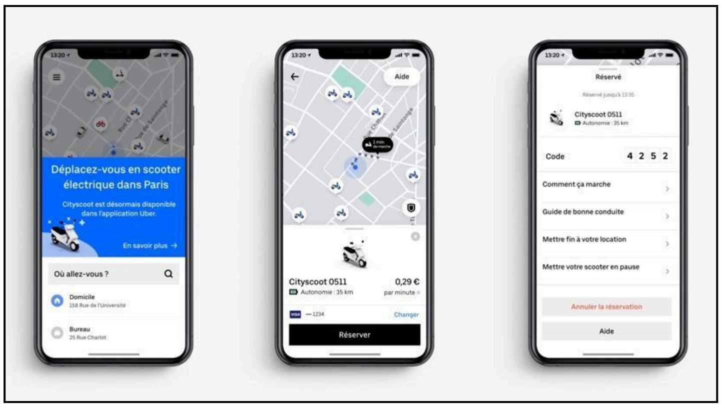
The AI Image Captioning section of this project is perhaps the most ambitious – This is due to the large amount of processing and data moving. Unity would perhaps not be able to be equipped to handle the task of captioning real-life images and as a result, the handling of this task would most likely be done on an external python flask server and the results would be sent back to unity. This section will require deeper planning and its execution will most likely be in the second or third iteration, since it may not be necessary for the program.


AI Image Captioning
Import the show-and-tell model, pre-trained as an onnx format, or use an alternative image captioning model, suitable for use with OpenCV
Use the pre-processed webcam image on the image captioning model

Proposed Screen Designs and usability features

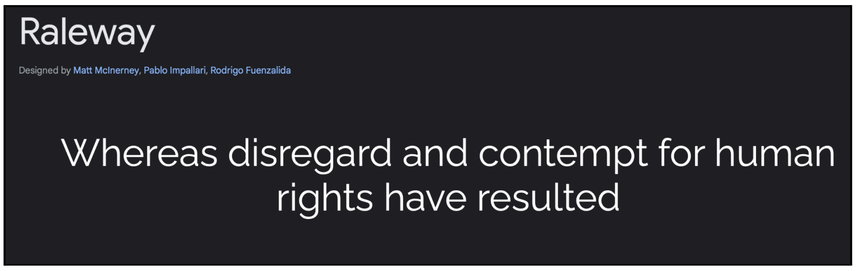
Design style

I will be designing my app using a minimalistic aesthetic, similar to that of Uber, which follows a basic, black and white colour scheme, along with rounded buttons and UI, pleasing to the eye and easy to follow, for people of all levels of tech experience – this is particularly useful as the app will be used by many different demographics and thus should be a universally understandable interface.



Font

I will be using the "Raleway" font throughout this project. This font caught my eye due to its level of minimalism and easy readability. This is a free, open-source font available online which is useful for my project, as it aims to be built on open-source functionalities.



Raleway

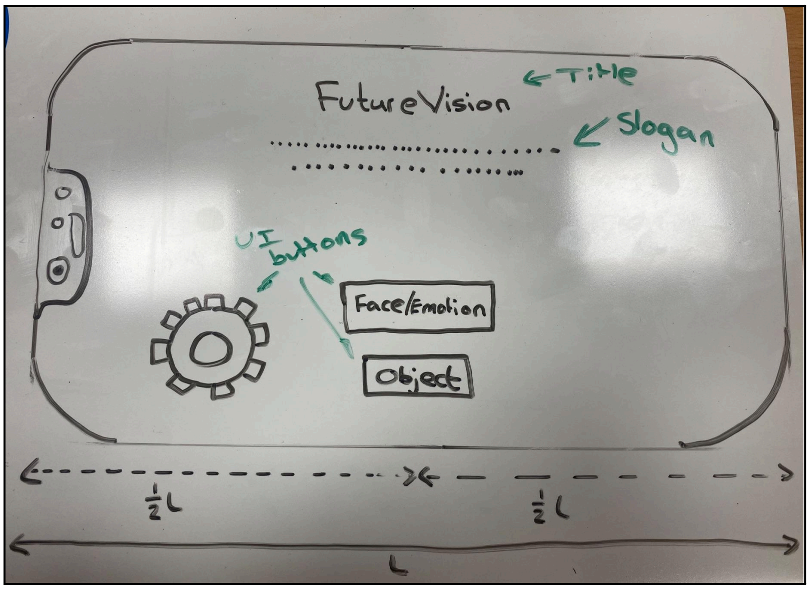
Designed by Matt McInerney, Pablo Impallari, Rodrigo Fuenzalida

Whereas disregard and contempt for human rights have resulted

Colours

I want my application to be easy-to-follow, while following an appealing aesthetic. This will include a high-contrast, aesthetic colour scheme, such as dark grey/black, white/off-white and light blue.

Main menu



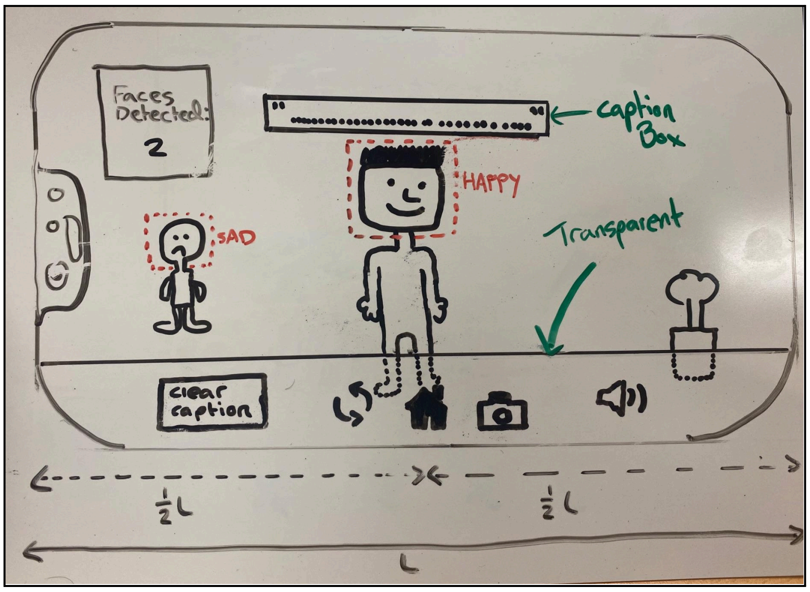
When starting the application, the user should be greeted with a pleasant main menu, directing them to appropriate functions/scenes, such as object detection, emotion detection, image captioning etc…

The main menu should be welcoming and responsive, with an emphasis on accessibility, since my project is surrounded around this.

The home screen of an app serves as the entry point for user interaction and sets the tone for their experience. An accessible and eye-catching home screen is essential for several reasons:

1) **First Impressions Matter**: The home screen is the first thing users encounter, so it should make a positive impact. An eye-catching design can captivate users, encourage engagement, and reflect the app’s purpose. The home screen, with its clean, modern typography and a slogan that suggests innovation and forward-thinking, can attract users interested in the app.
2) **Accessibility Ensures Inclusivity**: Accessibility is vital for providing all users, including those with disabilities, the ability to use the app effectively. By making the home screen accessible, I can ensure that the app is usable for a wider audience, which not only broadens the user base but also demonstrates social responsibility. The clear contrast between the elements on the home screen and the use of large, readable text is a good start toward this inclusivity.
3) **Clear Navigation**: A well-designed home screen guides users intuitively to the next steps. The use of clearly labeled buttons such as "FACE/EMOTION" and "OBJECT" helps users understand what actions they can take, reducing confusion and improving the overall experience.

Emotion Detection Scene



This will be the main scene of the application. When opening this scene, the user should be greeted with a highly user-friendly interface and basic layout. There shouldn't be any unnecessary buttons and text on the screen. It will be focused on the camera output, and any UI that will benefit the user by allowing them to interact with the scene through the various functions such as image captioning, Text-to-speech and more.

Face and Emotion Detection:

The upper-left corner of the interface features a notification area that indicates the number of faces detected. This immediate feedback is for users to understand that the app is actively processing the scene in real time.

Each detected face is highlighted and labeled with the identified emotion. This direct visual cue ensures users know the AI's analysis outcome without searching through menus or text, which enhances the interactive experience.

Caption Box and Transparency:

The caption box, located at the top, is designed to be transparent to minimize obstruction of the camera's view. This design choice respects the user's primary need to view the camera feed while simultaneously providing contextual information.

When the user takes a picture, the AI-generated caption will appear in this box, providing an unobtrusive yet clear narrative of the image.

Camera and Captioning Function:

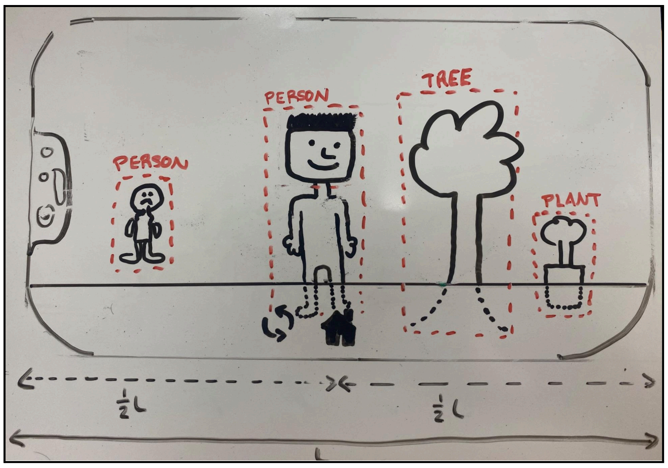
A camera icon serves as the central action button, intuitively understood as the trigger for image capture and subsequent captioning. This familiar iconography reduces the learning curve for new users.

The "Clear caption" feature is a thoughtful addition, allowing users to remove the caption if they desire a clear view or wish to retake the photo without previous context lingering on the screen.

Text-to-Speech:

Along the bottom of the screen, the text-to-speech function is represented by a speaker icon. This feature is essential for accessibility, allowing users with visual impairments to hear the captions and emotional analysis.

Object Classification Scene



The object classification screen is an extra feature that I will implement, due to the vast amount of readily available pre-trained models such as YoLo (You only look once). This will not have any excess UI or TTS, rather it would simply be a 'playground' of some sorts that the user can play around with, to test the functionality and capabilities of AI.

This scene emphasizes clarity and direct interaction with the environment through the camera. Here's an in-depth analysis of this scene's design:

Object Detection Interface:

The scene prominently displays the object detection feature by highlighting objects with dotted outlines and labeling them. This instant visual feedback is crucial for the user to understand what the app is identifying in real-time, providing an interactive and informative experience.

Simplified Functionality:

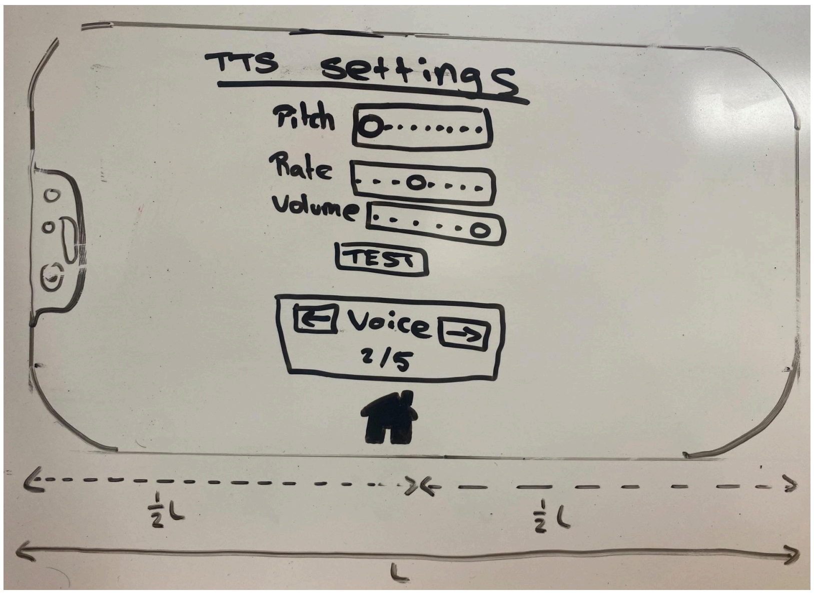
In contrast to the main scene, this scene is even more simplified, with only the essential functionalities present: the home button and the flip camera button. This decluttering aligns with the scene's purpose, which is solely to identify objects, and avoids overwhelming the user with options that are not required for the task.

Direct User Engagement:

By focusing on the camera feed and the identified objects, the scene encourages users to engage directly with their environment. This kind of interaction can be particularly appealing to users interested in learning about their surroundings or needing to quickly identify objects.

This additional scene's design is effective in its simplicity and functionality. It aligns with the app's overall theme of using technology to interact with and understand the environment

Settings Page



The settings scene for the app is a critical component that offers personalization and control over the Text-to-Speech (TTS) functionality. This scene should be intuitive, easy to navigate, and provide users with clear feedback.

The key elements for the design of the settings page are as follows:

TTS Customization Options:

Pitch, Rate, and Volume Sliders: These controls allow users to customize the TTS output to their preference. Sliders are an excellent choice for these settings, as they offer granular control and are easily adjusted with touch input.

Test Button: A ‘Test’ button is present to enable users to immediately hear the effects of their adjustments. This instant feedback is essential for fine-tuning the TTS settings without leaving the settings scene.

Voice Selection:

Voice Carousel: I have included a carousel for voice selection, indicated by the "Voice" box with arrows to navigate between different options. This lets users sample and choose from multiple TTS voices, enhancing the user experience through customization. This is because iOS’s built-in TTS package has many different voices to choose from, so perhaps I will try to allow for the user to cycle through them

Confidence Threshold:

While not visually represented in the provided image, the confidence threshold is a sophisticated feature that allows users to set the level of certainty the AI needs before making a decision or providing output, which can reduce false positives or ensure only high-confidence information is spoken.

Navigation:

Similar to the other scenes, the bottom navigation buttons allows for easy movement between different parts of the app. This consistency in navigation design helps users quickly learn how to move around the app.

Useability Features

To make the AI app useful and accessible to everyone, especially those who are visually impaired, it's crucial to focus on several key principles. The application must be effective, efficient, engaging, error-tolerant, and easy to learn. These qualities are essential for developing a robust solution that provides a positive user experience, encouraging continued use of the app for its intended purpose.

Effectiveness

The effectiveness of an application is measured by its ability to help users accomplish their intended tasks with accuracy and reliability. In the case of my AI face detection software, effectiveness is paramount, as it directly impacts the user's ability to understand their surroundings through audio descriptions of people and objects captured by their iPhone's cameras. By leveraging advanced AI algorithms, the app accurately identifies faces and objects in the camera's view, providing detailed captions that are then converted into spoken words through text-to-speech technology. This process ensures that users, regardless of their visual capabilities or technical expertise, can achieve their goal of comprehending their environment more fully. To cater to our diverse target audience, from those with visual impairments to tech enthusiasts, the software is designed with simplicity and precision in mind. The application's effectiveness is further enhanced by its ability to adapt to various lighting conditions and environments, ensuring reliable performance in real-world use. This commitment to effective design and functionality makes our AI face detection and captioning software a powerful tool for enhancing accessibility and understanding, empowering users to engage more confidently with the world around them.

The core of its effectiveness lies in the sophisticated AI algorithms that drive the software. These algorithms are trained on a vast dataset of images, enabling the software to recognize faces and objects with high precision across different environments, lighting conditions, and angles. This breadth of recognition capabilities ensures that users receive accurate and useful descriptions of their surroundings, regardless of the complexity of the scene in front of them.

The software's effectiveness is further amplified by its seamless integration with the iPhone's hardware, utilizing both the rear and front cameras to capture comprehensive views of the user's environment. This dual-camera approach allows for a versatile application, whether the user is navigating new spaces or engaging in social interactions, the app provides audible feedback about what is in their immediate vicinity.

Moreover, the application's text-to-speech (TTS) feature is a critical component that enhances its effectiveness. By converting visual information into audio, it bridges the gap between the digital and physical worlds for users with visual impairments. The TTS technology isn't just a straightforward narration tool; it's designed with customization in mind. Users can adjust the voice settings, including pitch, rate, volume, and voice type, allowing for a personalized experience that caters to individual preferences and needs. This level of personalization not only makes the software more accessible but also more engaging, encouraging users to rely on it in their daily lives.

Efficient

Efficiency in this context encompasses several dimensions, starting with the application's processing speed. The software is engineered to perform face and object detection swiftly, minimizing the delay between capturing an image and providing audio feedback, while also maintaining a relatively smooth 20-30fps. This rapid processing ensures that users receive real-time information about their environment, a critical feature for those relying on the app to navigate unfamiliar spaces or interact with nearby objects and individuals. Fast response times are achieved through the optimization of AI algorithms, which are designed to work effectively within the computational constraints of mobile devices without compromising accuracy.

Moreover, the app's efficiency is evident in its minimal impact on the device's battery life. By leveraging the iPhone's hardware efficiently, the software minimizes energy consumption, enabling users to rely on the app for extended periods without concern for rapid battery depletion. This consideration is paramount for users who depend on the app throughout their daily activities, ensuring that the technology supports their needs without introducing additional constraints.

The software also demonstrates efficiency in data usage. By processing images directly on the device, it reduces the need for constant data transfers to cloud servers, which not only speeds up recognition tasks but also addresses privacy concerns. Local processing ensures that users can access the app's features even in areas with limited or no internet connectivity, enhancing the app's utility across various scenarios.

Furthermore, the app's user interface contributes to its overall efficiency. By employing a design that is both simple and intuitive, users can navigate the app with minimal effort, reducing the cognitive load and making the technology accessible to individuals with varying levels of tech savviness. This ease of use is critical for ensuring that the app can be quickly adopted and integrated into the user's daily routine, maximizing its effectiveness as a tool for enhancing independence and quality of life.

Engaging

The engaging nature of the software is rooted in its interactive design. By utilizing the iPhone's front and rear cameras, the app invites users to explore their surroundings actively, turning what could be a routine activity into an interactive experience. This exploration is further enhanced by the software’s ability to provide detailed, real-time descriptions of people and objects, transforming the mundane into opportunities for discovery and learning. Such features make the app not just a tool but a companion for daily adventures, encouraging users to engage more deeply with their environment.

Personalization plays a crucial role in making the software engaging. The ability for users to adjust text-to-speech settings, including pitch, rate, volume, and voice type, allows them to tailor the auditory feedback to their preferences, making the experience more comfortable and enjoyable. This level of customization ensures that the app speaks to the user in a voice that feels familiar and friendly, fostering a stronger connection between the user and the technology.

The app's simple and easy-to-follow user interface (UI) enhances engagement by reducing friction in the user experience. By prioritizing intuitive design and basic icons while minimizing unnecessary text, the app ensures that users of all tech-savviness levels can navigate it effortlessly. This ease of use encourages frequent use, as users are not deterred by a complex or confusing UI. The straightforward design philosophy not only makes the app accessible but also more inviting to users, encouraging them to explore its features and capabilities fully.

Additionally, the software’s engaging character is amplified by its educational aspect. Users are not only informed about their immediate surroundings but also have the opportunity to learn about different objects and facial recognition technology. This educational value adds a layer of depth to the user experience, making the app not just a functional tool but also a source of knowledge and personal growth.

Error-Tolerant

The software's error tolerance is significantly enhanced by the inclusion of a user-adjustable minimum confidence level for detections. This setting empowers users to customize the balance between accuracy and the quantity of information provided. By setting a higher confidence threshold, users can reduce the likelihood of receiving incorrect identifications, tailoring the app's performance to prioritize precision. Conversely, users willing to accept a broader range of detections, potentially with a higher risk of errors, can lower this threshold according to their preferences or needs. This level of customization is particularly beneficial in diverse environments where the challenge of accurate detection varies, allowing users to adapt the app's sensitivity to their current situation.

This approach acknowledges the inherent challenges in face and object detection technologies, where varying conditions can affect the AI's performance. By giving users control over the confidence level, the app not only enhances its usability under different circumstances but also instills a sense of agency, allowing individuals to dictate how they interact with the technology based on their comfort level with its accuracy.

Furthermore, this feature demonstrates the software's commitment to error tolerance without directly exposing users to the complexities of its AI algorithms. Instead of burdening users with technical feedback on the nature of errors, the application simplifies error management, focusing on delivering the most reliable results possible. This decision respects the user's desire for a straightforward, effective tool, avoiding unnecessary confusion or frustration that could arise from technical explanations of AI limitations.

Moreover, the settings page, where users can adjust the confidence level, is designed with clarity and accessibility in mind. This ensures that even those not technically savvy can understand and control this feature, enhancing the overall user experience. By seamlessly integrating this setting into the app's intuitive interface, the software maintains its user-friendly appeal while offering sophisticated customization options.

Easy to Learn

The cornerstone of the software's easy-to-learn nature is its intuitive user interface (UI). The UI is designed with simplicity in mind, featuring clear, large icons and a minimalistic layout that reduces cognitive load and visual clutter. This design choice ensures that users can navigate the app without feeling overwhelmed, making it particularly accommodating for those who are not tech-savvy or may have visual impairments. By employing universally recognizable symbols and organizing features in a logical, straightforward manner, the app minimizes the learning curve, allowing users to become proficient in its use quickly.

Another aspect that contributes to the software's ease of learning is its consistency across different sections. Consistency in design and interaction patterns reassures users, as the skills learned in one part of the app are transferable to others. This coherence in user experience reduces confusion and builds user confidence, as they know what to expect and how to interact with the app as they explore its various features.

The software also includes an advanced settings page, which, while offering advanced options like adjusting the minimum confidence level for detections and personalizing text-to-speech settings, is designed to be straightforward and easy to understand. Explanations for each setting are clear and concise, guiding users through the customization process without overwhelming them with technical jargon or excessive options. This balance between customization and simplicity ensures that users can tailor the app to their needs without a steep learning curve.

To further facilitate easy learning, the app incorporates onboarding tutorials at the initial launch, offering users a guided tour of its features and how to use them effectively. These tutorials are designed to be engaging and informative, providing just enough information to get started without inundating users with details.

Algorithm Designs

CaptureCamera()	Description

```
Texture2D texture;
WebcamTexture WebCamTexture;
private int currentCameraIndex = -1;

static string FER_model_filepath = "OpenCV/FER.onnx";
static string face_detection_model_filepath = "OpenCV/FaceDetect.onnx";

do {
        currentCameraIndex++;
        if (currentCameraIndex >= WebCamTexture.devices.Length)
         {
             currentCameraIndex = 0;
         }
   } while (WebCamTexture.devices[currentCameraIndex] == UltraWideAngle)


webCamTexture = (WebCamTexture.devices[currentCameraIndex].name);

// Start the camera
webCamTexture.Play();

texture =Texture2D(webCamTexture.width, webCamTexture.height);
gameObject.GetComponent<Renderer>().material.mainTexture = texture;

faceDetector = new FaceDetectorClass(face_detection_model_filepath);
FER = new FacialExpressionRecognizerClass(FER_model_filepath);
```
	This algorithm operates by initializing a new blank texture and scanning for a camera that isn't categorized as UltraWideAngle, due to its potential to distort the visual field, which is critical for precise object and face detection. It systematically increases the camera index until it locates a suitable camera. Once found, it activates this camera and captures the live feed, transforming it into a texture that matches the camera's dimensions. This texture is then applied to a gameObject within the Unity environment, effectively rendering the live camera feed.

Preconditions	Inputs	Output
The target devices must have a camera	1. Device Camera	1. Texture material on object renderer
Key Variables		
Name	**How it's used**	**Data Type**
texture	This is the texture material that will be generated from the camera output and applied to the gameObject renderer, in this case, it will be the camera gameObject	**UnityEngine.Texture2D** - I used this as this is the class that represents textures in C# code within Unity.
WebCamTexture	This is the texture that will be captured from the camera. Depending on the camera currently selected.	**UnityEngine.WebcamTexture** - I used WebCam Textures as these are the textures onto which the live video input is rendered.
currentCameraIndex	This is used to cycle through the onboard cameras by indexing them.	**Integer** - an integer is used to keep track of the currently selected camera as it will be a whole number cycled through
FER_model_filepath	This links the filepath to the emotion detection model that will be used	**String** - a string is always used when passing a file as a variable as the filepath is in a string format
face_detection_model_filepath	The links the filepath to the face detection model that will be used	**String** - a string is always used when passing a file as a variable as the filepath is in a string format

Link to success criteria
This covers success criteria point 1 (Camera functionality when user opens app). The camera feed is fundamental for the app to function as it provides the visual input for all processing. Without it, none of the application features can be achieved.

Update() - Face Detection	Description

```
int faceCount = 0;

texture.SetPixels(webCamTexture.GetPixels());
texture.Apply();

Mat faces = faceDetector.infer(texture);
List<Mat> expressions = new List<Mat>();

for (int i = 0; i < faces.rows(); ++i)
            {
               faceCount++;
               Mat facialExpression = FER.infer(texture, faces.row(i));

               if (!facialExpression.empty())
                 {
                  expressions.Add(facialExpression);
                  }
            }


faceDetector.visualize(faces, texture);
FER.visualize(expressions, texture);

gameObject.GetComponent<Renderer>().material.mainTexture = texture;

faceCountText.text = ("Faces Detected:\n" + faceCount);
```
	This algorithm, executed every frame in Unity's Update() method, refreshes a texture with the webcam's live feed, then applies face detection to identify faces within the frame. For each detected face, it evaluates facial expressions, incrementing a count for faces and collecting expression data. Detected faces and expressions are visualized on-screen, with the face count displayed for immediate feedback. The process concludes by converting the detection results back into a texture for Unity rendering, enabling real-time monitoring and interaction with detected facial features and expressions.

Preconditions	Inputs	Output
A camera is available and capable of capturing video feed. The faceDetector and FER (Facial Expression Recognition) models are preloaded and initialized.	1) Webcam Texture	1) Updated Texture 2) Face Count 3) Expressions List
Key Variables		
Name	**How it's used**	**Data Type**
faceCount	Counts the number of faces detected in the current frame.	**Integer** - an integer is used for this as there can only be a whole number of faces. Keeping track and calculating with an integer is much easier than other data types
texture	Stores the texture that is updated with the webcam feed and later with the detection and recognition results.	**UnityEngine.Texture2D**
faces	Stores the detected faces in the frame.	**UnityEngine.Mat** - The Unity material class. This class exposes all properties from a material, which can then be processed and manipulated.
expressions	Collects the facial expression data for each detected face.	**List** - a list is used as there may be multiple faces. I did not use an array as the number of faces may changed and is not fixed, and a C# list is dynamic
facialExpression	Temporarily holds the facial expression data for a single detected face before	**UnityEngine.Mat**

	adding it to the expressions list if not empty.	
Link to success criteria		
This covers success criteria points 3 (Identification of people in camera's view/facial recognition), 6 (Mood recognition based on facial expressions), and 9 (Responsiveness (20fps consistent feed)). The face detection is fundamental for this project as it is provides the backbone of all the other features		

SendScreenshot.cs - start()	Description

private bool connected = false;
public Text = captionTextbox;
 	
connected = false;
InvokeRepeating("CheckConnection", 0.1f, 0.5f);
captionTextbox = "";
	The Start function initializes application settings by marking the 'connected' variable as false to indicate no initial internet connectivity. It employs InvokeRepeating to periodically call the 'CheckConnection' method every 0.5 seconds after a 0.1-second delay, aiming to assess and update the internet connection status. Simultaneously, it clears any text in the 'captionTextbox' textbox to prevent displaying captions until the app confirms internet access, ensuring functionality like image captioning is dependent on current connectivity. This approach ensures the app remains responsive to changes in internet status.

Preconditions	Inputs	Output
1) A textbox UI element is available for displaying captions. 2) A coroutine named 'CheckConnection' exists for checking internet connectivity. 3) A boolean variable 'connected' is declared to track internet connectivity status.	none	1) Updated 'connected' Variable: Reflects the current internet connectivity status. 2) Textbox Content: The content of the textbox ('captionTextbox') is reset to an empty string.
Key Variables		
Name	**How it's used**	**Data Type**
'connected'	Initially set to false, this variable indicates whether the application has internet connectivity. Its value is intended to be updated based on the outcome of the 'CheckConnection' coroutine, enabling or disabling features like image captioning based on the internet connection status.	**Boolean** - connected can only be 1 of 2 values, so a boolean is used to easily keep track in order to successfully maintain network validation
'captionTextbox'	A UI element meant for displaying captions. It is cleared at the start to ensure no caption is displayed until one is properly loaded or generated.	**UnityEngine.UI.Text** - a UI textbox is used as the user should be able to read the text on screen. A unity textbox is able to be manipulated and changed via a script and is the best choice.
Link to success criteria		
This covers parts of success criteria 8 (Image capture and captioning) and 11 (Ease of use). Clearing the caption text box and ensuring an internet connection allows the user to get a clear view of the screen and prevent any connection errors from arising when captioning an image.		

SendScreenshot.cs - CheckConnection()	Description

```
private string ip = "8.8.8.8";      //Google's public ip address

IEnumerator CheckConnection()
  {
    Ping = new Ping(ip);
    float startTime = Time.time;
    while (Ping.complete == false)
    {
      yield return null;
      if (Time.time - startTime > 4.0)
      {
        connected  = false;
        captionButton.enabled = false;
        Debug.Log("Not Connected");
        yield break;
      }
    }
    connected = true;
    captionButton.enabled = true;
    Debug.Log("Connected");
  }
```
	This algorithm assesses the device's internet connectivity by attempting to ping Google's public IP address (8.8.8.8). It initiates a Ping operation to this address and monitors the ping response within a coroutine called CheckConnection. The process starts by recording the current time (startTime) and enters a while loop that continuously checks if the ping operation has completed. If the ping response is not received within 4 seconds, the algorithm concludes that the device is not connected to the internet (connected = false), logs a "Not Connected" message, and exits the coroutine to avoid unnecessary processing. If the ping operation completes successfully within the 4-second window, it sets the connected variable to true and logs "Connected," indicating a successful internet connection. This method provides a straightforward, time-bound check for internet connectivity, crucial for applications that require an active network connection to function correctly.

Preconditions	Inputs	Output
1) The device is capable of initiating ping requests. 2) Google's public IP address (8.8.8.8) is accessible and can be used to test internet connectivity.	1) Google's ip address	1) Connection status boolean updated 2) Debug Connection status
Key Variables		
Name	**How it's used**	**Data Type**
'ip'	Stores Google's public IP address as the target for the ping operation to test internet connectivity.	**String** - an ip address is in the format "x.x.x.x" due to the multiple 'dots', the ip must be passed as a string as it is not an integer, or a float
'startTime'	Records the time at the start of the ping operation to track its duration.	**Float** - Time will be a decimal value which is why a float is used for this.
Link to success criteria		
This covers parts of success criteria 8 (Image capture and captioning) and 11 (Ease of use). Preventing any connection errors from arising when captioning an image.		

SendScreenshot.cs - SendScreenshot()

```
public List<GameObject> UIElements;
public string serverIP;



if (connected == false)
     {
        captionTextbox = "not connected";
        return;
     }
else
     {
       captionButton.enabled = false;

       foreach (GameObject obj in UIElements)
         {
        obj.SetActive(false);
         }

       Texture2D screenshot = Screenshot.Texture();
       byte[] screenshotBytes = screenshot.Encode();

       WWWForm form = new WWWForm();
       form.AddData(screenshotBytes);

     // Send the POST request to the Flask server
     using (UnityWebRequest www = Post(serverIP, form))
     {
        yield return www.SendRequest();

        if (Request.Success != true)
        {

           Debug.Log("upload failed: " + errorCode);
        }
        else
        {
           Debug.Log("Uploaded successfully!");

           string caption = www.downloadHandler.text;
           Debug.Log("Caption: " + caption);
           captionTextbox= captionJson;

        }

        captionButton.enabled = true;

       foreach (GameObject obj in UIElements)
         {
        obj.SetActive(true);
         }

     }
```

Description

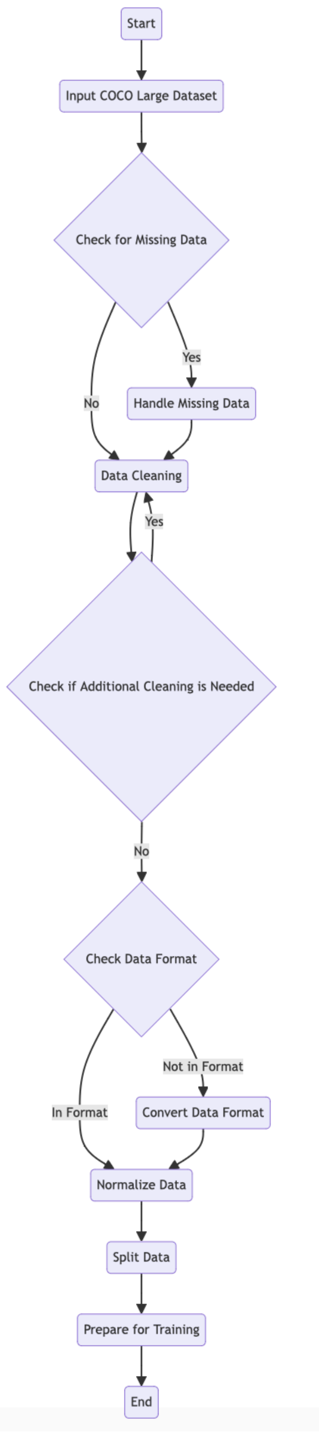
This algorithm allows a Unity application to send screenshots to a server for captioning, based on whether the device has an internet connection. If there's no internet, it immediately informs the user by displaying "not connected" in the caption textbox and stops. If connected, it temporarily disables a button to prevent multiple requests and hides user interface elements to process the screenshot.

It captures the current screen, converts it to a digital format, and sends this data to a server. The application waits for the server to analyze the image and return a text caption describing it. If the server responds successfully, the application displays this caption in a textbox. If there's a problem with sending the image or receiving the caption, it logs an error message.

After receiving the caption or encountering an error, the application re-enables the button and shows the user interface elements again, ready for further interaction. This process enhances the application's functionality by providing users with descriptions of their screenshots, making it more interactive and accessible.

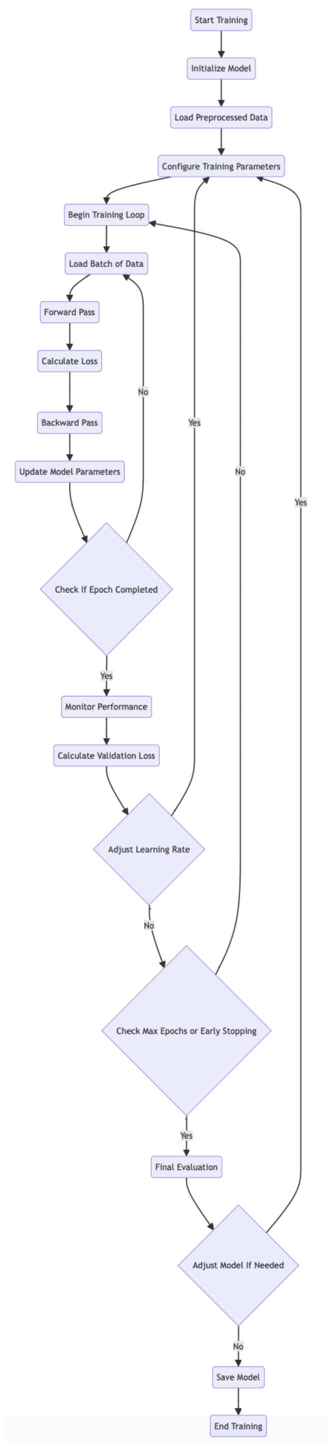
Preconditions	Inputs	Output
1) A Python server capable of receiving images and returning	1) Screenshot: The image captured from the current screen within the	1) Caption: The descriptive text returned by the server, based on

captions is available and accessible.	Unity application. 2) Server IP: The address of the Python server to which the screenshot is sent for captioning.	the analysis of the screenshot. 2) Error message (optional): Displayed in case of a failure in sending the screenshot or receiving the caption.
Key Variables		
Name	**How it's used**	**Data Type**
'connected'	Indicates whether the application has internet connectivity.	**boolean**
'captionTextbox'	Displays the caption received from the server or a connectivity error message.	**UnityEngine.UI.Text**
'UIElements'	Represents the UI elements that are temporarily hidden during the screenshot processing to focus on the operation.	**list**
'serverIP'	Stores Google's public IP address as the target for the ping operation to test internet connectivity.	**string**
Link to success criteria		
This covers parts of success criteria 8 (Image capture and captioning) and 11 (Ease of use). Preventing any connection errors from arising when captioning an image.		

PrepareData()	Description
 Start Input COCO Large Dataset Check for Missing Data Yes No Handle Missing Data Data Cleaning Yes Check if Additional Cleaning is Needed No Check Data Format Not in Format In Format Convert Data Format Normalize Data Split Data Prepare for Training End	The flowchart describes an algorithm for processing a large COCO dataset, commonly used in machine learning for object detection, segmentation, and captioning tasks. The algorithm ensures the data is ready for training a machine learning model by following these steps: Start: Initiate the data preparation process. Input COCO Large Dataset: Load the COCO dataset, which is known for its large volume and rich annotations for object detection and segmentation tasks. Check for Missing Data: Evaluate the dataset to identify any missing data points or annotations. This is a crucial step to ensure the integrity of the dataset. Handle Missing Data: If missing data is detected, appropriate actions are taken to handle it. This can include removing the incomplete samples, filling in missing values with estimated data, or using algorithms to predict missing annotations. Data Cleaning: Perform a general cleaning process on the data, which may include removing duplicates, correcting errors, or standardizing the format of the data entries. Check if Additional Cleaning is Needed: Assess whether the dataset requires further cleaning. This is an iterative process that may be repeated multiple times to ensure the data quality is up to the standards necessary for effective model training. Check Data Format: Verify if the dataset is in the desired format that is compatible with the machine learning model or the training process. Convert Data Format: If the dataset is not in the required format, convert it to the correct format. This might involve changing file types, adjusting data structures, or transforming the way annotations are recorded. Normalize Data: Apply normalization to the dataset to ensure that the numerical values are within a standard range, which helps in the convergence of the model during training. Split Data: Divide the dataset into subsets typically used for training, validation, and testing. This helps in evaluating the model's performance and avoiding overfitting. Prepare for Training: Finalize the preprocessing steps to make the dataset ready for training a machine learning model. This could include compiling the data into a specific file structure, generating metadata, or shuffling the data for randomness. End: Conclude the data preparation process. The dataset is now ready to be used in the training of a machine learning model. This flowchart is a high-level representation of the preprocessing steps necessary before the actual training of a machine learning model can begin. It ensures that the dataset is clean, well-formatted, and properly structured to promote efficient and effective training outcomes.

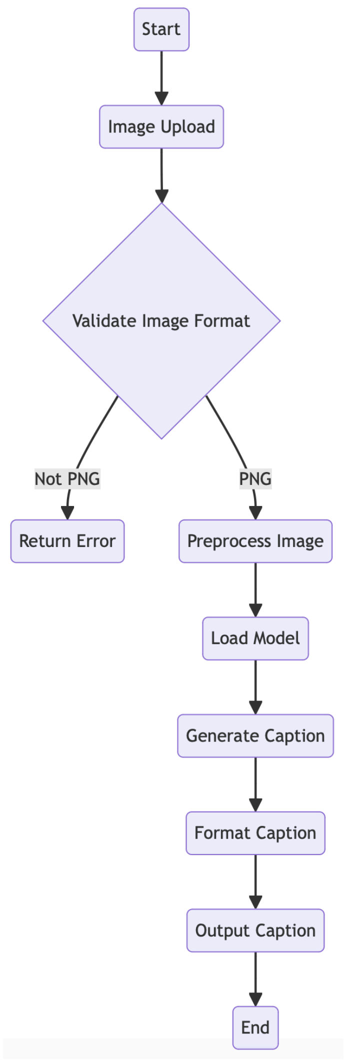
Preconditions	Inputs	Output
1) The COCO dataset is available and accessible. 2) There's sufficient computational	1) COCO Large Dataset: The actual dataset that contains images along with their corresponding	1) Clean and Prepared Dataset: A version of the COCO dataset that has been cleansed of missing or

resources to handle large dataset processing.	annotations, which may include object bounding boxes, segmentation masks, and image captions.	erroneous data, formatted correctly, normalized, and split into training, validation, and test sets, ready for input into a machine learning training process.
Key Variables		
Name	**How it's used**	**Data Type**
'MissingData' flag	A boolean or indicator that shows whether there is missing data within the dataset.	**boolean**
'DataFormat' flag	A boolean or indicator that signifies whether the data is in the required format for processing.	**boolean**
Link to success criteria		
This covers parts of success criteria 8 (Image capture and captioning) and 10 (Accuracy)		

TrainData()	Description
 Start Training Initialize Model Load Preprocessed Data Configure Training Parameters Begin Training Loop Load Batch of Data Forward Pass Calculate Loss Backward Pass Update Model Parameters No Yes No Yes Check If Epoch Completed Yes Monitor Performance Calculate Validation Loss Adjust Learning Rate No Check Max Epochs or Early Stopping Yes Final Evaluation Adjust Model If Needed No Save Model End Training	This flowchart outlines the steps involved in training a machine learning model. The process begins with the initiation of the training phase, where the model is first initialized. Following this, the preprocessed data that will be used to train the model is loaded. Next, training parameters such as learning rate, batch size, and number of epochs are configured. The core of the training takes place in a loop where a batch of data is loaded and fed into the model to perform a forward pass. This forward pass computes the predictions of the model, and then the loss is calculated to measure the discrepancy between the predictions and the true labels. A backward pass follows, which involves backpropagation to calculate the gradients of the loss with respect to each parameter. The model's parameters are then updated in the direction that minimizes the loss. After updating the model's parameters, the algorithm checks whether an epoch is completed. An epoch is completed when the entire dataset has been fed through the model. Once an epoch is completed, the model's performance is monitored by calculating the validation loss on a separate set of data not seen by the model during training. If necessary, the learning rate is adjusted based on the performance on the validation set. Then, the algorithm checks if the maximum number of epochs has been reached or if early stopping criteria have been met. Early stopping is a condition where training can be halted if the validation loss stops improving for a set number of epochs, preventing overfitting. If the stopping criterion is met, the model undergoes a final evaluation. If the model's performance is not satisfactory, adjustments to the model may be made. Once the model is deemed ready, it is saved for future use, and the training process ends.

Preconditions	Inputs	Output
1) A machine learning model	1) Preprocessed Data: The dataset	1) Trained Model: The machine

architecture is defined and ready to be initialized. 2) Preprocessed data is available in a format compatible with the model.	that has been cleaned, normalized, and split into training and validation sets, ready for the model to learn from. 2) Training Parameters: Settings like learning rate, batch size, number of epochs, and other hyperparameters critical for training.	learning model that has been trained on the dataset, with its parameters tuned for optimal performance. 2) Validation Loss: A measure of the model's performance on the validation set, used to monitor learning progress and adjust hyperparameters. 3) Final Model Performance: The results of the final evaluation, determining if the model's training is sufficient or needs further adjustment.
Key Variables		
Name	**How it's used**	**Data Type**
'ModelParameters'	The weights and biases within the model that are adjusted to minimize the loss.	**string/dictionary** - The parameters will be a number of variables and inputs which will be passed as a string or dictionary depending on the parameter type used for image training.
'Loss'	A value calculated from the forward pass indicating how well the model's predictions match the true data.	**Float** - the loss value is a decimal value that evaluates the success rate of the model so a float is used
'Learning Rate'	A hyperparameter that controls the size of the steps taken during the optimization process.	**Float** - The Learning rate is a coefficient between 0 and 1 and thus a float is used to represent this
Link to success criteria		
This covers parts of success criteria 8 (Image capture and captioning) and 10 (Accuracy)		

ProcessImage()	Description
 Start Image Upload Validate Image Format Not PNG PNG Return Error Preprocess Image Load Model Generate Caption Format Caption Output Caption End	This flowchart outlines the procedure for an image captioning AI that takes an image as input and outputs a descriptive caption. The process starts when a user initiates an image upload. Upon upload, the first step is to validate the image format. The AI is programmed to work with PNG format images, so it checks if the uploaded image is a PNG. If not, the system returns an error message to the user. Assuming the image is in the correct PNG format, the process moves forward to the preprocessing stage. During preprocessing, the image is prepared for analysis, which may involve resizing, normalizing pixel values, or applying other transformations to make the image suitable for the model. Once preprocessed, the AI loads a trained model designed for image captioning. This model has been trained on a dataset of images and corresponding captions to learn how to generate descriptions for new images. With the model loaded, the AI performs inference to generate a caption for the uploaded image. After the caption is generated, it may go through a formatting step to ensure that it's in a human-readable form, correcting grammar, or adjusting the sentence structure as needed. Finally, the AI outputs the formatted caption to the user, providing a descriptive text of the uploaded image. The process then ends until a new image upload is initiated.

Preconditions	Inputs	Output
1) The trained image captioning model is available 2) Preprocessed data is available in a format compatible with the model.	1) Image Upload: A user-uploaded image that the system will process to generate a caption.	1) Caption: The descriptive text produced by the model that describes the content of the uploaded image. 2) Error Message: If the uploaded image is not in PNG format, an error message is returned to the user.
Key Variables		
Name	**How it’s used**	**Data Type**
‘Model’	The image captioning model loaded into	**string (filepath)** - file paths are passed

	memory for generating captions.	as strings
‘GeneratedCaption’	The initial caption produced by the model before any formatting is applied.	**string**
‘FormattedCaption’	The final version of the caption after it has been processed for readability, which is then presented to the user.	**string**
Link to success criteria		
This covers parts of success criteria 8 (Image capture and captioning) and 10 (Accuracy)		

SceneSwitcher.cs	Description

```
public void emotionDetection()
  {
    SceneManager.LoadScene(sceneName: "Emotion Detection");
  }
  public void objectDetection()
  {
    SceneManager.LoadScene(sceneName: "Object Detection");
  }
  public void Splash()
  {
    SceneManager.LoadScene(sceneName: "Splash");
  }
  public void Settings()
  {
    SceneManager.LoadScene(sceneName: "ExampleScene");
  }
```
	This script is part of a user interface in a Unity application that allows users to navigate between different scenes, each providing a unique functionality. The script contains four public functions, each linked to a different button within the application. When a user presses one of these buttons, the corresponding function is called, executing a scene change within the Unity environment.

Preconditions	Inputs	Output
1) The Unity application has multiple scenes created and named correctly: "Emotion Detection", "Object Detection", "Splash", and "ExampleScene". 2) Each function is correctly hooked up to its respective button within the Unity Editor.	1) Button Press: The user interaction that triggers the function calls. There is no data input to the functions themselves, as they are simply called in response to the user's action.	1) Scene Change: The visible output is the new scene that the application navigates to when a function is called.
Link to success criteria		
This covers parts of success criteria 2 (User interface design), 5 (Home screen functionality) and 11 (Ease of use). For a project to be successful, UI and page navigation should be easy and seamless, which is what this script aims to acheive		

Algorithm Validation

Internet connection validation

For the image captioning to work, a screenshot must be taken from the unity application and sent to an external python server to be processed, this is because as of my knowledge, there are no useful image captioning models that work within Unity and if I were to train my own, importing and converting to a Unity-compatible format would be very difficult and take a lot of time, so instead I will be processing externally. For this to work, the device must maintain a constant internet connection.

In order to do this, I will be making a call to a function every few seconds that attempts to ping Google's public ip address (8.8.8.8). If the ping is successful, the application will continue as normal, indicating that the device is connected to the internet, however if the ping fails, it is likely that device is not connected to the internet, or has a very weak connection. In this case, I will disable the image captioning button and display a small error message in the corner of the screen, telling the user to connect to the internet. This validation ensures that the user can enjoy the full experience of the program, without experiencing connection errors from the server, which may leave them confused and unsatisfied.

Providing them with a reason why a certain function is not working is critical in maintaining the user experience and overall useability of the application.

Below is the pseudo code that contains the logic for this functionality.

```
InvokeRepeating("CheckConnection", 0.1f, 0.5f);

private string ip = "8.8.8.8";     //Google's public ip
address

IEnumerator CheckConnection()
  {
     Ping = new Ping(ip);
     float startTime = Time.time;
     while (Ping.complete == false)
     {
        yield return null;
        if (Time.time - startTime > 4.0)
        {
           connected  = false;
           captionButton.enabled = false;
           Debug.Log("Not Connected");
           yield break;
        }
     }
     connected = true;
     captionButton.enabled = true;
     Debug.Log("Connected");
  }
```

The 'InvokeRepeating()' function keeps calling a function every *n* seconds until stopped.

For this application, The "CheckConnection" coroutine is called every 0.5 seconds where it attempts a ping and starts a 'timer'.

If the timer exceeds 4 seconds, it can be deemed that the connection is either too slow or has failed.

When this happens, the button for image captioning is disabled and an error message is logged in the console.

This function will be called again and again until the application is closed.

If the connection comes back online and works, the image captioning button is re-enabled and 'connected' is logged in the console.

Touchscreen input validation

My project will be primarily a mobile applicaiton, which means that the sole input of the user will be the touchscreen and various buttons. This approach inherently limits the scope of input types, effectively simplifying the validation process as users cannot input data through traditional means such as text entry. Validation in this scenario focuses on recognizing valid touch interactions, distinguishing between intentional touches on actionable buttons and accidental or non-targeted touches elsewhere on the screen. The application can accurately determine the intent behind each touch, ensuring that only deliberate and correctly executed touches trigger corresponding actions within the app. This method of input validation enhances user experience by minimizing errors and ensuring that the application responds promptly and accurately to user commands, maintaining a streamlined and intuitive interface.

Image caption validation

Upon confirming internet connectivity and user initiation (through a button press), the application proceeds to capture a screenshot. Before capturing, it temporarily disables UI elements that should not be included in the screenshot, ensuring that only the relevant screen content is captured. This meticulous preparation step is vital for maintaining the relevance and purity of the screenshot data sent for captioning.

The captured screenshot is then encoded into PNG format and packaged into a ‘WWWForm’ object as binary data. This step is critical for converting the screenshot into a format that can be transmitted over the network and understood by the server. The use of a well-defined content type ‘(image/png)’ further validates the data by specifying the format explicitly to the server, ensuring the server can correctly process the received image.

The application then initiates a POST request to the server, including the screenshot data. Upon receiving a response, it conducts a basic validation check to ensure the request was successful. If the upload fails, an error is logged, and no further action is taken, effectively preventing invalid data from proceeding through the system. For successful uploads, the server's response, presumably a caption in JSON format, is directly displayed to the user. This direct use of the server's response without modification ensures that the validation responsibility is primarily on the server side, which is expected to return correctly formatted and meaningful captions.

Finally, the application re-enables previously disabled UI elements and updates the UI to reflect the received caption, thereby completing the validation loop. This step ensures that the user is provided with immediate, visible feedback based on the validity of the data exchange process.

Iterative Development Test Data

Iteration 1

The main focus for iteration 1 will be the camera output and the initial set up and imports for the AI models. This will perhaps take the most time as it will require the greatest amount of research. The tests will be focused on the app’s basic functionality such as camera output, face detection and iphone deployment.

Test num.	What is being tested	Priority	Input	Expected output	Justification for this data
1	Camera appears on screen and updates in real time	1	Device webcam	Live feed of camera on screen	This is the most crucial part of the project – if the user cannot see what the camera is seeing then there will be no functionality to the rest of the project.
2	Bounding box around detected faces	1	Device webcam	Coloured box being drawn around the face, mapped by calculating coordinates of face corners and adding padding to accommodate	In order for the face detection functionality of my project to work, I will need to make sure that there are bounding boxes that display whether or not there are faces in frame. This will allow the processing of emotion detection.
3	Confidence rating on face detection bounding box	2	Device webcam + detected face Mat	Float between 0 to 1 that is the confidence rating from the AI on its certainty that the bounding boxes are bounding around a face where 1 is 100% certain and 0 is 0% certain	This is not critical to the program, although it is a great feature to have as with the confidence rating, we are able to omit certain detections that fall below the threshold, meaning that we will get less false positive errors throughout the app. Additionally, being able to visually see how accurate it is will provide a greater insight into the performance of the AI model
4	Second bounding box, with detected emotions, colour coded	1	Device webcam + list of detected faces	Inner bounding box, that displays the detected emotion as a text that follows the person’s face, also colour co-ordinated so that each different emotion is	This is needed as the core functionality of my project, before all else is an emotion detection app AI, and thus the ability to detect emotions is crucial – This would not be possible without the face detection.

				correspondent to a different coloured bounding box	
5	Confidence rating on the emotions detected by the AI	2	Detected emotion + bounding box	Float between 0 to 1 that is the confidence rating from the AI on its certainty of the prediction of the person's emotion,	See Test num. 3
6	Ability to detect multiple faces	2	Device webcam + FaceDetection AI	Multiple different bounding boxes for each detected face, along with their confidence rating and detected emotion for each	This is fairly important as there will often be more than one person in frame when pointing the camera at someone, especially in a public environment, so it is important to be able to process multiple different objects simultaneously.
7	Resizing quad dependent on screen size	3	Device screen width + height, Webcam resolution	Quad should scale adequality with screen size and maintain aspect ratio	This is not heavily important to the functioning of the program, but to ease useability and prevent cropping and issues, this feature should be executed well
8	Ability to switch between cameras	1	List of device cameras, switch camera button	When button is pressed, program will cycle through the available cameras, skipping those that are unnecessary, such as ultrawide	This is very important as the end-user will most likely want to be able to use the program for the front and rear camera of their phone and the ability to seamlessly switch between them will increase the quality and useability of the program.
9	Switch Camera button disappears when only one camera on system	2	List of device cameras	If there is only one camera, such as on a laptop, the switch camera button should be disabled	This is not critical but it is important as the switch camera button will be useless to a user who is using a device with only one camera – this is because it will call a function to cycle through cameras, yet there are none to cycle through, so it will only cause the screen to lag slightly.

10	Auto-Rotate scale camera	2	Device orientation	Camera matches device orientation seamlessly	This is important as many users like to use the camera in portrait, while others in landscape, so in order to maximise accessibility, there needs to be functionality to switch between both.
11	iPhone Deployment	1	n/a	The application runs and works when built on an iphone using xCode for deployment	My application is aimed to be accessible via a mobile phone. In order to test this, I must be able to build and use the application externally on my phone.

Iteration 2

This iteration introduces image captioning, which requires extensive amounts of research and programming. The captioning will likely take place via an external python server, which will result in lots of extra code. I will also program the internet connection validation subroutine.

Furthermore, I will also introduce YoLo object detection as an additional features in this iteration, should it be easy and I have the time.

Test num.	What is being tested	Priority	Input	Expected output	Justification for this data
1	On-screen face counter	1	Device webcam + detected faces	Integer displaying number of faces in real-time	This will aid in accessibility and allow the user to gain a knowledge of the estimated number of people in the image
2	YoLo object detection	3	Device webcam	Coloured boxes are drawn around different objects, and mapped by calculating the coordinates of each object and assigning a label to them.	The YoLo object detection is an extra feature within this project. It is important that this functionality works well, however not critical to the application as this is not going to be the primary focus.
3	Splash screen/scene switcher	1	N/A	The splash screen will allow the user to choose which functionality they would like to use in the app, whether it be emotion/face detection, or a general-purpose object detection AI.	This is quite important as in order for the user to navigate the application, they need a splash screen that will allow it. The splash screen should be the one the loads up when the user opens the application.
4	Check internet connection	2	N/A	If there is an internet connection, debug “connected” if there is no connection, the screenshot button, when pressed will show a “not connected to the internet” text instead of the caption.	This is needed as the image captioning part of the project is hosted on a server and therefore an internet connection is needed. If there is no connection, pressing the button will try to send outbound connections and return errors, therefore by disabling this, we are able to reduce any unnecessary code execution, thus optimizing the code.

5	Image Captioning	1	Send Screenshot button	A small delay, then a message in the middle of the screen displaying an AI generated caption of the camera feed	This is the main part of iteration 2 that took me the most time to implement. This is one of the key aspects in my project. Someone perhaps with visual problems, is able to have the image in front of them described, and then, in a future iteration, this can be coupled with a text-to-speech system so that the caption is spoken aloud to the user.
6	Home screen button functionality	2	'home' button	Takes the user back to the splash screen	This is so that, if they wanted, the user can return to the splash screen and choose a different functionality to use for the application
7	Image resolution	3	Device camera	High resolution, but doesn't make the program slow	This is so that the user can have a reasonable experience when using this app and also be able to distinguish the camera feed at a high resolution.
8	Aspect ratio/rotation	2	Device orientation	Correctly displayed output when using the application	This is so that the image is consistently upright when using the program
9	iPhone deployment	1	n/a	The application runs and works when built on an iphone using xCode for deployment	This will be tested at each iteration in order to ensure that during development, no build errors occur which may cause the application to fail when deploying to an iPhone.

Iteration 3

This iteration will focus on UI overhaul and useability of the project, where I will develop the Text-To-Speech system, allowing the user to hear aloud the generated caption, along with creating eye-catching and easy-to-follow navigation and UI buttons, with a settings page to provide customisation to the TTS system.

Test num.	What is being tested	Priority	Input	Expected output	Justification for this data
1	On-screen face counter UI updating	2	Detected faces	Integer displaying number of faces in real-time on a grey background	This will aid in useability and promote a more user-friendly readable interface
2	Image caption on grey background	2	Image Captions	Grey background that the captions are displayed on so the user can see clearly	The YoLo object detection is another key aspect of this project. It is important that this functionality works well.
3	Splash screen/scene switcher UI Buttons	3	touchscreen	The splash screen will allow the user to choose which functionality they would like to use in the app, whether it be emotion/face detection, or a general-purpose object detection AI.	This is to allow the user to navigate the application. They need a splash screen that will allow it. The splash screen should be the one the loads up when the user opens the application.
4	Settings/Voice Settings page	1	Settings Page button	Opens the settings scene for the TTS functionality	This is needed as the captions will be read aloud through a TTS system. This should be customizable by the user in order to tailor to their needs and preferences
5	Caption Text-To-Speech	1	TTS button	Upon pressing the button, the application will read out the displayed caption out loud, in the voice/setup done by the user	This is one of the key features of iteration 3 and the project itself, which aims to assist those with visual impairments by providing them with a new, audible way of viewing their surroundings using AI.
6	Clear Captions button	2	Clear Captions button	When this button is pressed, any caption loaded and displayed on screen is cleared to provide a clearer view of the screen.	This is important as if the user wants to focus more on the camera output and facial expressions, the caption can get in the way, so there is a button to get rid of this.

7	‘Cancel Upload’ functionality	2	‘Cancel upload’ button	When this button is pressed, the caption request is cancelled and the UI elements re-appear on screen	This is important as if the client has a very slow internet connection, this process can be tedious and boring, therefore they should have an option to cancel this operation on request.
8	‘Loading’ icon for image captioning	2	Image caption button	When the image is being processed and request is sent to server, animated icon starts playing to indicate it is being processed.	The loading icon will indicate to the user when the image is being captioned, that way they aren’t confused about the lack of UI and functionality when the button is pressed.
9	TTS customisation	2	‘Settings’ Page	The user will have the ability to customise the speed, pitch, and volume of the Text-to-Speech voice	This will help the user have a more personalised experience within the app, which is the primary focus for iteration 3
10	iPhone deployment	1	n/a	The application runs and works when built on an iphone using xCode for deployment	This will be tested at each iteration in order to ensure that during development, no build errors occur which may cause the application to fail when deploying to an iPhone.

Post Development Test Data

Test No.	Aspect being tested/Link to success criteria	Justification	How to perform test	Type of test (valid, invalid, boundary)	Expected result	Actual result	Evidence (screenshot, screencast etc)
1	A title page/main menu is loaded Succcess criteria points #2 and #5 (User interface design) (Home screen)	Ensuring the title page/main menu loads correctly is crucial for providing users with a clear starting point and navigation options, enhancing their overall application experience.	Start the application	Valid	App displays the home splash screen		
2	Main menu should allow the user to navigate to ‘emotion detection’ scene Succcess criteria points #2, #5 and #11 (User interface design) (Home screen) (Ease of use)	Testing the main menu's navigation to the 'emotion detection' scene is essential for ensuring users can access core features seamlessly, improving usability and engagement with the application.	Press ‘emotion detection’ button in home screen	Valid	App displays the ‘emotion detection’ scene		
3	Main menu should allow the user to navigate to ‘object detection’ scene Succcess	Testing the main menu's navigation to the ‘object detection' scene is essential for ensuring users can access core features seamlessly, improving usability	Press ‘object detection’ button in home screen	Valid	App displays the ‘object detection’ scene		

	criteria points #2, #5 and #11 (User interface design) (Home screen) (Ease of use)	and engagement with the application.					
4	Main menu should allow the user to navigate to ‘settings’ scene Succcess criteria points #2, #5 and #11 (User interface design) (Home screen) (Ease of use)	Testing the main menu's navigation to the ‘settings' scene is essential for ensuring users can access core features seamlessly, improving usability and engagement with the application.	Press ‘settings’ button in home screen	Valid	App displays the ‘settings’ scene		
5	Camera view appears when in the ‘emotion detection’ scene Success criteria points #1 (Camera functionality when user opens app)	Verifying that the camera view appears in the 'emotion detection' scene is vital for enabling users to capture images in real-time, which is fundamental to the functionality and user interaction with the application.	Start the ‘emotion detection’ scene	Valid	App displays the output of the camera		
6	Camera functionality within the 'object detection' scene Success criteria points #1, #4	Verifying camera functionality and integration within the object detection scene to ensure that the app can effectively use the camera to identify and categorise	Start the ‘object detection’ scene	Valid	App displays the output of the camera		

	(Camera functionality when user opens app) (Object detection)	objects in real-time, a core feature that must perform reliably for the app's intended purpose.					
7	Display of facial recognition feature in the 'emotion detection' scene Success criteria points #1, #3 (Camera functionality when user opens app) (Identification of people in camera's view/facial recognition)	Demonstrating the display and functionality of the facial recognition feature in the emotion detection scene to ensure it operates correctly within this context, accurately identifying and analysing facial expressions for mood assessment.	Start the ‘emotion detection’ scene and point camera at someone’s face.	Valid	App displays a bounding box around faces and prints the detected emotion close the them.		
8	Objects listed in the 'object detection' scene Success criteria points #4, #8 (Object detection) (Image capture and captioning)	Turn on an test the object detection scene to verify that the feature works seamlessly, enhancing the educational and accessibility aspects of the app.	Press ‘object detection’ button on homescreen	Valid	Object detection scene loads and displays object names besides their bounding boxes.		
9	Responsivenes s and fluidity of the live camera feed	Assessing the responsiveness and fluidity of the live camera feed for real-time	Starts either the ‘emotion detection’ or ‘object detection’ scene, as both use the same logic for the live	Valid	Camera is fluid and responsive ~20 fps		

	Success criteria points #1, #9 (Camera functionality when user opens app) (Responsiveness - 20fps consistent feed)	interaction to ensure the application can handle streaming video at a consistent frame rate, critical for a smooth and engaging user experience in features like live emotion detection or object recognition.	camera feed				
10	Accuracy of facial expression analysis for mood recognition Success criteria points #3, #6, #10 (Identification of people in camera's view/facial recognition) (Mood recognition based on facial expressions) (Accuracy)	Validating the accuracy of facial expression analysis for mood recognition to ensure the technology correctly interprets a wide range of human emotions, pivotal for applications relying on emotional intelligence for interaction or feedback.	Start 'emotion detection' scene and point camera at someone displaying different moods	Valid	'Mood' text updates in real time and accurately detects a person's emotion		
11	User interface intuitiveness and simplicity Success criteria points #2, #11 (User interface design) (Ease of use)	Ensuring the user interface is intuitive and simple, facilitating ease of use for a wide demographic of users, including those who may not be tech-savvy, to reduce the learning curve and enhance the overall user	Start the application and look at the UI elements on screen	Valid	UI elements appear on screen. The UI elements should be intuitive and simple to follow from the point of view of someone who may not be confident		

		engagement with the app.			using tech.		
12	Customization of text-to-speech settings in the 'settings' scene Success criteria points #7, #12 (Text-to-speech output for processed attributes) (Variable text-to-speech voice pitch/volume/speed)	Customising text-to-speech settings in the 'settings' scene to validate the range of personalisation options available to users, ensuring they can adjust voice pitch, volume, and speed to suit their preferences and needs, enhancing accessibility and user experience.	Vary the sliders and voice carousel in the settings page until preferred TTS voice is as desired	Valid	User is able to change the voice, volume, pitch and speed of TTS speech.		
13	Object detection accuracy and performance Success criteria points #4, #10 (Object detection) (Accuracy)	Checking the accuracy and performance of object detection under a variety of conditions to validate the technology's ability to recognise and categorise objects reliably, which is central to the app's functionality and user engagement.	Turn on ‘object detection’ scene and evaluate performance when pointing at many objects in different environments	Valid	Object detection should be accurate and remain highly responsive, even when many objects are in view		
14	Responsiveness of the main menu and navigation elements Success criteria points #2, #9, #11 (User interface design)	Testing the responsiveness of the main menu and navigation elements to ensure that users can quickly and easily access different parts of the application without frustration, critical for usability and user satisfaction.	Press different buttons and evaluate the response time and loading time of different scenes and processing.	Valid	Application should respond to user inputs quickly and with minimal delay		

	(Responsivenes s - 20fps consistent feed) (Ease of use)						
15	Evaluation of mood recognition under various lighting conditions Success criteria points #6, #10 (Mood recognition based on facial expressions) (Accuracy)	Evaluating the accuracy of mood recognition under various lighting conditions to ensure the technology is versatile and reliable in different environments, essential for a feature that relies on visual cues for emotion detection.	Attempt to identify different moods in low light, normal light, and very bright light. Evaluate if the AI is able to distinguish between moods such as happy/sad/angry	Boundary	In low light, emotion recognition may fail or output incorrect emotions due to a lack of clarity in the image		
16	Ensuring app stability and performance during extended use Success criteria points #9, #11 (Responsivenes s - 20fps consistent feed) (Ease of use)	Monitoring the application's stability and performance during extended use to identify any potential memory leaks, slowdowns, or crashes that could detract from the user experience, aiming for robustness and reliability over time.	Run the application for a long time, preferably over 10 minutes, with debugging information on screen, monitoring device temperature, memory usage and battery usage.	Boundary	Application should manage resources efficiently, without any memory leaks or bottlenecks.		
17	Accuracy of text captions generated for captured images Success criteria points #8, #10 (Image capture and	Assessing the accuracy of text captions generated for captured images to ensure they are contextually relevant and precise, enhancing the value of the app's image recognition and	Take a wide variety of different images in various environments, evaluating the response and accuracy of the captions	Valid	Captions should remain fairly accurate, describing a wide range of given scenarios.		

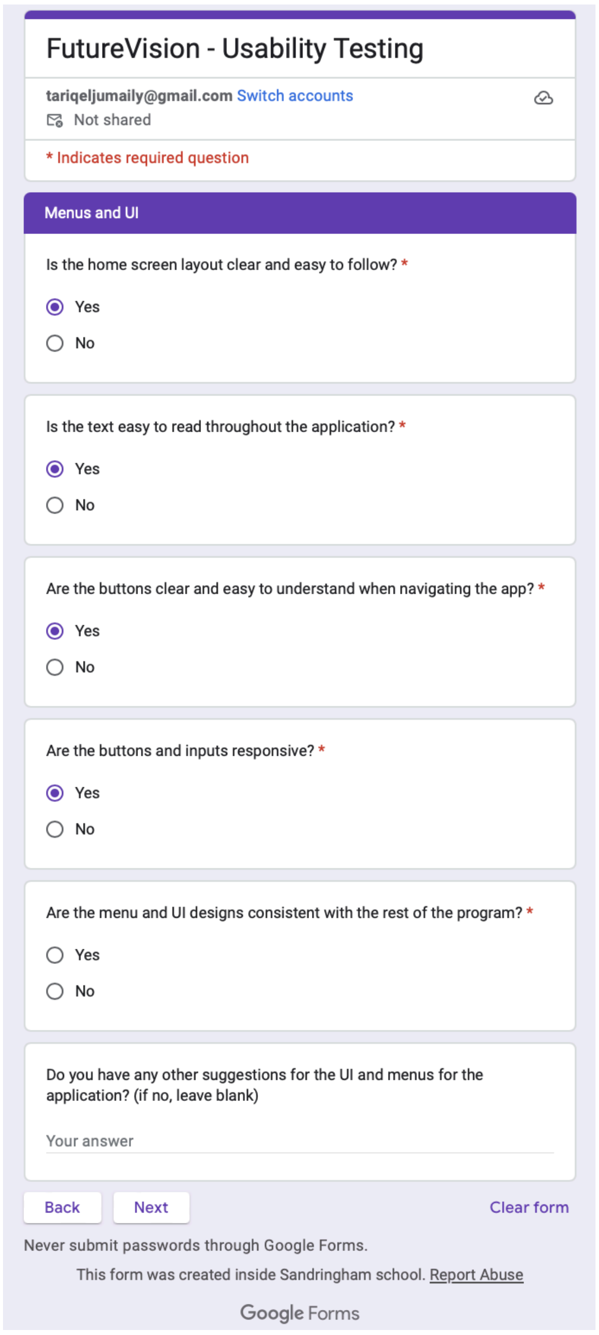
	captioning) (Accuracy)	captioning capabilities for educational, accessibility, or entertainment purposes.					
18	User experience and interface consistency across different devices Success criteria points #2, #11 (User interface design) (Ease of use)	Verifying the user experience and interface consistency across different devices, ensuring that the application provides a seamless and uniform experience regardless of screen size or resolution, catering to a wide range of users.	Build and test on a variety of different iPhones with different processing power and resolutions, evaluating the UI and scaling for each device. (e.g iPhone 7, iPhone 11, iPhone 5)	Valid	Application UI should scale appropriately depending on size and resolution of device.		
19	Functionality and accuracy of the facial recognition feature in crowded scenes Success criteria points #3, #10 (Identification of people in camera's view/facial recognition) (Accuracy)	Testing the functionality and accuracy of the facial recognition feature in crowded scenes to evaluate its ability to distinguish and analyse multiple faces simultaneously, crucial for real-world applicability and user trust.	Prepare an image of a large crowd with visible faces and asses the functionality when the AI is given large amounts of data	Boundary	AI should be able to recognise the majority of the faces in the image, however may reach its limit and only detect up to a specific amount.		
20	Evaluation of app's adaptability to different screen sizes and resolutions	Assessing the app's adaptability to various screen sizes and resolutions ensures a consistent and optimal user	Build and test on a variety of different iPhones with different sizes and resolutions. (e.g iPhone 7, iPhone 11, iPhone 5)	Valid	Application should scale appropriately depending on size and resolution of		

	Success criteria points #2, #11 (User interface design) (Ease of use)	experience across different devices.			device.		
21	Testing the app's performance and functionality without internet connectivity Success criteria points #1, #11 (Camera functionality when user opens app) (Ease of use)	Verifying the app's functionality without internet connectivity ensures critical features remain accessible offline, enhancing usability and reliability.	Turn off device WiFi and evaluate useability and features. In particular, ensure the image captioning is turned off	Erroneous and Valid	Image captioning button disappears and remaining features work perfectly.		
22	Testing voice pitch adjustment in text-to-speech for user personalization Success criteria points #7, #12 (Text-to-speech output for processed attributes) (Variable text-to-speech voice pitch/volume/speed)	Assessing voice pitch adjustment in text-to-speech validates personalization features.	In the settings page, adjust the TTS slider, type in sample text and press 'test' to hear the audio.	Valid	Users can tailor audio outputs to their preferences and test this.		
23	Responsiveness and	Evaluating the UI's responsiveness and	Within the object detection, or emotion	Boundary	During peak processing		

	effectiveness of the user interface during peak processing tasks Success criteria points #2, #9, #11 (User interface design) (Responsivenes s - 20fps consistent feed) (Ease of use)	effectiveness during peak processing tasks ensures the application remains user-friendly and efficient under heavy load conditions.	detection script, include many faces/objects which creates many bounding boxes and evaluate the responsiveness.		tasks, the app should slow down and reduce in responsiveness due to a bottleneck in system resources		
24	Test the text-to-speech feature with the maximum and minimum allowed pitch, volume, and speed settings. Success criteria points: #7, #12 (Text-to-speech output for processed attributes) (Variable text-to-speech voice pitch/volume/s peed)	Evaluate the limits of text-to-speech customization settings to ensure that extreme values do not cause unintelligibility or application errors.	Within the settings page, turn the slider to the maximum and minimum value, then test the audio.	Boundary	Audio should be clear and easy to understand, at an acceptable volume.		

Useability Testing Plan

I have also planned a questionnaire that I will send to 5 users based on the target audience, including my stakeholders:



FutureVision - Usability Testing

tariqeljumaily@gmail.com Switch accounts

Not shared

* Indicates required question

Menus and UI

Is the home screen layout clear and easy to follow? *

[✓ Yes] No

Is the text easy to read throughout the application? *

[✓ Yes] No

Are the buttons clear and easy to understand when navigating the app? *

[✓ Yes] No

Are the buttons and inputs responsive? *

[✓ Yes] No

Are the menu and UI designs consistent with the rest of the program? *

Yes No

Do you have any other suggestions for the UI and menus for the application? (if no, leave blank)

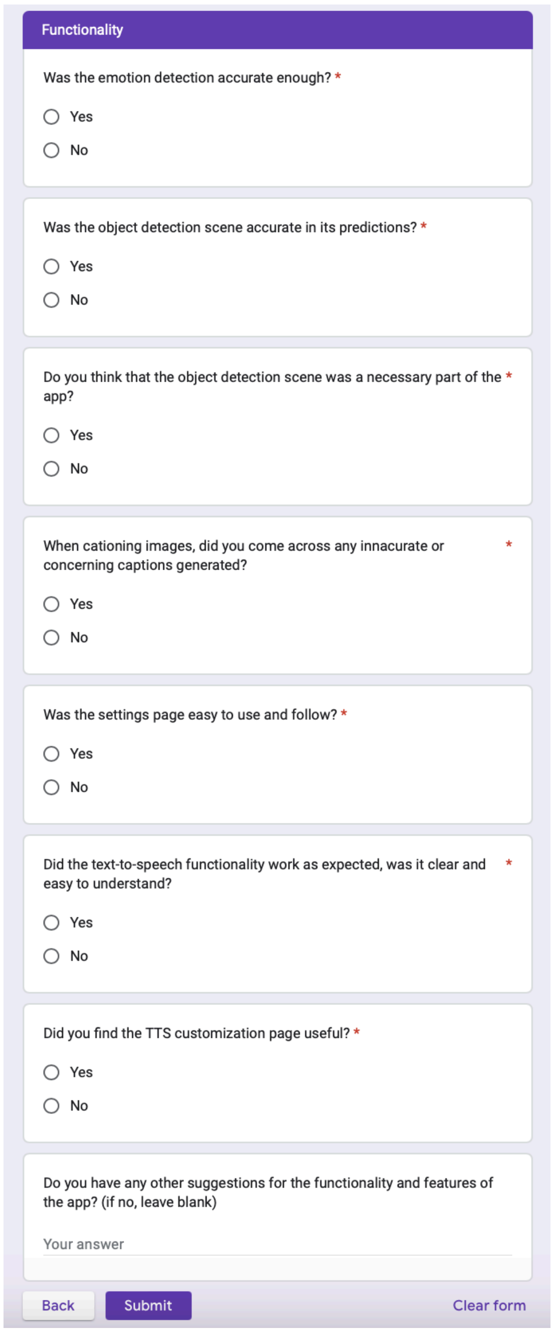
Your answer

Back Next Clear form

Never submit passwords through Google Forms.

This form was created inside Sandringham school. Report Abuse

Google Forms



Functionality

Was the emotion detection accurate enough? *

Yes No

Was the object detection scene accurate in its predictions? *

Yes No

Do you think that the object detection scene was a necessary part of the app? *

Yes No

When cationing images, did you come across any innacurate or concerning captions generated? *

Yes No

Was the settings page easy to use and follow? *

Yes No

Did the text-to-speech functionality work as expected, was it clear and easy to understand? *

Yes No

Did you find the TTS customization page useful? *

Yes No

Do you have any other suggestions for the functionality and features of the app? (if no, leave blank)

Your answer

Back Submit Clear form

Iterative Development

Iteration 1	Iteration 2	Iteration 3
• Device Camera input • Convert Camera texture to be used in OpenCV format • Import OpenCV • Import OpenCV library of pre-trained AI models • Set up face-tracking • Draw bounding box • Emotion Recognition AI import • Change Camera Button • Camera/UI positioning for different devices • Face counter	• Add Age/Gender detection • Object detection system system also (new scene) (yoloV2 lightweight) • Variable confidence metre • Button to send an image to the Python Flask server for Image captioning AI	• Custom image import • Voice commands? • UI Overhaul • Text-To-Speech Button

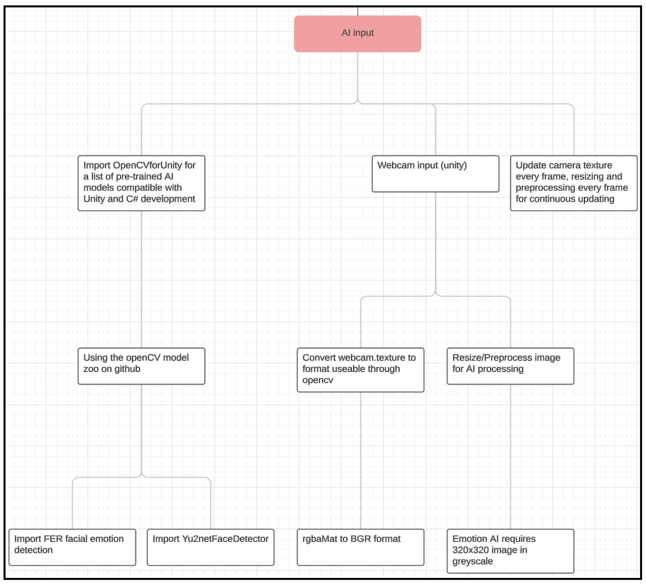
I am going to use the table above to split my prototype 1 into 3 iterations, starting with the key, main features that will be the defining skeleton of my project, and slowly adding features and parts to the project that will be key for later on during the development. Features in Green are considered 'essential', features in Orange are considered 'Key but not needed', and features in Red are considered 'optional (dependent on time constraints)'. Any features that I would like to include, that may not make it into prototype 1, will be considered for future prototypes

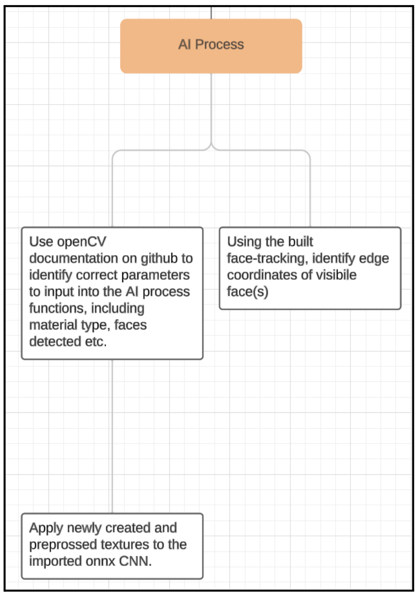
Iteration 1 - *Date 25/07/2023*

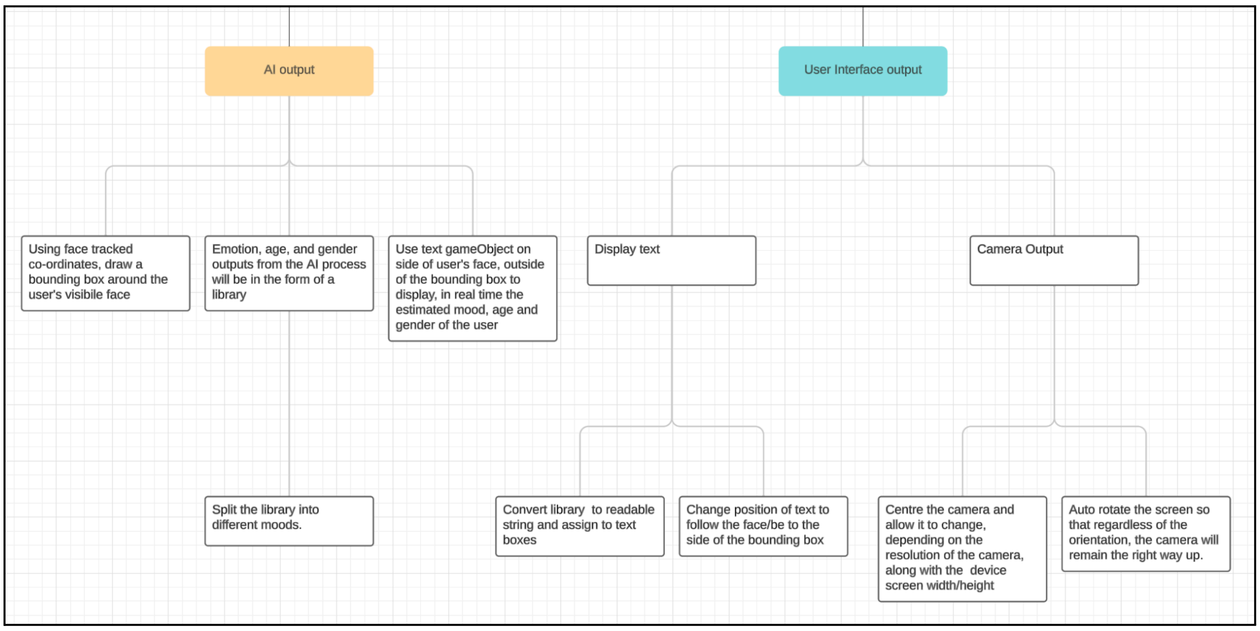
Aims for this iteration

The aim for this iteration is to develop the 'skeleton'/backend of the project, this involves importing and fine-tuning the built-in AI models that are included in the OpenCVforUnity package. By the end of this iteration, I want an adequately positioned central plane that displays a camera feed, with functional face-tracking and emotion-detection ability, along with the option to change the camera type (e.g., front/back). This means that I needed to change my top-down design to accommodate this change so that I could continue with the development of this project smoothly.

Below are the functions that I will be working on for this portion of the project:


AI input
Import OpenCVforUnity for a list of pre-trained AI models compatible with Unity and C# development
Webcam input (unity)
Update camera texture every frame, resizing and preprocessing every frame for continuous updating
Using the openCV model zoo on github
Convert webcam.texture to format useable through opencv
Resize/Preprocess image for AI processing
Import FER facial emotion detection
Import Yu2netFaceDetector
rgbaMat to BGR format
Emotion AI requires 320x320 image in greyscale


AI Process
Use openCV documentation on github to identify correct parameters to input into the AI process functions, including material type, faces detected etc.
Using the built face-tracking, identify edge coordinates of visibile face(s)
Apply newly created and prepressed textures to the imported onnx CNN.


AI output
User Interface output
Using face tracked co-ordinates, draw a bounding box around the user's visibile face
Emotion, age, and gender outputs from the AI process will be in the form of a library
Use text gameObject on side of user's face, outside of the bounding box to display, in real time the estimated mood, age and gender of the user
Display text
Camera Output
Split the library into different moods.
Convert library to readable string and assign to text boxes
Change position of text to follow the face/be to the side of the bounding box
Centre the camera and allow it to change, depending on the resolution of the camera, along with the device screen width/height
Auto rotate the screen so that regardless of the orientation, the camera will remain the right way up.

The functions that are included in the top-down design, make up the key aspects of this project, which include the input, process and output, which is critical to a visualization AI. The other iterations throughout this project will rely heavily on the functions included in iteration 1, which is why it is important that they are completed and fine-tuned first.

For this iteration, I will be using basic unity UI, as the design and style of the program will be implemented in later iterations throughout the project, due to its lower level of priority. Additionally, this includes the age/gender recognition AI model. I may later incorporate this as a separate model, or in a later prototype, import or train a custom model myself that includes the emotion, age and gender functionality all in one, to simplify the process, rather than use 2 separate models for this.

Summary of aims:

- Set up/import OpenCVforUnity
    - Use OpenCV DNN (deep neural network)
    - Import DNN models (face detection, emotion recognition)
- Camera input
    - Ensure the camera is centred to the screen and the aspect ratio is maintained, with variable sizing, dependent on the screen/camera type.
    - Set up textures and colour Mats through the camera view.
- Initialize pre-trained models through OpenCVLibrary
    - FaceDetector
    - FER (FacialEmotionRecogntion)
- Process Camera texture for AI processing.
- Output detected faces using bounding boxes.
- Output emotions with a bounding box.
- Change camera button.

Functionality that the prototype will have:

Within this iteration, I hope to achieve the following parts of my success criteria:
(1, 3, 6, 9, and 10)

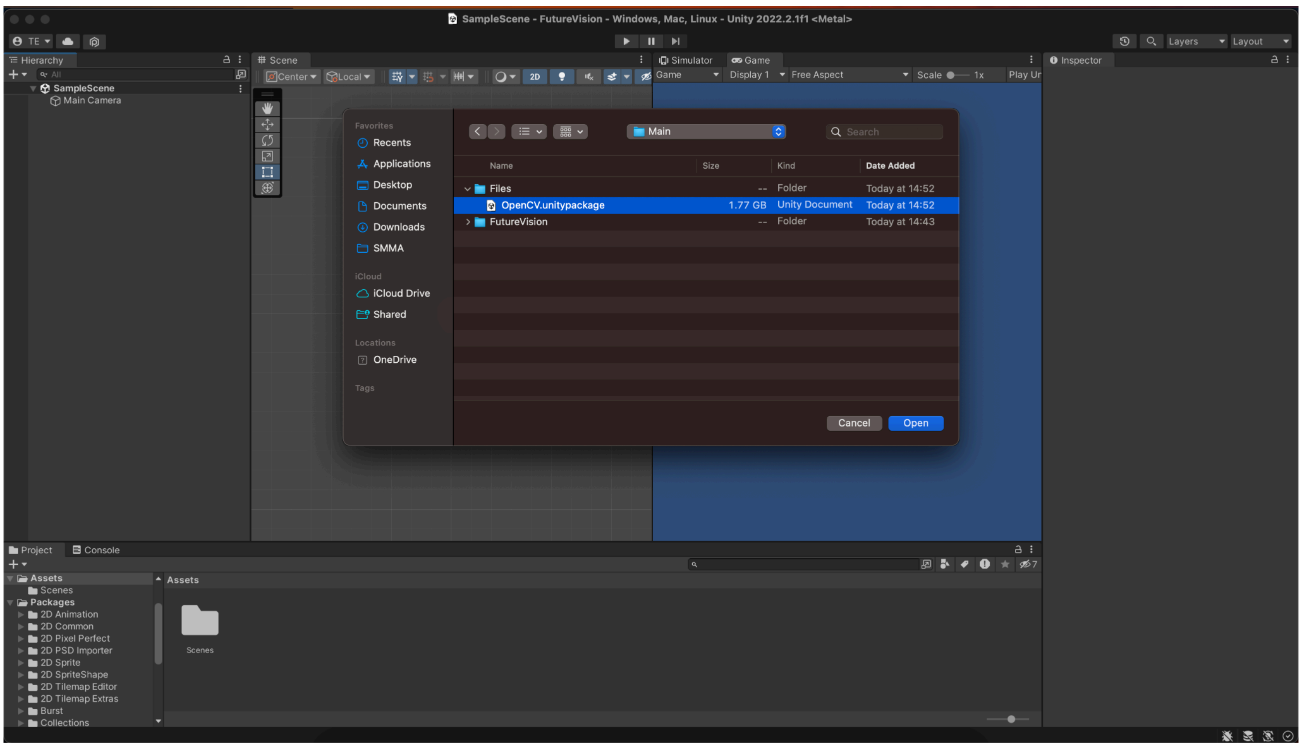
1	Camera functionality when user opens app	The camera feed is fundamental for the app to function as it provides the visual input for all processing. Without it, none of the following features can be achieved.	1
3	Identification of people in camera's view	This is crucial for the application as it sets the basis for further analysis such as mood detection, age prediction, and gender identification.	1
6	Mood recognition based on facial expressions	Identifying the mood of the person in view can help visually impaired users understand non-verbal cues and respond appropriately.	1
9	Responsiveness (20fps consistent feed)	The application should operate with minimal lag between capturing an image and generating results. A quick response time is critical for the user's experience. Due to the time limits and computational bottlenecks, I may result in single frame captures where the inputs are processed slower and outputted afterwards (almost like taking a photo and analysing what's inside it)	2
10	Accuracy	The output (age, gender, mood, image caption) should be as accurate as possible to provide the user with reliable information about their surroundings. Although due to the time pressure and computational power required, I may not be able to get very accurate results all of the time.	2

<u>Extra functionality to be included in this iteration:</u>

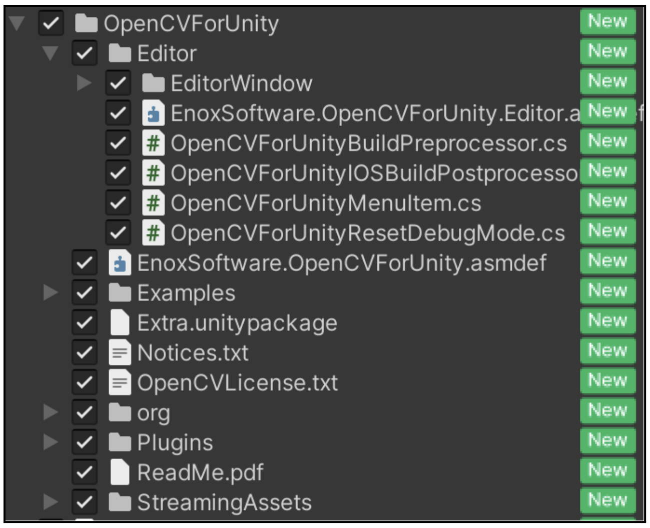
- Ability to cycle through onboard device cameras
- Auto-Rotate functionality, allowing the user to use the app comfortably, however they please
- Face counter, a UI text that displays the number of detected faces on the screen
- Confidence view to show the user the level of accuracy/confidence the AI has detected

Annotated code screenshots with description

Initial set-up:


SampleScene - FutureVision - Windows, Mac, Linux - Unity 2022.2.1f1 <Metal>

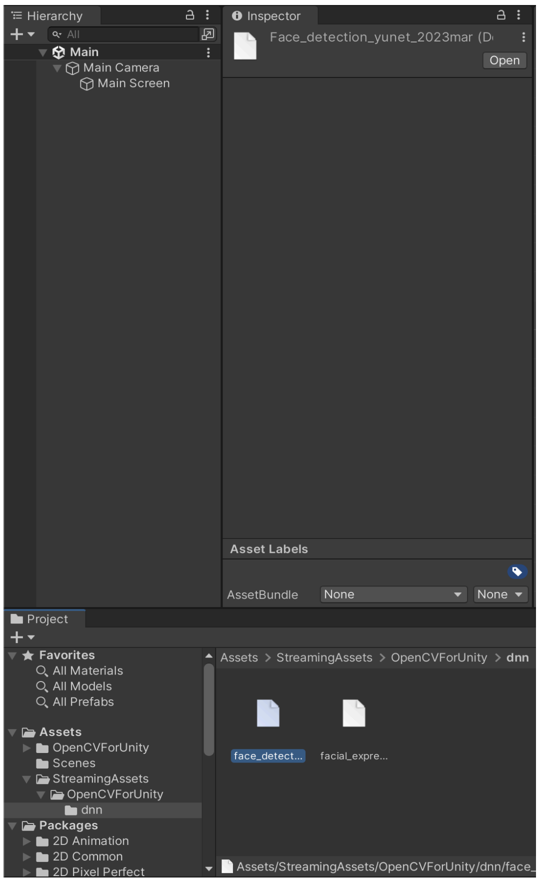
Name	Size	Kind	Date Added
Files	--	Folder	Today at 14:52
OpenCV.unitypackage	1.77 GB	Unity Document	Today at 14:52
FutureVision	--	Folder	Today at 14:43


```
✓ OpenCVForUnity                                   New
  ✓ Editor                                         New
    ✓ EditorWindow                                 New
    ✓ EnoxSoftware.OpenCVForUnity.Editor.a         New
    ✓ OpenCVForUnityBuildPreprocessor.cs           New
    ✓ OpenCVForUnityIOSBuildPostprocesso           New
    ✓ OpenCVForUnityMenuItem.cs                    New
    ✓ OpenCVForUnityResetDebugMode.cs              New
  ✓ EnoxSoftware.OpenCVForUnity.asmdef             New
  ✓ Examples                                       New
  ✓ Extra.unitypackage                             New
  ✓ Notices.txt                                    New
  ✓ OpenCVLicense.txt                              New
  ✓ org                                            New
  ✓ Plugins                                        New
  ✓ ReadMe.pdf                                     New
  ✓ StreamingAssets                                New
```

Importing OpenCV is key as this is going to be the backbone to my project and the primary library used to execute and process AI functions and models in C#, instead of python or C++.

OpenCV comes with many unnecessary packages, examples and models, many of which we can delete later, or choose to not import them manually.

For now, I am going to import the whole library for simplicity.



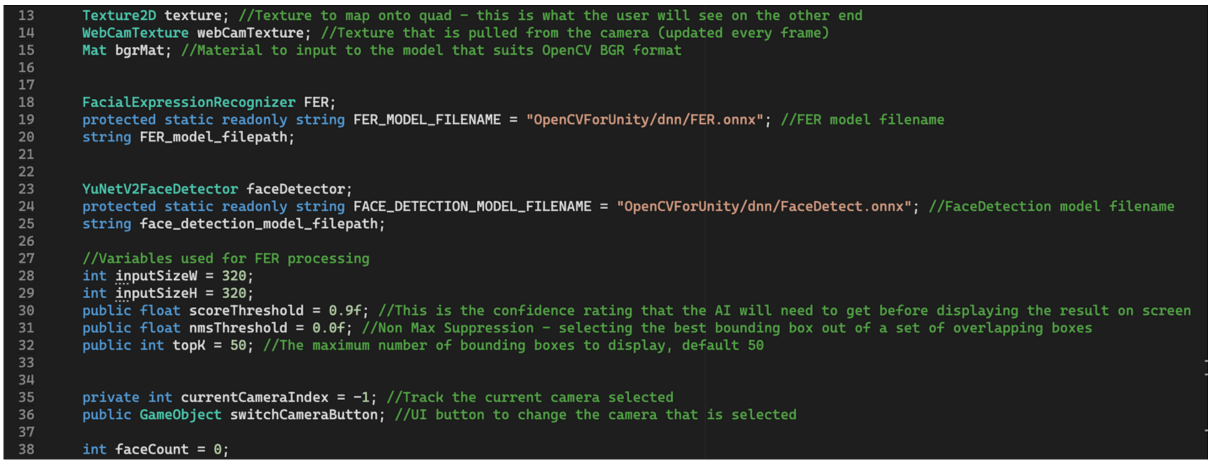
2 .onnx files are imported into the 'streamingAssets' folder of the project. These are assets that are loaded when needed from unity. The files that are imported are pre-trained openCV AI models, each designed for a separate function, one is used for face recognition, while the other is used for emotion recognition. Without these models, the AI will have no reference as to what it is trying to identify or predict.

These models take an input that is a texture of size 320x320 with greyscale color. This allows quick and accurate processing. This means that later, we will need to 'pre-process' our camera image to match this criteria for the project.

Additionally, I imported a quad named "main screen". This quad currently has no texture. However, when the main camera starts rendering, a texture will be mapped onto the quad allowing a direct view of the camera and will show all the outputs on screen. I have placed this as a child of the main camera.

Variable initialization:

EmotionDetectionScript.cs



```
    Texture2D texture; //Texture to map onto quad - this is what the user will see on the other end
    WebCamTexture webCamTexture; //Texture that is pulled from the camera (updated every frame)
    Mat bgrMat; //Material to input to the model that suits OpenCV BGR format


    FacialExpressionRecognizer FER;
    protected static readonly string FER_MODEL_FILENAME = "OpenCVForUnity/dnn/FER.onnx"; //FER model filename
    string FER_model_filepath;


    YuNetV2FaceDetector faceDetector;
    protected static readonly string FACE_DETECTION_MODEL_FILENAME = "OpenCVForUnity/dnn/FaceDetect.onnx"; //FaceDetection model filename
    string face_detection_model_filepath;

    //Variables used for FER processing
    int inputSizeW = 320;
    int inputSizeH = 320;
    public float scoreThreshold = 0.9f; //This is the confidence rating that the AI will need to get before displaying the result on screen
    public float nmsThreshold = 0.0f; //Non Max Suppression - selecting the best bounding box out of a set of overlapping boxes
    public int topK = 50; //The maximum number of bounding boxes to display, default 50


    private int currentCameraIndex = -1; //Track the current camera selected
    public GameObject switchCameraButton; //UI button to change the camera that is selected

    int faceCount = 0;
```

texture - This Texture2D variable will be used towards the camera initialization. The *webcamTexture* will be mapped onto the *texture* variable. This is going to be updated every frame in *update()* and will be attached to the quad gameObject. This means that the webcam's view will be visible on the quad.

bgrMat - OpenCV uses BGR material format, which is not the same as unity's mat format. This means that at some point, I will need to convert the unity material to an OpenCV compatible bgr material

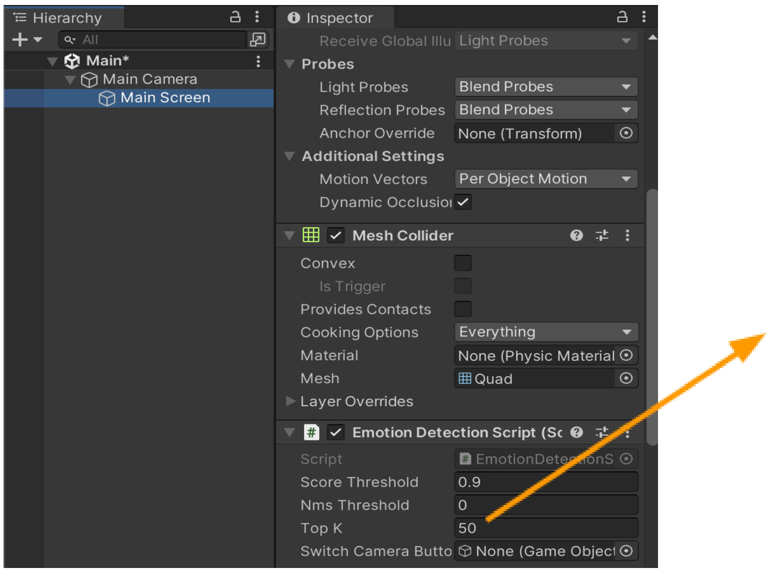
FER/faceDetector - This is a separate script included in openCV that will take the inputs and parameters, pass them through the pre-trained model, and form an output.

scoreThreshold - This is the float variable that will determine how confident the model must be in order to produce a valid output for Face Detection and FER. 0.9f is the default value - this means that the model is 90% confident in its analysis.

nmsThreshold - non max suppression. This is a technique used in numerous computer vision tasks. It is a class of algorithms to select one bounding box out of many overlapping. The higher the variable score, the more accurate the bounding box will be. Due to face Detection being fairly basic, we can keep this score relatively low, to keep processing speeds high. (src: https://learnopencv.com/non-maximum-suppression-theory-and-implementation-in-pytorch/)

currentCameraIndex - many devices have more than one camera, During the initialization of the program, the script will detect the number of cameras on the device and assign them all an index. *currentCameraIndex* will keep track of which camera is currently turned on. Many devices also have unnecessary cameras, such as 0.5x zoom or telephoto, that need to be indexed and skipped when cycling through the available cameras.

faceCount – integer that keeps track of the number of faces detected on screen

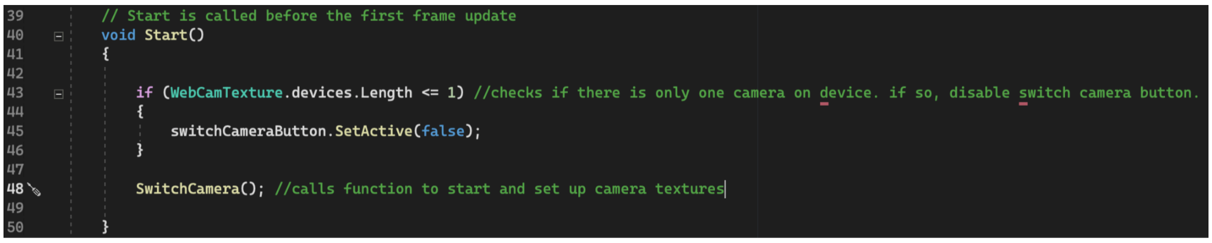


```
Receive Global Illu     Light Probes
Probes
  Light Probes          Blend Probes
  Reflection Probes     Blend Probes
  Anchor Override       None (Transform)
Additional Settings
  Motion Vectors        Per Object Motion
  Dynamic Occlusioı     ✓
Mesh Collider
  Convex
    Is Trigger
  Provides Contacts
  Cooking Options       Everything
  Material              None (Physic Material
  Mesh                  Quad
  Layer Overrides
Emotion Detection Script (Sc
  Script                EmotionDetectionS
  Score Threshold       0.9
  Nms Threshold         0
  Top K                 50
  Switch Camera Butto   None (Game Object
```

Public variables are displayed in the inspector as shown. This will allow me to change the values of the variables easily throughout development and testing.

Start():

EmotionDetectionScript.cs



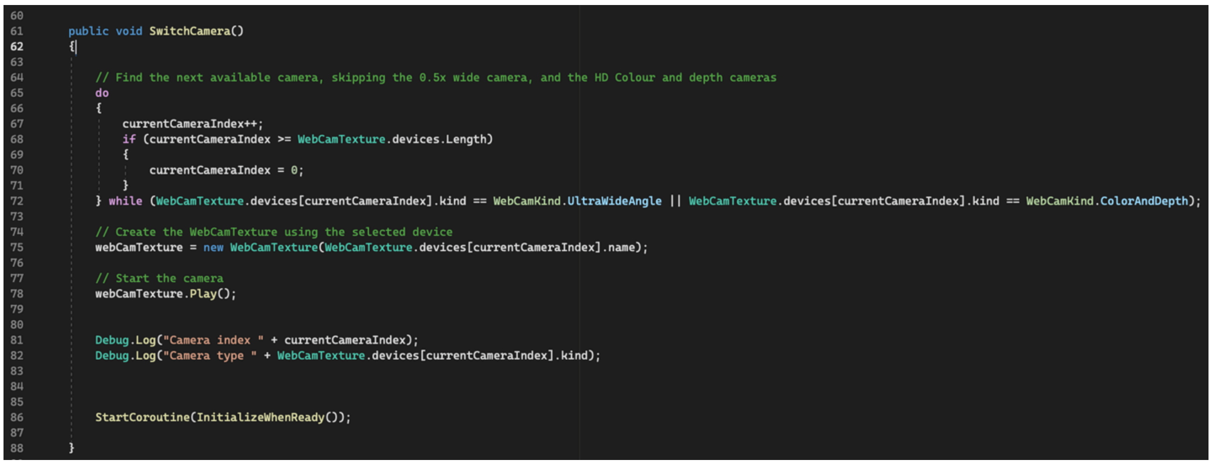
```
    // Start is called before the first frame update
    void Start()
    {

        if (WebCamTexture.devices.Length <= 1) //checks if there is only one camera on device. if so, disable switch camera button.
        {
            switchCameraButton.SetActive(false);
        }

        SwitchCamera(); //calls function to start and set up camera textures

    }
```

As soon as the program begins, the script checks whether the device has more than one camera type. If so, it will keep the ‘switchCameraButton’ active, otherwise it will turn it off. This is because the button will be useless since there are no other cameras to switch to and so the function will not need to be called.

After this check, the SwitchCamera() function will be called - details below.

SwitchCamera():

EmotionDetectionScript.cs



```

    public void SwitchCamera()
    {

        // Find the next available camera, skipping the 0.5x wide camera, and the HD Colour and depth cameras
        do
        {
            currentCameraIndex++;
            if (currentCameraIndex >= WebCamTexture.devices.Length)
            {
                currentCameraIndex = 0;
            }
        } while (WebCamTexture.devices[currentCameraIndex].kind == WebCamKind.UltraWideAngle || WebCamTexture.devices[currentCameraIndex].kind == WebCamKind.ColorAndDepth);

        // Create the WebCamTexture using the selected device
        webCamTexture = new WebCamTexture(WebCamTexture.devices[currentCameraIndex].name);

        // Start the camera
        webCamTexture.Play();


        Debug.Log("Camera index " + currentCameraIndex);
        Debug.Log("Camera type " + WebCamTexture.devices[currentCameraIndex].kind);



        StartCoroutine(InitializeWhenReady());

    }
```

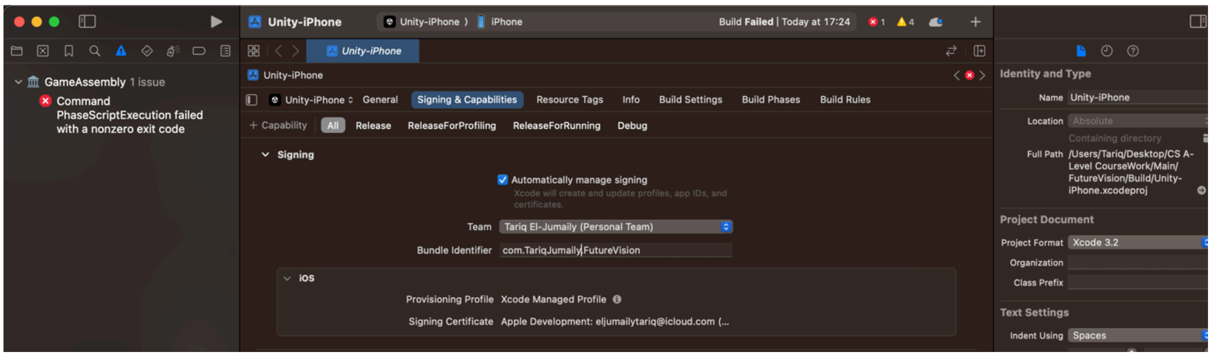
The function will cycle through the available cameras, skipping the ones labeled “UltraWideAngle” and “ColorAndDepth”. The UltraWideAngle on iOS is commonly known as the 0.5x zoom camera. Furthermore, the ColorAndDepth camera is the standard camera at a higher resolution and colour depth. We can omit this – this is because we do not need a high-resolution output due to the nature of this project, although I am going to consider adding it in a settings page, based on the user’s display preferences.

After selecting the next available camera, the webCamTexture will be initialized and linked to the texture that the webcam can see. Then we start the camera using webCamTexture.Play(); - This means that camera is turned on and it’s texture is being captured.

At the end of the function, a debug statement is included for identifying the camera type and index and then the “*InitializeWhenReady()*” Coroutine is started.

At this point, I was met with an error while building the app in unity, prompting me to update Xcode for compatibility.

After updating Xcode, my app no longer builds and shows a GameAssembly issue.



Unity-iPhone | Unity-iPhone 〉 iPhone | Build **Failed** | Today at 17:24 | 1 | 4

GameAssembly 1 issue

Command PhaseScriptExecution failed with a nonzero exit code

Unity-iPhone

Unity-iPhone | General | **Signing & Capabilities** | Resource Tags | Info | Build Settings | Build Phases | Build Rules

+ Capability | **All** | Release | ReleaseForProfiling | ReleaseForRunning | Debug

Signing

✓ Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.

Team: Tariq El-Jumaily (Personal Team)
Bundle Identifier: com.TariqJumaily.FutureVision

iOS
Provisioning Profile: Xcode Managed Profile
Signing Certificate: Apple Development: eljumailytariq@icloud.com (...

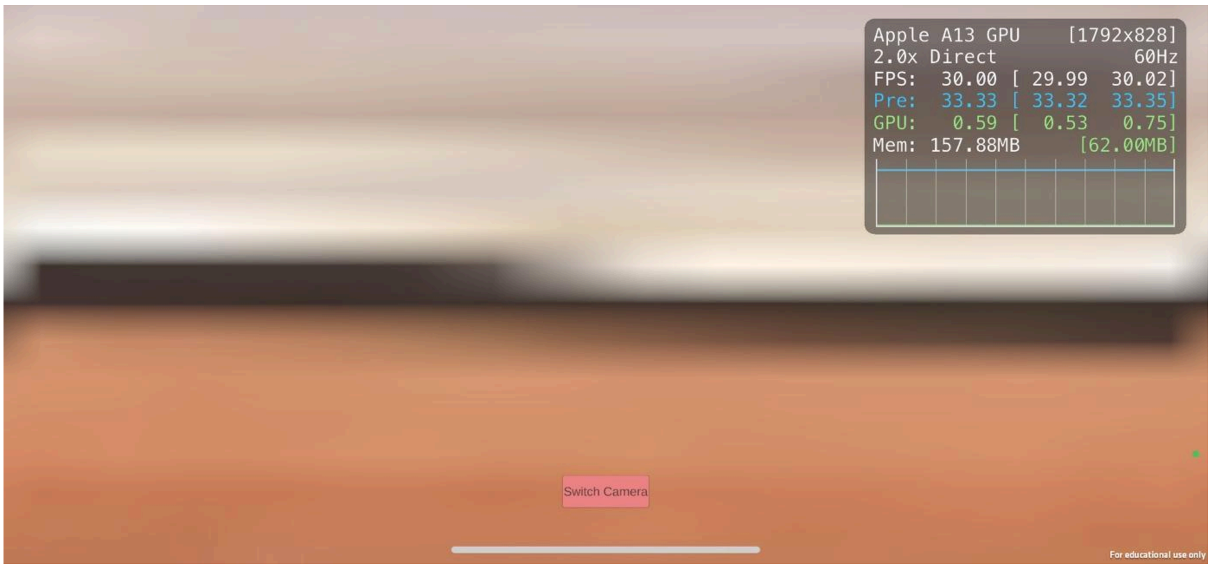
Identity and Type
Name: Unity-iPhone
Location: Absolute
Containing directory
Full Path: /Users/Tariq/Desktop/CS A-Level CourseWork/Main/FutureVision/Build/Unity-iPhone.xcodeproj

Project Document
Project Format: Xcode 3.2
Organization:
Class Prefix:

Text Settings
Indent Using: Spaces

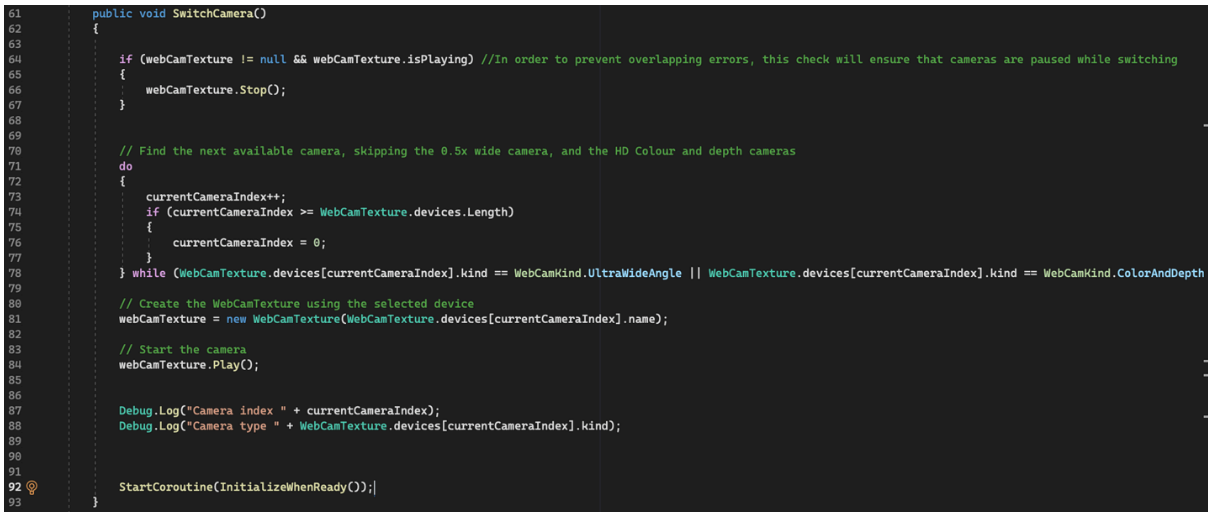
This PhaseScriptExecution error was caused due to a mismatch in versions of Xcode and unity. After upgrading my Unity editor to version 2022.3.10f1, my app began to build successfully.

When running the app to test the camera functionality, on an iphone, I get an error where the front and rear camera are overlapping to cause a visual error, where it is impossible to see a clear image as shown below:


```
Apple A13 GPU     [1792x828]
2.0x Direct             60Hz
FPS:   30.00 [ 29.99   30.02]
Pre:   33.33 [ 33.32   33.35]
GPU:    0.59 [  0.53    0.75]
Mem:  157.88MB       [62.00MB]
```

In order to fix this, I need to add functionality that will ensure one camera turns off before the next one is initialised.

EmotionDetectionScript.cs

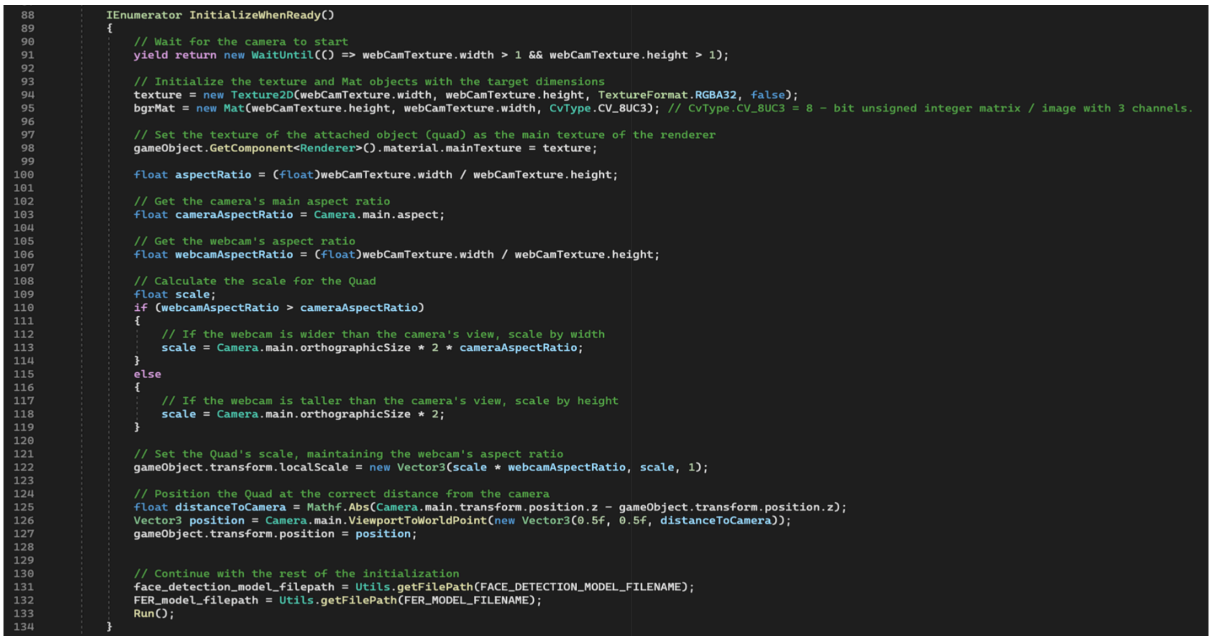


```csharp
public void SwitchCamera()
{

    if (webCamTexture != null && webCamTexture.isPlaying) //In order to prevent overlapping errors, this check will ensure that cameras are paused while switching
    {
        webCamTexture.Stop();
    }


    // Find the next available camera, skipping the 0.5x wide camera, and the HD Colour and depth cameras
    do
    {
        currentCameraIndex++;
        if (currentCameraIndex >= WebCamTexture.devices.Length)
        {
            currentCameraIndex = 0;
        }
    } while (WebCamTexture.devices[currentCameraIndex].kind == WebCamKind.UltraWideAngle || WebCamTexture.devices[currentCameraIndex].kind == WebCamKind.ColorAndDepth

    // Create the WebCamTexture using the selected device
    webCamTexture = new WebCamTexture(WebCamTexture.devices[currentCameraIndex].name);

    // Start the camera
    webCamTexture.Play();


    Debug.Log("Camera index " + currentCameraIndex);
    Debug.Log("Camera type " + WebCamTexture.devices[currentCameraIndex].kind);



    StartCoroutine(InitializeWhenReady());
}
```

This new updated function starts by checking if any webcams are playing and if the webcamTexture variable has a placeholder. If there are, then it will stop the camera. This is important as without this, there will be significant errors during camera switching, which will cause the horizontal lines of one camera and vertical lines of the next camera to overlap, so we need to turn them off, before turning on the next camera and assigning the texture.

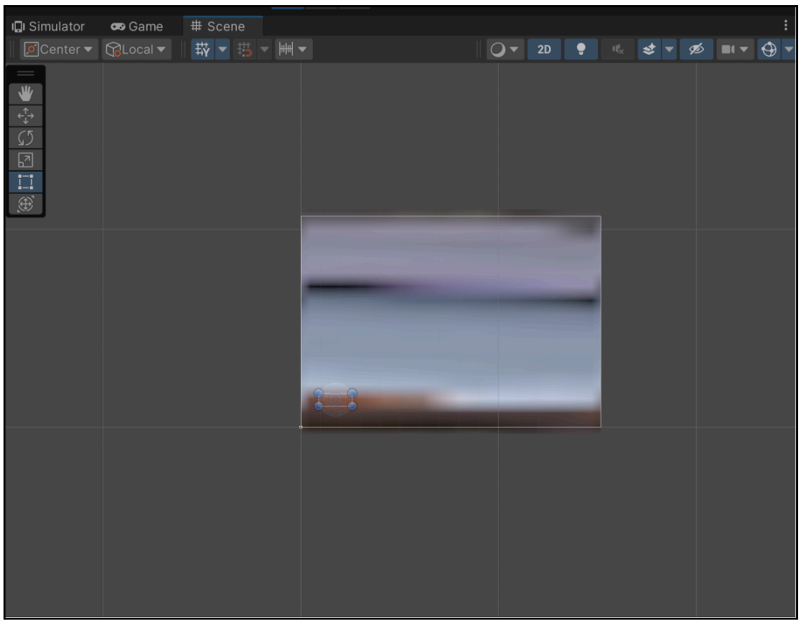
InitializeWhenReady():

EmotionDetectionScript.cs

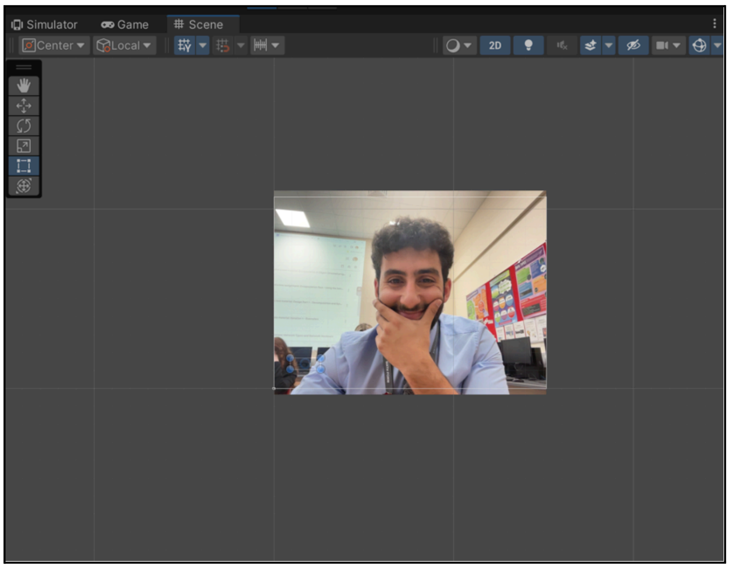


```csharp
IEnumerator InitializeWhenReady()
{
    // Wait for the camera to start
    yield return new WaitUntil(() => webCamTexture.width > 1 && webCamTexture.height > 1);

    // Initialize the texture and Mat objects with the target dimensions
    texture = new Texture2D(webCamTexture.width, webCamTexture.height, TextureFormat.RGBA32, false);
    bgrMat = new Mat(webCamTexture.height, webCamTexture.width, CvType.CV_8UC3); // CvType.CV_8UC3 = 8 - bit unsigned integer matrix / image with 3 channels.

    // Set the texture of the attached object (quad) as the main texture of the renderer
    gameObject.GetComponent<Renderer>().material.mainTexture = texture;

    float aspectRatio = (float)webCamTexture.width / webCamTexture.height;

    // Get the camera's main aspect ratio
    float cameraAspectRatio = Camera.main.aspect;

    // Get the webcam's aspect ratio
    float webcamAspectRatio = (float)webCamTexture.width / webCamTexture.height;

    // Calculate the scale for the Quad
    float scale;
    if (webcamAspectRatio > cameraAspectRatio)
    {
        // If the webcam is wider than the camera's view, scale by width
        scale = Camera.main.orthographicSize * 2 * cameraAspectRatio;
    }
    else
    {
        // If the webcam is taller than the camera's view, scale by height
        scale = Camera.main.orthographicSize * 2;
    }

    // Set the Quad's scale, maintaining the webcam's aspect ratio
    gameObject.transform.localScale = new Vector3(scale * webcamAspectRatio, scale, 1);

    // Position the Quad at the correct distance from the camera
    float distanceToCamera = Mathf.Abs(Camera.main.transform.position.z - gameObject.transform.position.z);
    Vector3 position = Camera.main.ViewportToWorldPoint(new Vector3(0.5f, 0.5f, distanceToCamera));
    gameObject.transform.position = position;


    // Continue with the rest of the initialization
    face_detection_model_filepath = Utils.getFilePath(FACE_DETECTION_MODEL_FILENAME);
    FER_model_filepath = Utils.getFilePath(FER_MODEL_FILENAME);
    Run();
}
```

This coroutine starts by waiting for the webcamTexture to initialize and start. This is because the code execution will be faster than the camera turning on, therefore without the yield WaitUntil(), the program will try to assign and reshape different textures that don't exist, which is an indicator that unity is trying to work on variables that haven't loaded yet or with no data. I found this out when testing the camera functionality. Without line '91', the screen output is very similar as before, which indicates a similar issue:



Due to this error occurring, where the image is not displaying properly, I turned the function into an enumerator and added a 'yield waituntil()' function in order to give the CPU time to catch up with the camera's image, which fixed the problem and outputted the following:



After waiting for the camera to turn on, the main *texture* variable is assigned the camera texture – this is the texture that will be mapped onto the quad which will allow the user to see the camera's view which is what is shown above. The *bgrMat* material is also initialized – this is needed for the OpenCV models to process since we will be using this format. The material is assigned a Cv.Type.CV_8UC3 material type – this is defined as an 8-bit unsigned integer matrix with 3 colour channels (we will assign these channels to be BGR (blue, green, red)).

After initializing the materials and textures, we assign the texture to the attached renderer of the gameObject (the quad).

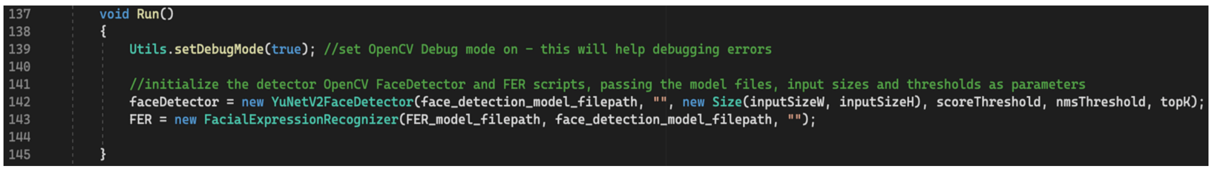
Afterwards, the coroutine will use the width/height of the webcam, with the height/width of the screen to determine the correct positioning and centering of the quad gameObject, to maintain the aspect ratio and re-sizing between different devices.

In the end, the filenames of the onnx models are assigned to the model filepath strings. This is the method the OpenCV recommend to use for loading the models into unity.

The *Run()* function is then called – this is in charge of initializing the AI processing scripts

Run():

EmotionDetectionScript.cs



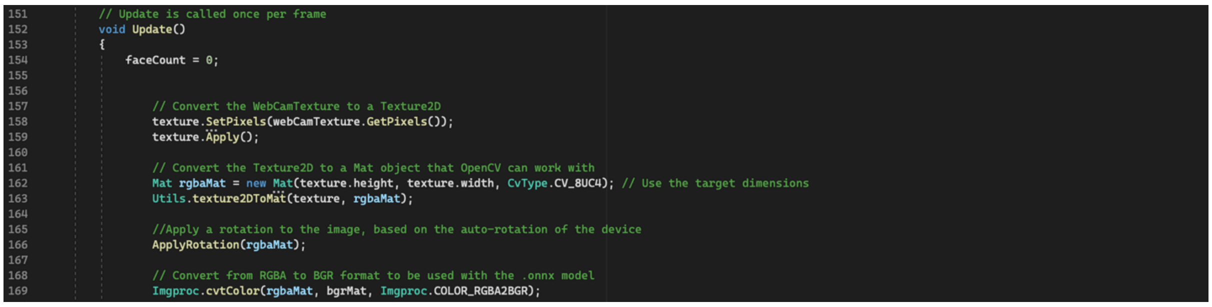
```
void Run()
{
    Utils.setDebugMode(true); //set OpenCV Debug mode on - this will help debugging errors

    //initialize the detector OpenCV FaceDetector and FER scripts, passing the model files, input sizes and thresholds as parameters
    faceDetector = new YuNetV2FaceDetector(face_detection_model_filepath, "", new Size(inputSizeW, inputSizeH), scoreThreshold, nmsThreshold, topK);
    FER = new FacialExpressionRecognizer(FER_model_filepath, face_detection_model_filepath, "");

}
```

The Run() function is fairly simple; turn on debug mode and initialize the OpenCV DNN scripts using the thresholds, input sizes and other parameters that we've passed through so that it is able to execute the models.

The built-in YuNetV2FaceDetector and FacialExpressionRecognizer script manage the bulk of the preprocessing and postprocessing of the inputs.

Update():

EmotionDetectionScript.cs



```
// Update is called once per frame
void Update()
{
    faceCount = 0;


        // Convert the WebCamTexture to a Texture2D
        texture.SetPixels(webCamTexture.GetPixels());
        texture.Apply();

        // Convert the Texture2D to a Mat object that OpenCV can work with
        Mat rgbaMat = new Mat(texture.height, texture.width, CvType.CV_8UC4); // Use the target dimensions
        Utils.texture2DToMat(texture, rgbaMat);

        //Apply a rotation to the image, based on the auto-rotation of the device
        ApplyRotation(rgbaMat);

        // Convert from RGBA to BGR format to be used with the .onnx model
        Imgproc.cvtColor(rgbaMat, bgrMat, Imgproc.COLOR_RGBA2BGR);
```

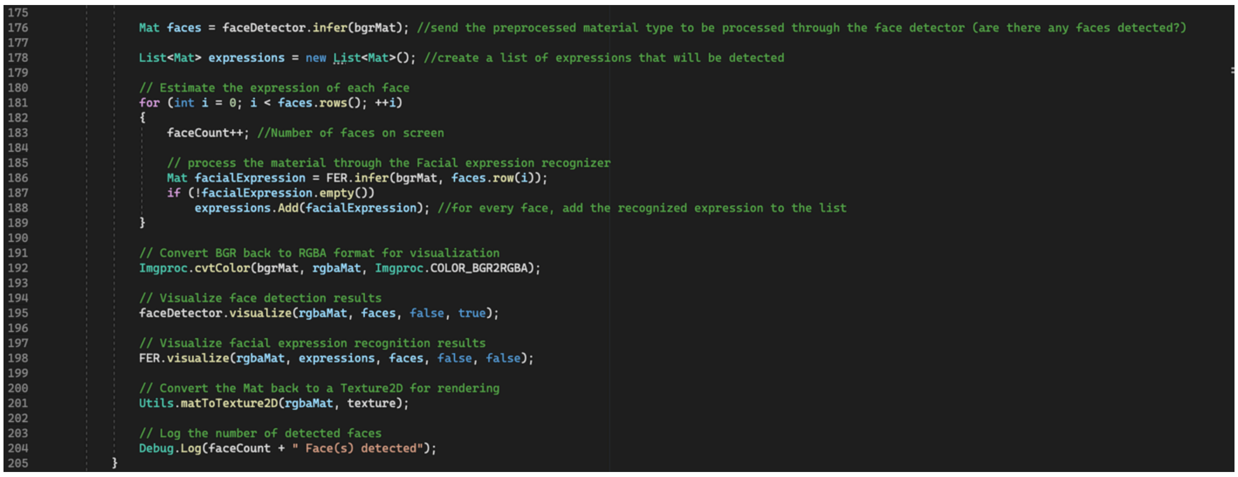
The update function starts by setting the number of detected faces to 0 and then WebCamTexture is converted to a Texture2D so that we can then convert the Texture2D to an RGBA material format – this can then work well with OpenCV. This is done by setting a new material to a variable called *rgbaMat* – this new material would have the height and width of the webcam passed through, and then the material type is selected as a *CvType.CV_8UC4* – this is an OpenCV type material, consisting of and 8-bit unsigned integer with 4 colour channels; RGBA (red, green, blue, alpha)

After converting the materials and textures to be compatible with the models, unity and OpenCV, we apply a rotation matrix, based on the orientation of the device (portrait/landscape) – This will allow us to use the device in any orientation we want, without having the image be flipped or rotated the wrong way.

After the rotation function returns the new material, we convert it to a BGR format in order to be used directly within the .onnx model – most OpenCV AI models require a BGR colour material format.

Update() Continued:

EmotionDetectionScript.cs



```
            Mat faces = faceDetector.infer(bgrMat); //send the preprocessed material type to be processed through the face detector (are there any faces detected?)

            List<Mat> expressions = new List<Mat>(); //create a list of expressions that will be detected

            // Estimate the expression of each face
            for (int i = 0; i < faces.rows(); ++i)
            {
                faceCount++; //Number of faces on screen

                // process the material through the Facial expression recognizer
                Mat facialExpression = FER.infer(bgrMat, faces.row(i));
                if (!facialExpression.empty())
                    expressions.Add(facialExpression); //for every face, add the recognized expression to the list
            }

            // Convert BGR back to RGBA format for visualization
            Imgproc.cvtColor(bgrMat, rgbaMat, Imgproc.COLOR_BGR2RGBA);

            // Visualize face detection results
            faceDetector.visualize(rgbaMat, faces, false, true);

            // Visualize facial expression recognition results
            FER.visualize(rgbaMat, expressions, faces, false, false);

            // Convert the Mat back to a Texture2D for rendering
            Utils.matToTexture2D(rgbaMat, texture);

            // Log the number of detected faces
            Debug.Log(faceCount + " Face(s) detected");
        }
```

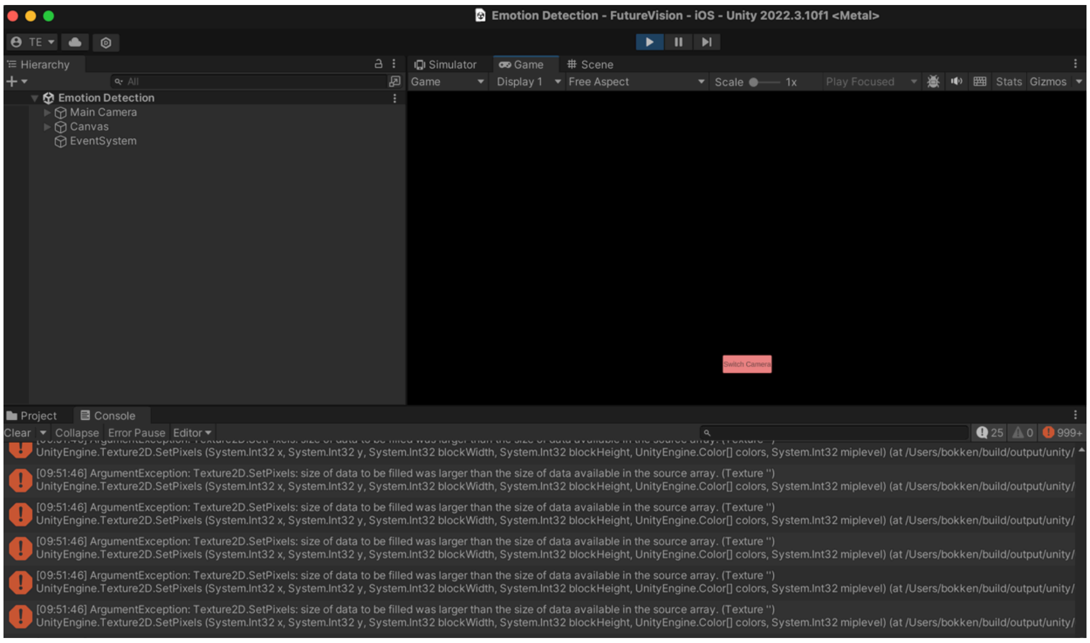
After the conversion to BGR formats, we send the new material through the faceDetector AI model for processing through the *faceDetector.infer()* function – this will return an array of faces that it has detected.

Using this array, we use a count-controlled loop to loop through each face, updating the number of detected faces and then send the BGR material through another AI model – this time we send it through the emotion detector using *FER.infer()* This will return a list of facial expressions, which we add to each corresponding face – this means we are able to have multiple faces have different emotions on the screen at the same time.

We now have faces and emotions in the form of a list and we need to visualise it on screen, the way we do this is by using the included face detection and emotion detection visualisation functions through OpenCV which assist in drawing bounding boxes and labelling the corresponding emotions.

Finally, we print the number of faces detected in the console – in the next iteration, I will assign this a UI text box that will update on-screen.

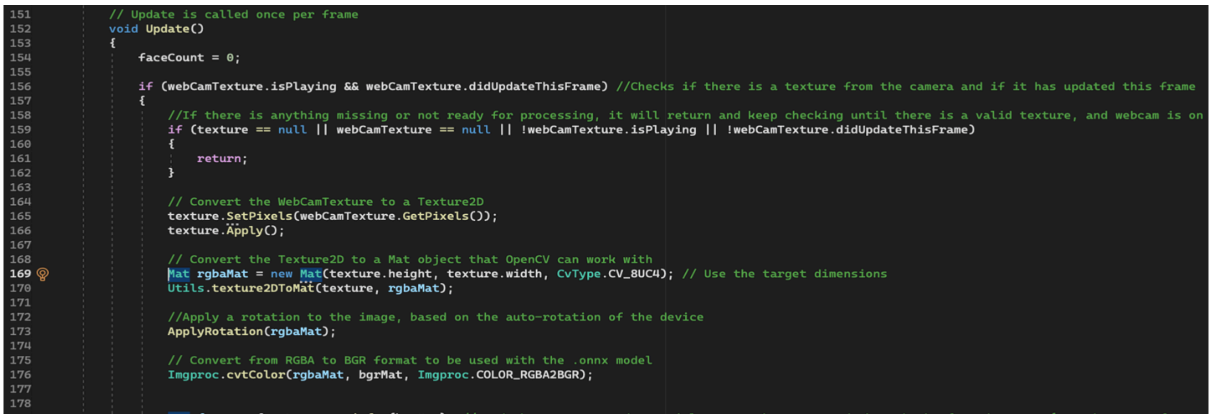
When testing Update() and it's functionality, I notice that when switching cameras, I have a long delay and the application crashes due to thousands of argument exception errors as seen below:



```
UnityEngine.Texture2D.SetPixels (System.Int32 x, System.Int32 y, System.Int32 blockWidth, System.Int32 blockHeight, UnityEngine.Color[] colors, System.Int32 miplevel) (at /Users/bokken/build/output/unity/
[09:51:46] ArgumentException: Texture2D.SetPixels: size of data to be filled was larger than the size of data available in the source array. (Texture '')
UnityEngine.Texture2D.SetPixels (System.Int32 x, System.Int32 y, System.Int32 blockWidth, System.Int32 blockHeight, UnityEngine.Color[] colors, System.Int32 miplevel) (at /Users/bokken/build/output/unity/
[09:51:46] ArgumentException: Texture2D.SetPixels: size of data to be filled was larger than the size of data available in the source array. (Texture '')
UnityEngine.Texture2D.SetPixels (System.Int32 x, System.Int32 y, System.Int32 blockWidth, System.Int32 blockHeight, UnityEngine.Color[] colors, System.Int32 miplevel) (at /Users/bokken/build/output/unity/
[09:51:46] ArgumentException: Texture2D.SetPixels: size of data to be filled was larger than the size of data available in the source array. (Texture '')
UnityEngine.Texture2D.SetPixels (System.Int32 x, System.Int32 y, System.Int32 blockWidth, System.Int32 blockHeight, UnityEngine.Color[] colors, System.Int32 miplevel) (at /Users/bokken/build/output/unity/
[09:51:46] ArgumentException: Texture2D.SetPixels: size of data to be filled was larger than the size of data available in the source array. (Texture '')
UnityEngine.Texture2D.SetPixels (System.Int32 x, System.Int32 y, System.Int32 blockWidth, System.Int32 blockHeight, UnityEngine.Color[] colors, System.Int32 miplevel) (at /Users/bokken/build/output/unity/
[09:51:46] ArgumentException: Texture2D.SetPixels: size of data to be filled was larger than the size of data available in the source array. (Texture '')
UnityEngine.Texture2D.SetPixels (System.Int32 x, System.Int32 y, System.Int32 blockWidth, System.Int32 blockHeight, UnityEngine.Color[] colors, System.Int32 miplevel) (at /Users/bokken/build/output/unity/
```

Upon researching this error, it was clear to me that the program was trying to execute functions on a texture that didn't exist. This is due to the delay in the camera initialization and pre-processing. In order to overcome this issue, I need to introduce functionality that will check if the camera texture has updated since the last frame and if the textures are empty that will prevent further program executions.

EmotionDetectionScript.cs



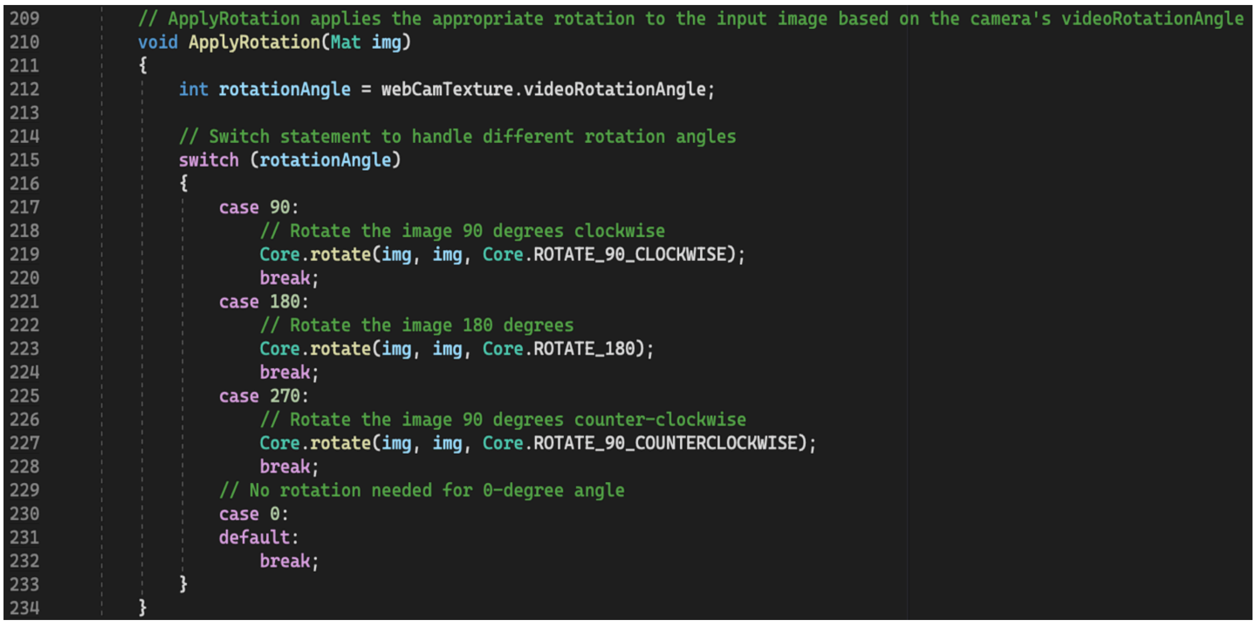
```
    // Update is called once per frame
    void Update()
    {
        faceCount = 0;

        if (webCamTexture.isPlaying && webCamTexture.didUpdateThisFrame) //Checks if there is a texture from the camera and if it has updated this frame
        {
            //If there is anything missing or not ready for processing, it will return and keep checking until there is a valid texture, and webcam is on
            if (texture == null || webCamTexture == null || !webCamTexture.isPlaying || !webCamTexture.didUpdateThisFrame)
            {
                return;
            }

            // Convert the WebCamTexture to a Texture2D
            texture.SetPixels(webCamTexture.GetPixels());
            texture.Apply();

            // Convert the Texture2D to a Mat object that OpenCV can work with
            Mat rgbaMat = new Mat(texture.height, texture.width, CvType.CV_8UC4); // Use the target dimensions
            Utils.texture2DToMat(texture, rgbaMat);

            //Apply a rotation to the image, based on the auto-rotation of the device
            ApplyRotation(rgbaMat);

            // Convert from RGBA to BGR format to be used with the .onnx model
            Imgproc.cvtColor(rgbaMat, bgrMat, Imgproc.COLOR_RGBA2BGR);

```

Now, the update() function proceeds to undergo some checks to ensure that there is a valid texture that has been applied from earlier in the *InitializateWhenReady()* Coroutine and also checks if the webcam texture has a valid texture applied to it from the camera. If not, then the function will keep looping until there is a valid texture and the webcam is on – this is important as if the textures were not ready, there would not be anything that we can input to the AI model to detect, which will cause a large amount of nullExceptionReference errors from unity. As a result, we need to wait for a valid input.

After the checks, are complete, the texture conversions and executions can occur and result in a smooth output with no errors.

ApplyRotation():

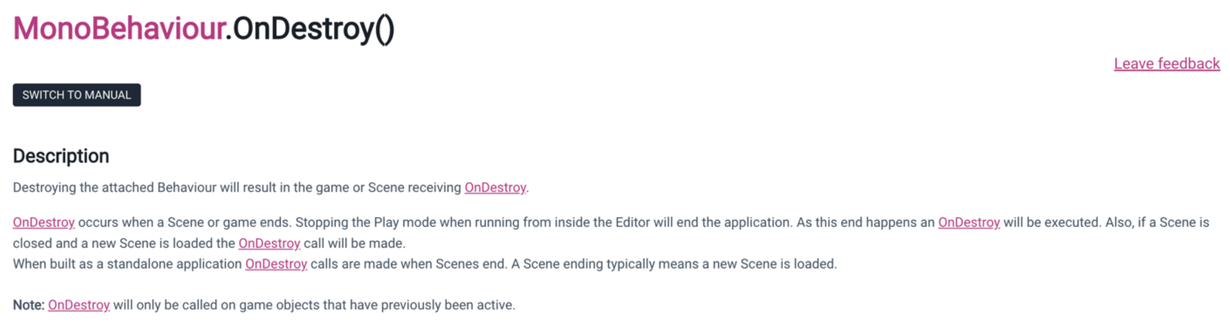
EmotionDetectionScript.cs



```
    // ApplyRotation applies the appropriate rotation to the input image based on the camera's videoRotationAngle
    void ApplyRotation(Mat img)
    {
        int rotationAngle = webCamTexture.videoRotationAngle;

        // Switch statement to handle different rotation angles
        switch (rotationAngle)
        {
            case 90:
                // Rotate the image 90 degrees clockwise
                Core.rotate(img, img, Core.ROTATE_90_CLOCKWISE);
                break;
            case 180:
                // Rotate the image 180 degrees
                Core.rotate(img, img, Core.ROTATE_180);
                break;
            case 270:
                // Rotate the image 90 degrees counter-clockwise
                Core.rotate(img, img, Core.ROTATE_90_COUNTERCLOCKWISE);
                break;
            // No rotation needed for 0-degree angle
            case 0:
            default:
                break;
        }
    }
```

The ApplyRotation() function is called every frame through the Update() function – this is designed to correct the rotation of the image on the screen, dependent on the screen's orientation – this will help correct issues when using the app landscape and portrait. A switch case is used to achieve this.

OnDestroy():


MonoBehaviour.OnDestroy()

Leave feedback

SWITCH TO MANUAL

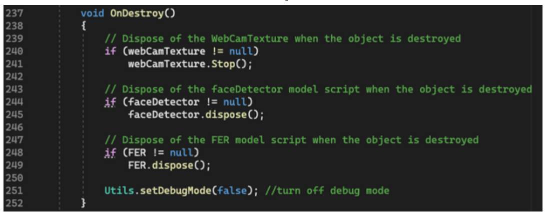
Description

Destroying the attached Behaviour will result in the game or Scene receiving OnDestroy.

OnDestroy occurs when a Scene or game ends. Stopping the Play mode when running from inside the Editor will end the application. As this end happens an OnDestroy will be executed. Also, if a Scene is closed and a new Scene is loaded the OnDestroy call will be made.
When built as a standalone application OnDestroy calls are made when Scenes end. A Scene ending typically means a new Scene is loaded.

Note: OnDestroy will only be called on game objects that have previously been active.

EmotionDetectionScript.cs



```
void OnDestroy()
{
    // Dispose of the WebCamTexture when the object is destroyed
    if (webCamTexture != null)
        webCamTexture.Stop();

    // Dispose of the faceDetector model script when the object is destroyed
    if (faceDetector != null)
        faceDetector.dispose();

    // Dispose of the FER model script when the object is destroyed
    if (FER != null)
        FER.dispose();

    Utils.setDebugMode(false); //turn off debug mode
}
```

The OnDestroy() behaviour will execute when a scene or game ends – this serves several purposes:

- Resource clean-up
- Release camera and device resources
- Debug mode clean-up

These are all important as this helps prevent resource leaks and ensures a proper clean-up when the script or game object is no longer needed or is being removed from the scene.

Test Plan for this version

Due to the nature of my project being dependent on many different functions and inputs, I will need to test that each of the desired outputs are displayed when the expected input is used. To track the test data, I will be using the table below, and providing each test data with a level of priority, (1 being high priority, 2 being mediocre priority, and 3 being low priority)

Test num.	What is being tested	Priority	Input	Expected output	Justification for this data
1	Camera appears on screen and updates in real time	1	Device webcam	Live feed of camera on screen	This is the most crucial part of the project – if the user cannot see what the camera is seeing then there will be no functionality to the rest of the project.
2	Bounding box around detected faces	1	Device webcam	Coloured box being drawn around the face, mapped by calculating coordinates of face corners and adding padding to accommodate	In order for the face detection functionality of my project to work, I will need to make sure that there are bounding boxes that display whether or not there are faces in frame. This will allow the processing of emotion detection.
3	Confidence rating on face detection bounding box	2	Device webcam + detected face Mat	Float between 0 to 1 that is the confidence rating from the AI on its certainty that the bounding boxes are bounding around a face where 1 is 100% certain and 0 is 0% certain	This is not critical to the program, although it is a great feature to have as with the confidence rating, we are able to omit certain detections that fall below the threshold, meaning that we will get less false positive errors throughout the app. Additionally, being able to visually see how accurate it is will provide a greater insight into the performance of the AI model
4	Second bounding box, with detected emotions, colour coded	1	Device webcam + list of detected faces	Inner bounding box, that displays the detected emotion as a text that follows the person's face, also colour co-ordinated so that each different emotion is correspondent to a different coloured bounding box	This is needed as the core functionality of my project, before all else is an emotion detection app AI, and thus the ability to detect emotions is crucial – This would not be possible without the face detection.

5	Confidence rating on the emotions detected by the AI	2	Detected emotion + bounding box	Float between 0 to 1 that is the confidence rating from the AI on its certainty of the prediction of the person’s emotion,	See Test num. 3
6	Ability to detect multiple faces	2	Device webcam + FaceDetection AI	Multiple different bounding boxes for each detected face, along with their confidence rating and detected emotion for each	This is fairly important as there will often be more than one person in frame when pointing the camera at someone, especially in a public environment, so it is important to be able to process multiple different objects simultaneously.
7	Resizing quad dependent on screen size	3	Device screen width + height, Webcam resolution	Quad should scale adequality with screen size and maintain aspect ratio	This is not heavily important to the functioning of the program, but to ease useability and prevent cropping and issues, this feature should be executed well
8	Ability to switch between cameras	1	List of device cameras, switch camera button	When button is pressed, program will cycle through the available cameras, skipping those that are unnecessary, such as ultrawide	This is very important as the end-user will most likely want to be able to use the program for the front and rear camera of their phone and the ability to seamlessly switch between them will increase the quality and useability of the program
9	Switch Camera button disappears when only one camera on system	2	List of device cameras	If there is only one camera, such as on a laptop, the switch camera button should be disabled	This is not critical but it is important as the switch camera button will be useless to a user who is using a device with only one camera – this is because it will call a function to cycle through cameras, yet there are none to cycle through, so it will only cause the screen to lag slightly.
10	Auto-Rotate scale camera	2	Device orientation	Camera matches device orientation seamlessly	This is important as many users like to use the camera in portrait, while others in landscape, so in order to maximise accessibility, there needs to be functionality to switch between both.

11	iPhone Deployment	1	n/a	The application runs and works when built on an iphone using xCode for deployment	My application is aimed to be accessible via a mobile phone. In order to test this, I must be able to build and use the application externally on my phone.

Test Results / Evidence

Test num.	What is being tested	Result	Input	Expected output	Comments	Evidence
1	Camera appears on screen and updates in real time	🟩	Device webcam	Live feed of camera on screen	n/a	Figure 1.1
2	Bounding box around detected faces	🟩	Device webcam	Coloured box being drawn around the face, mapped by calculating coordinates of face corners and adding padding to accommodate	n/a	Figure 1.1
3	Confidence rating on face detection bounding box	🟩	Device webcam + detected face Mat	Float between 0 to 1 that is the confidence rating from the AI on its certainty that the bounding boxes are bounding around a face where 1 is 100% certain and 0 is 0% certain	n/a	Figure 1.1
4	Second bounding box, with detected emotions, colour coded	🟩	Device webcam + list of detected faces	Inner bounding box, that displays the detected emotion as a text that follows the person's face, also colour co-ordinated so that each different emotion is correspondent to a different coloured bounding box	n/a	Figure 1.1
5	Confidence rating on the emotions detected by the AI	🟩	Detected emotion + bounding box	Float between 0 to 1 that is the confidence rating from the AI on its certainty of the prediction of the person's emotion.	n/a	Figure 1.1

		🟩				
6	Ability to detect multiple faces	🟩	Device webcam + FaceDetection AI	Multiple different bounding boxes for each detected face, along with their confidence rating and detected emotion for each	n/a	Figure 1.2
7	Resizing quad dependent on screen size	🟩	Device screen width + height, Webcam resolution	Quad should scale adequality with screen size and maintain aspect ratio	Letterboxes appear on left and right side all the time, although the height is fine	All Figures
8	Ability to switch between cameras	🟩	List of device cameras, switch camera button	When button is pressed, program will cycle through the available cameras, skipping those that are unnecessary, such as ultrawide	n/a	Figure 1.3
9	Switch Camera button disappears when only one camera on system	🟩	List of device cameras	If there is only one camera, such as on a laptop, the switch camera button should be disabled	n/a	Figure 1.4
10	Auto-Rotate scale camera	🟧	Device orientation	Camera matches device orientation seamlessly	When ported to iPhone, it did not work in portrait mode, it was rotated 90 degrees off to the side and did not detect faces when side ways	Figure 1.5
11	iPhone Deployment	🟩	n/a	The application runs and works when built on an iphone using xCode for deployment	n/a	Figure 1.6

All following screenshots have been captured on a working iPhone build (except Figure 1.4)

Figure 1.1

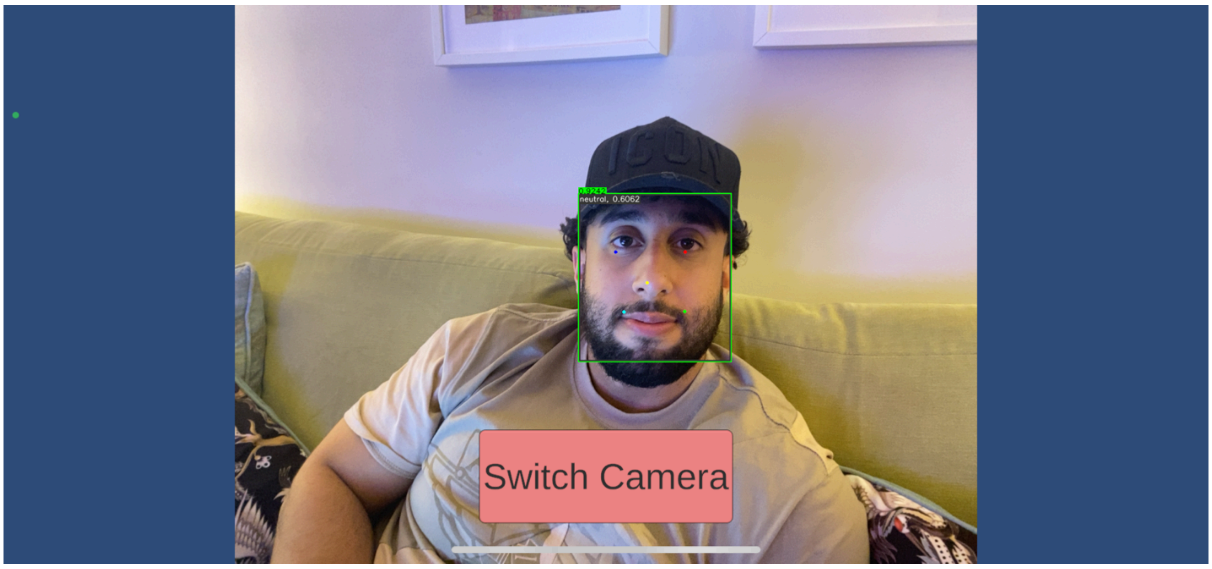


Figure 1.2

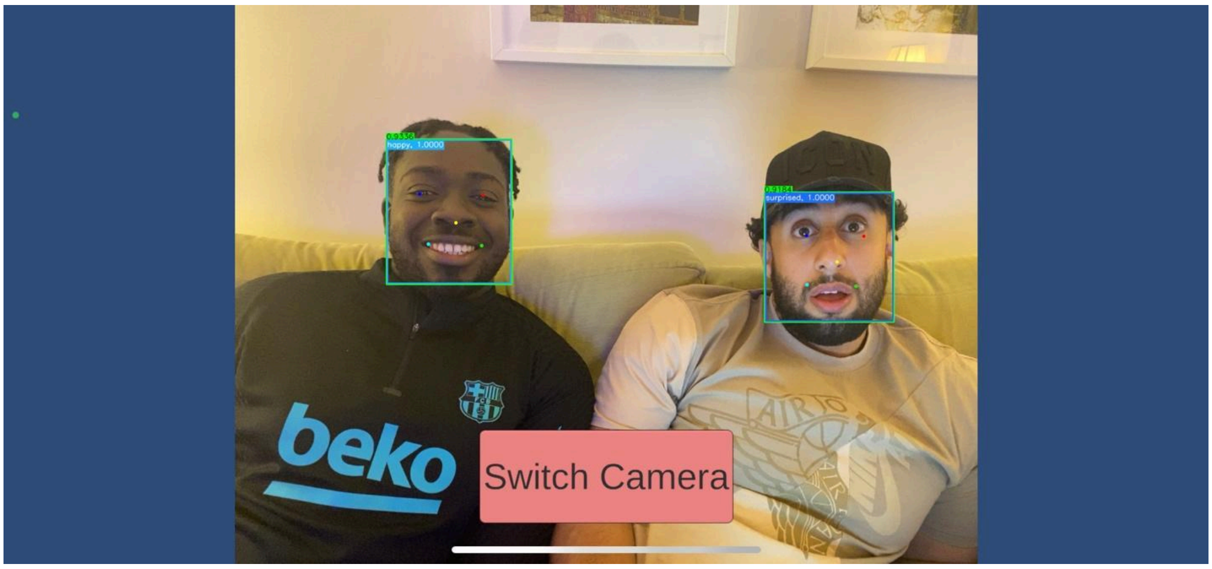


Figure 1.3

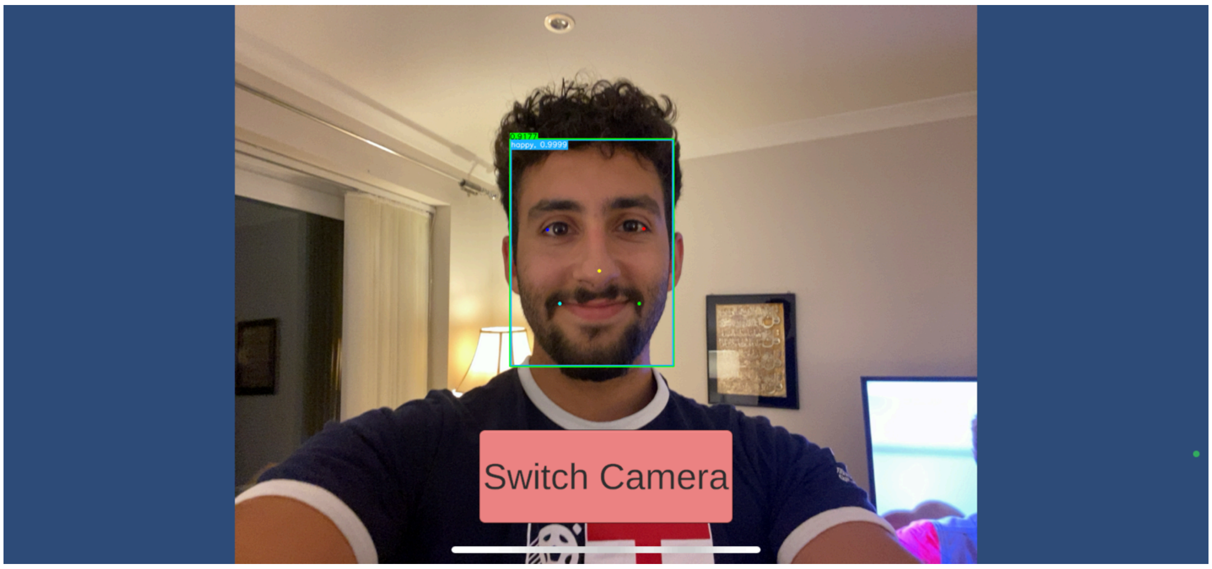


Figure 1.4 (on laptop with one camera the "Switch Camera" button is disabled)

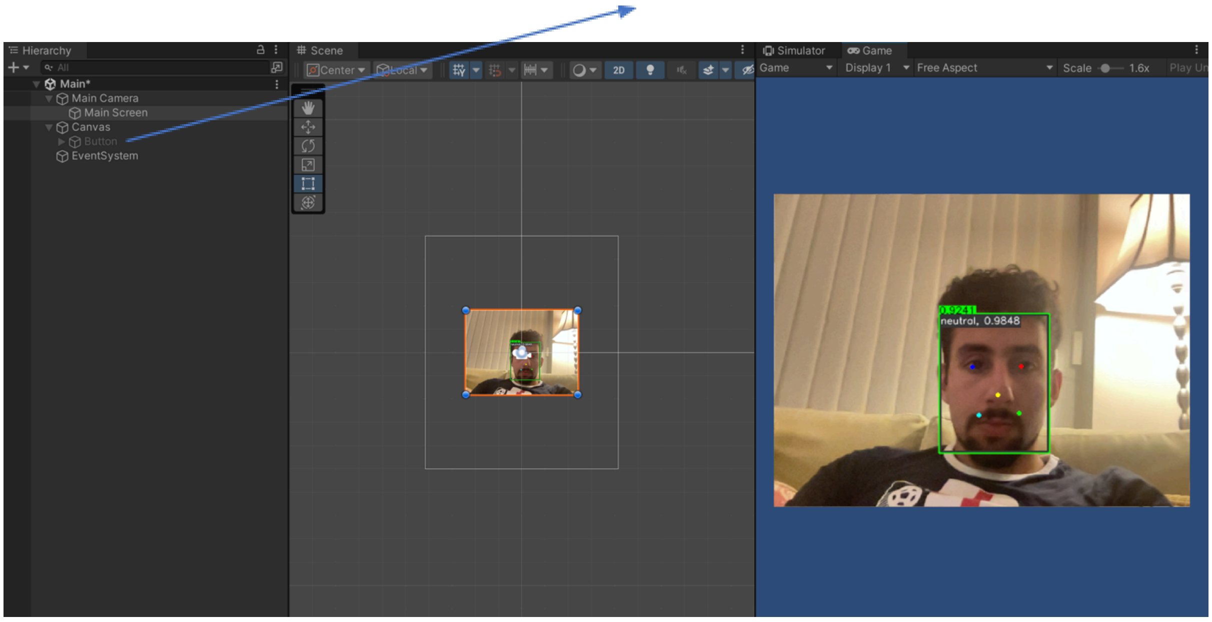


Figure 1.5 (When using portrait, the screen doesn't scale and rotate properly)

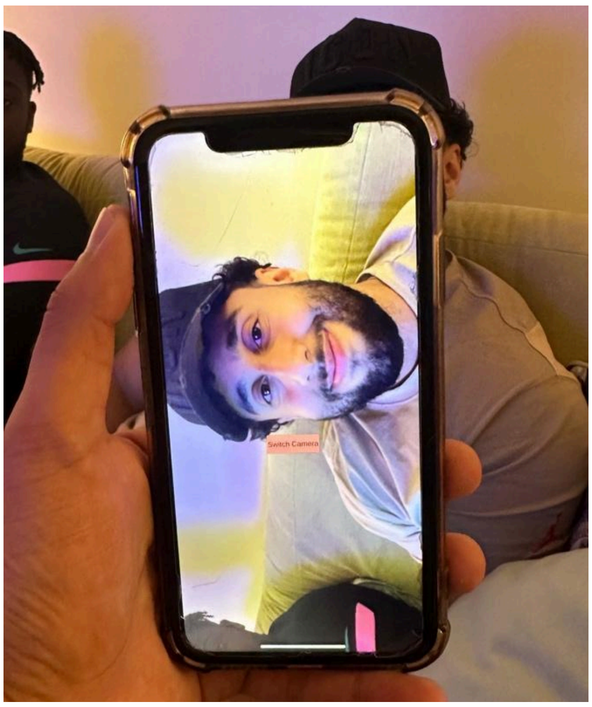
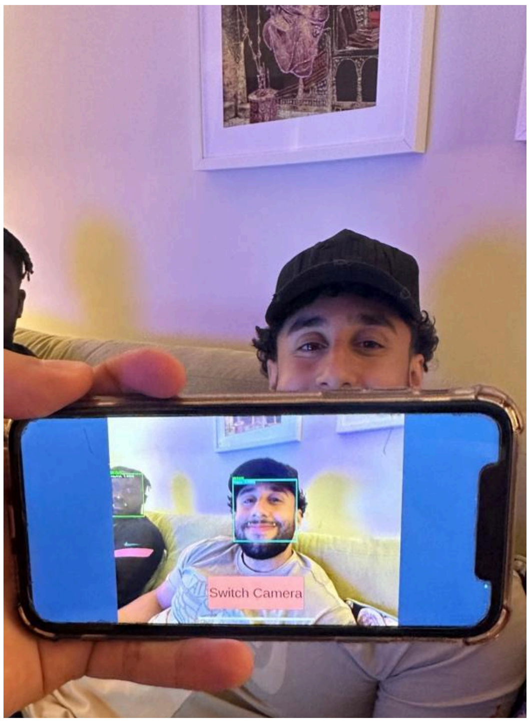


Figure 1.6 (perfectly working on iPhone)



Feedback from Stakeholder

My stakeholders, as explained earlier, are a mixture of potential users and tech enthusiasts who would be using this project to help benefit their personal life or learn more about the common use of artificial intelligence. Since many of the final features have not been implemented yet, I have chosen to interview Aiad Tarik and Mario Prifti for their opinions on the first iteration of prototype 1 as I didn't find any point in interviewing Mike Parish as many of the features that will assist in his useability will be implemented later on in the further iterations. The 2 stakeholders will be referred to as AT and MP to indicate the answers that they will provide.

Is the app easy to use?

MP: "I think the app is very easy to use at the moment – as soon as you start the app, you are launched into the main screen and simply point the camera. The 'Switch Camera' button is clear and well defined"

AT: "I think the app has great potential, although I believe there should be a landing screen before the initial start-up, where the user is able to change preferences or learn about how to use the application. Furthermore, I think that the emotion labels are very small to read and could be particularly difficult to those who are more visually impaired than most."

How do you feel about the accuracy of the detections?

MP: "It's very accurate at detecting faces and emotions – I didn't have any problems throughout. I pointed it at a person far away and it recognised their presence"

AT: "The accuracy of the face detection is brilliant, although I believe that that confidence threshold should be increased slightly – this is because it is much more difficult to detect emotions than faces. And if there is a face that can be detected from far away, the emotion will most likely be inaccurate and for this reason, I think it would be more beneficial to compromise some low-quality faces, for higher quality emotion recognition"

How did you find the seamlessness of switching cameras?

MP: "Switching cameras was not an issue, simply a matter of pressing the button – perfect!"

AT: "No flaws whatsoever, little lag and quite smooth – I didn't find any errors or problems switching the cameras"

Are there any improvements you would recommend to the currently implemented features?

MP: “I would prefer it if the camera spans across the whole screen and perhaps allowing it to work in portrait mode”

AT: “I think it would benefit from increasing the frame rate, as it is slightly low at the moment – by reducing the resolution of the camera input and adjusting the code to update when the device auto-rotates, allowing portrait mode functionality”

Changes/Fixes that I now plan to make to the design or code as a result of testing and feedback

The main failing point for this iteration was the scale and rotation of the device – Both issues stem from the setup of the unity scene game objects and the code – For my next iteration, I am going to alter the ApplyRotation() function to auto-update when the screen orientation changes, and also altering the maths from within the initializeWhenReady() function in order to span the camera across the whole screen, while maintaining an acceptable aspect ratio.

For my next iteration, I plan to add:

- On-screen face counter
- Variable confidence meter
- Text-to-speech button
- Image Captioning

Evaluation

This iteration aimed to create the foundation of this project, the skeleton, if you will. By doing so, I can now easily add or alter features within the program without the need to change the majority of the program, It all comes down to adding and fixing, until the stakeholders and I are content with the product. Ultimately, I believe I was successful in achieving most of the aims for this iteration, with the exception of 1 or 2 bugs to be fixed. The goals that have been achieved in this iteration include:

1. Device Camera input
2. Import OpenCV library of pre-trained AI models
3. Face tracking
4. Multi-face tracking
5. Confidence rating
6. Draw bounding box
7. Emotion Recognition AI import
8. Change Camera Button
9. Face counter (Console debug log)

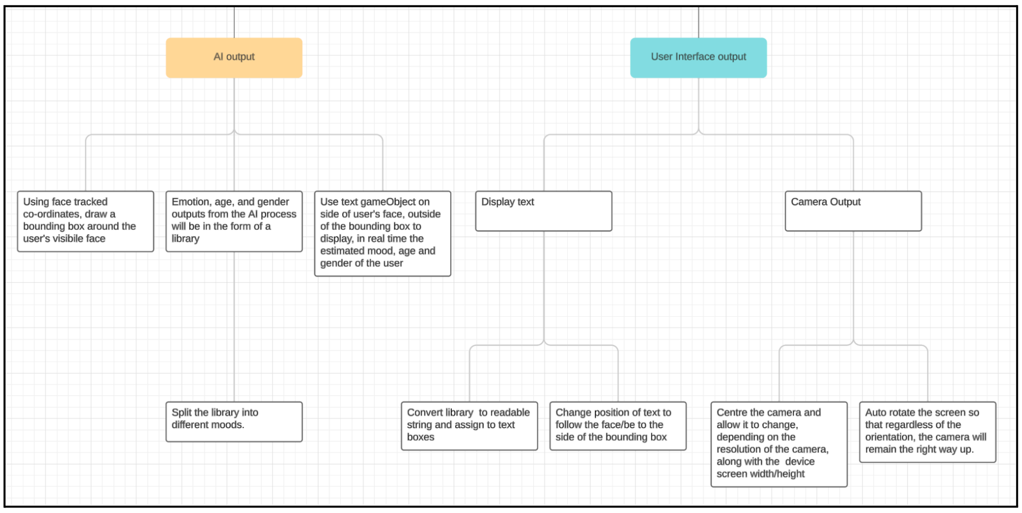
The sections of this iteration I found most difficult would have been the set-up of the AI models – I found this the most difficult as Unity does not have well-documented support for artificial intelligence – the built-in Unity library for this is barracuda, although, with low support and slow executing times, this would not fit the scale and nature of my project, due to its complexness. As a result, I settled for an external library, OpenCVForUnity. This library is a port of OpenCV C++ to work with c# using a library of pre-existing, pre-trained models, such as facial recognition and emotion recognition, which is precisely what I had wanted for this project, although it lacks many other key models that I will need such as an image captioning model and an age/gender model. As a result, in my next iteration, I am going to set up a Python server with these models, that will run and execute at a slower rate than the built-in OpenCV models, providing full functionality to my project.

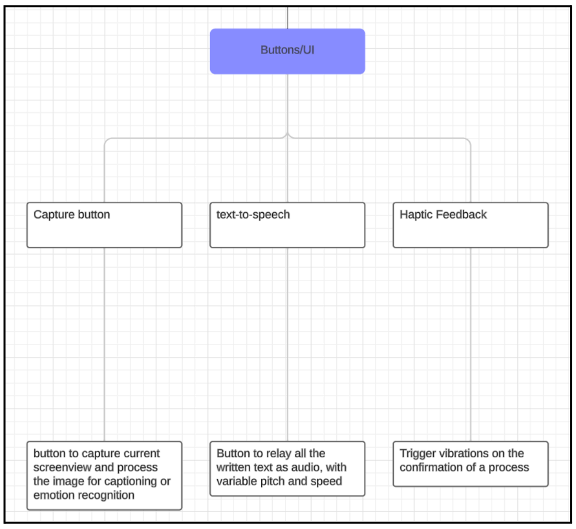
Iteration 2 - *Date 03/10/2023*

Aims for this iteration

The aim for iteration 2 is to further develop the backend of the project, this involves adding more AI features to the project, bug fixing from iteration 1, and beginning to involve my 3rd stakeholder, Mike Parish, as this iteration will tailor more towards the accessibility side of development. I am also going to attempt to fine-tune the pre-existing AI models that have been incorporated into iteration 1. By the end of this iteration, I want another AI functionality for the project, preferably an object detection with a confidence threshold slider, whereby the user is able to alter how confident the AI is before it replies with an output.

Below are some of the functions from the top-down design that I will be working on and improving from iteration 1 during this portion of development:


AI output
User Interface output
Using face tracked co-ordinates, draw a bounding box around the user's visibile face
Emotion, age, and gender outputs from the AI process will be in the form of a library
Use text gameObject on side of user's face, outside of the bounding box to display, in real time the estimated mood, age and gender of the user
Display text
Camera Output
Split the library into different moods.
Convert library to readable string and assign to text boxes
Change position of text to follow the face/be to the side of the bounding box
Centre the camera and allow it to change, depending on the resolution of the camera, along with the device screen width/height
Auto rotate the screen so that regardless of the orientation, the camera will remain the right way up.


Buttons/UI
Capture button
text-to-speech
Haptic Feedback
button to capture current screenview and process the image for captioning or emotion recognition
Button to relay all the written text as audio, with variable pitch and speed
Trigger vibrations on the confirmation of a process

Iteration 2 is not limited to the functions shown above as I plan to implement further improvements and functionalities throughout development.

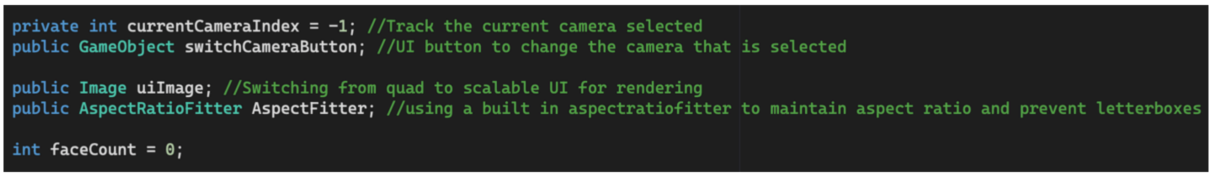
If I have time during this iteration, I will attempt to include AI image captioning, which is highly ambitious due to its complexity. I will plan to do this through a python flask server as Unity will be unable to handle the complex task of processing images and captioning. Additionally, nearly all image captioning models for this task are designed for Tensorflow or Pytorch, which is very difficult to implement into Unity, and for the sake of simplicity, I will be doing the processing externally, and returning the results, in a string format to unity, where it will be able to display the caption as a formatted text gameObject.

Finally, I aim to add a settings\pause button where the user can alter some of the program's functionality, such as the confidence threshold, text-to-speech volume etc.

Bug Fixing:

Before I begin adding features and continuing with development, I am going to need to fix all/most of the bugs that were discovered from the previous iteration. This includes full scalability of the camera on the screen. And if I have time, portrait functionality (This is not a critical bug as I can make the app landscape mode only, although, If I have enough time, I will attempt to introduce portrait mode logic)

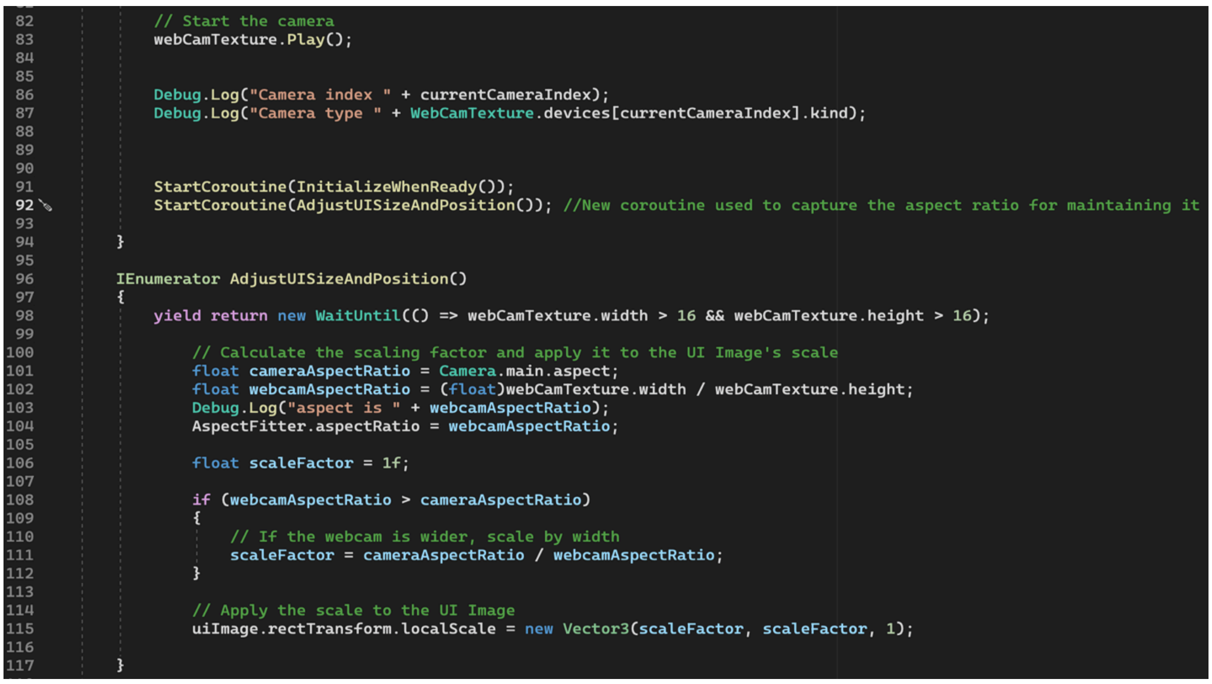
EmotionDetectionScript.cs



```
private int currentCameraIndex = -1; //Track the current camera selected
public GameObject switchCameraButton; //UI button to change the camera that is selected

public Image uiImage; //Switching from quad to scalable UI for rendering
public AspectRatioFitter AspectFitter; //using a built in aspectratiofitter to maintain aspect ratio and prevent letterboxes

int faceCount = 0;
```

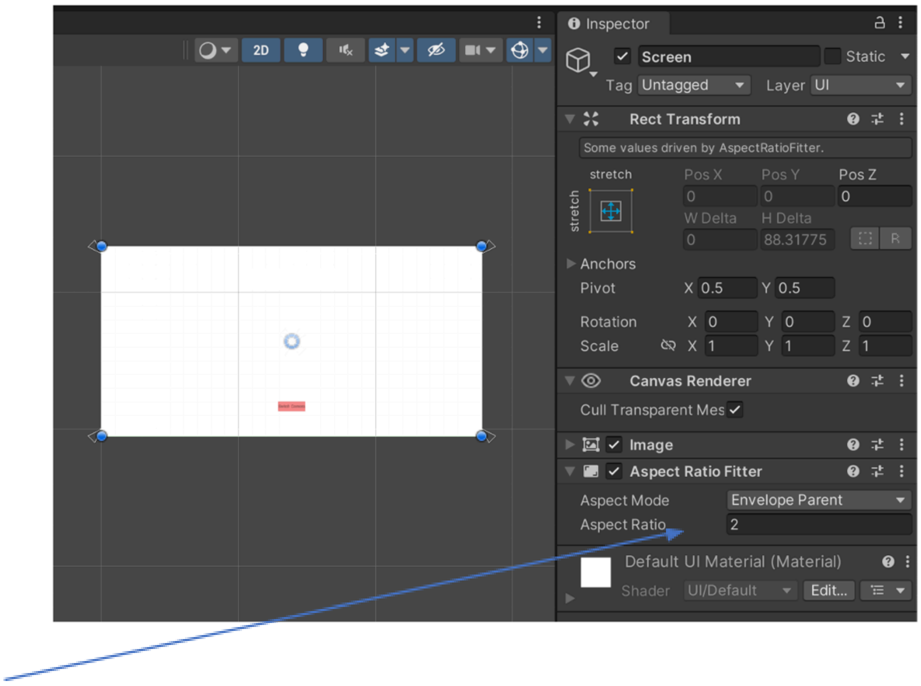
The first thing I did in tackling the scaling problem was switch the rendering mode from putting a texture onto the quad to creating a scalable UI gameObject that I can map the texture onto – this means I do not have to worry about unnecessary maths and rescaling issues since Unity will handle this for me.

EmotionDetectionScript.cs



```
        // Start the camera
        webCamTexture.Play();


        Debug.Log("Camera index " + currentCameraIndex);
        Debug.Log("Camera type " + WebCamTexture.devices[currentCameraIndex].kind);



        StartCoroutine(InitializeWhenReady());
        StartCoroutine(AdjustUISizeAndPosition()); //New coroutine used to capture the aspect ratio for maintaining it

    }

    IEnumerator AdjustUISizeAndPosition()
    {
        yield return new WaitUntil(() => webCamTexture.width > 16 && webCamTexture.height > 16);

            // Calculate the scaling factor and apply it to the UI Image's scale
            float cameraAspectRatio = Camera.main.aspect;
            float webcamAspectRatio = (float)webCamTexture.width / webCamTexture.height;
            Debug.Log("aspect is " + webcamAspectRatio);
            AspectFitter.aspectRatio = webcamAspectRatio;

            float scaleFactor = 1f;

            if (webcamAspectRatio > cameraAspectRatio)
            {
                // If the webcam is wider, scale by width
                scaleFactor = cameraAspectRatio / webcamAspectRatio;
            }

            // Apply the scale to the UI Image
            uiImage.rectTransform.localScale = new Vector3(scaleFactor, scaleFactor, 1);

    }
```

I then created a coroutine, similar to *InitializeWhenReady()* but instead focused on capturing the aspect ratio of the webcam and the screen size, adjusting as necessary and then assigning the values to the aspect ratio fitter.



Inspector

Screen   Static

Tag Untagged   Layer UI

Rect Transform

Some values driven by AspectRatioFitter.

stretch   Pos X 0   Pos Y 0   Pos Z 0

W Delta 0   H Delta 88.31775

Anchors

Pivot X 0.5 Y 0.5

Rotation X 0 Y 0 Z 0

Scale X 1 Y 1 Z 1

Canvas Renderer

Cull Transparent Mes ✓

Image

Aspect Ratio Fitter

Aspect Mode Envelope Parent

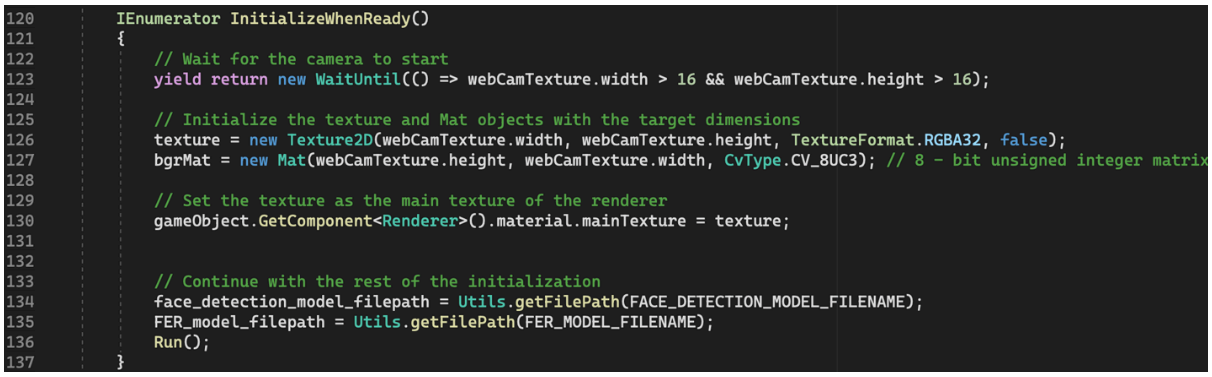
Aspect Ratio 2

Default UI Material (Material)

Shader UI/Default   Edit...

This aspect ratio variable will scale with the screen size, in order to prevent the letterboxes on the sides

EmotionDetectionScript.cs



```
    IEnumerator InitializeWhenReady()
    {
        // Wait for the camera to start
        yield return new WaitUntil(() => webCamTexture.width > 16 && webCamTexture.height > 16);

        // Initialize the texture and Mat objects with the target dimensions
        texture = new Texture2D(webCamTexture.width, webCamTexture.height, TextureFormat.RGBA32, false);
        bgrMat = new Mat(webCamTexture.height, webCamTexture.width, CvType.CV_8UC3); // 8 - bit unsigned integer matrix

        // Set the texture as the main texture of the renderer
        gameObject.GetComponent<Renderer>().material.mainTexture = texture;


        // Continue with the rest of the initialization
        face_detection_model_filepath = Utils.getFilePath(FACE_DETECTION_MODEL_FILENAME);
        FER_model_filepath = Utils.getFilePath(FER_MODEL_FILENAME);
        Run();
    }
```

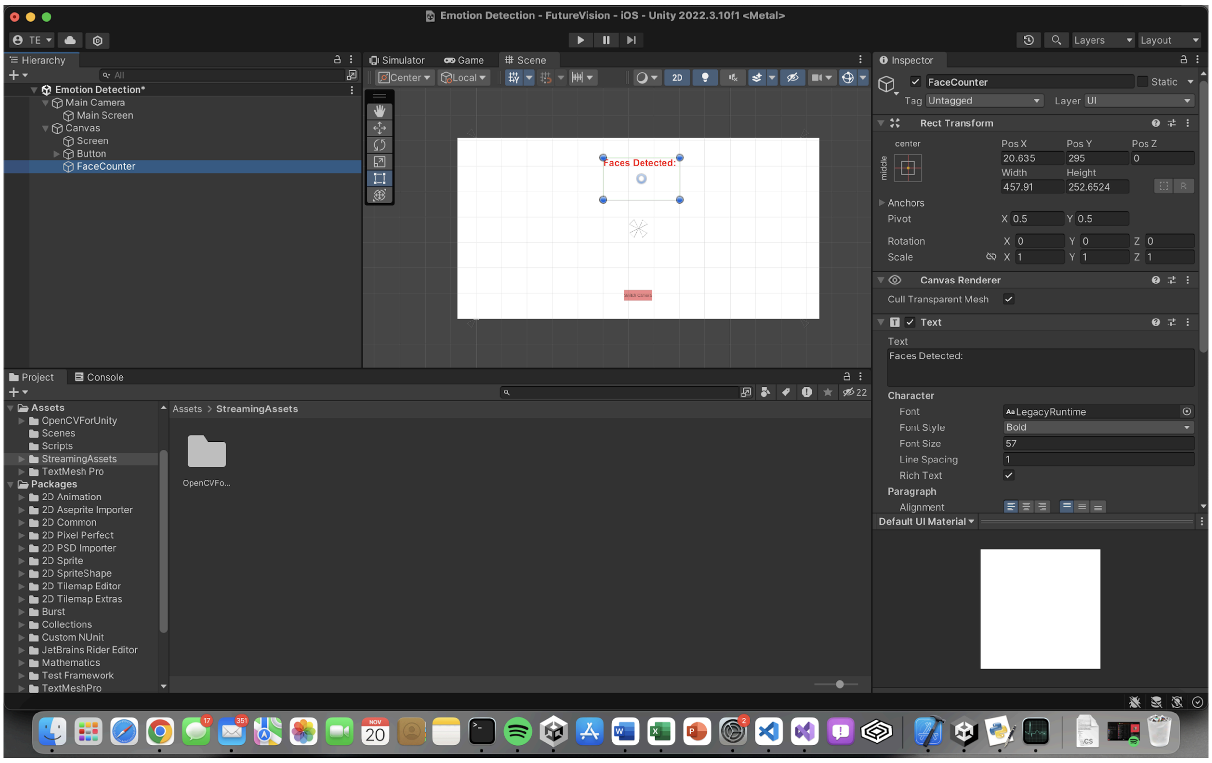
This now meant that I could remove the bulk of the *InitializeWhenReady()* coroutine as most of it was maths related to scaling the quad gameObject, but now it isn't needed as we have *AdjustUISizeAndPosition()* to do this for us.

Summary of aims:

- Set up a home splash screen to choose between scenes
  - Scenes will include Face-Emotion detection and Object detection
  - User scene management and UI buttons for navigation
- Set up a face counter on-screen that measures the number of faces detected and outputs an integer
- Import and set up the ObjectDetection scene using OpenCV's built-in model
  - Adjust Screen positioning to match the aesthetic of the rest of the project
- Set up a slider to alter the confidence threshold
  - AI will only output results if it is more confident than the set threshold
    - This is a scale from 0.00 to 1.00, where 1.00 is when the AI is 100% confident in its prediction
- Set up a Python server to process image captioning from unity.

Annotated code screenshots with description

Face counter (C#)



```
Emotion Detection - FutureVision - iOS - Unity 2022.3.10f1 <Metal>
```

Hierarchy:

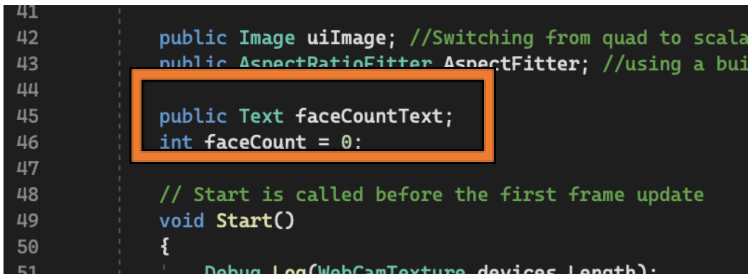
```
Emotion Detection*
  Main Camera
    Main Screen
  Canvas
    Screen
    Button
    FaceCounter
```

Inspector:

```
FaceCounter                          Static
Tag Untagged                         Layer UI

Rect Transform
center      Pos X     Pos Y     Pos Z
            20.635    295       0
            Width     Height
            457.91    252.6524
Anchors
Pivot       X 0.5     Y 0.5
Rotation    X 0       Y 0       Z 0
Scale       X 1       Y 1       Z 1

Canvas Renderer
Cull Transparent Mesh   ✓

Text
Text
Faces Detected:
Character
  Font          LegacyRuntime
  Font Style    Bold
  Font Size     57
  Line Spacing  1
  Rich Text     ✓
Paragraph
  Alignment
Default UI Material
```

To set up a face counter, I created a standard unity text object with the default text value “Faces detected: ” I will update my ‘*EmotionDetectionScript.cs’* to accommodate this.

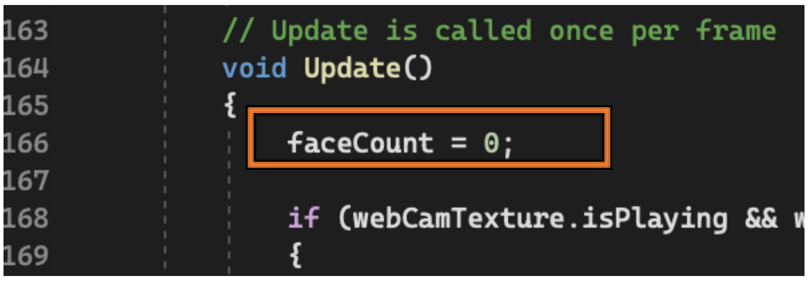
EmotionDetectionScript.cs



```
    public Image uiImage; //Switching from quad to scala
    public AspectRatioFitter AspectFitter; //using a bui

    public Text faceCountText;
    int faceCount = 0;

    // Start is called before the first frame update
    void Start()
    {
        Debug.Log(WebCamTexture.devices.Length);
```

I have initialised 1 new variable; the public faceCountText object which will be updated, instead of debugging the number of faces in the console, this will appear on screen

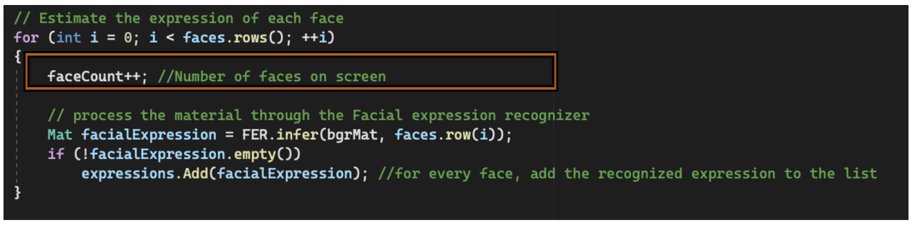
EmotionDetectionScript.cs



```
        // Update is called once per frame
        void Update()
        {
            faceCount = 0;

            if (webCamTexture.isPlaying && w
            {
```

At the start of each Update() call, the faceCount variable resets to 0.

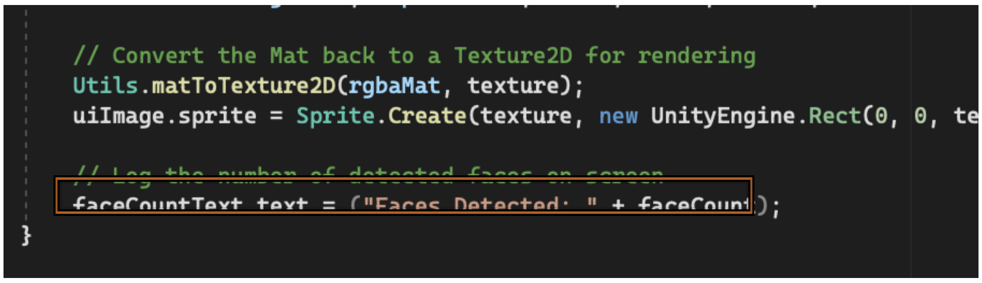
EmotionDetectionScript.cs



```
// Estimate the expression of each face
for (int i = 0; i < faces.rows(); ++i)
{
    faceCount++; //Number of faces on screen

    // process the material through the Facial expression recognizer
    Mat facialExpression = FER.infer(bgrMat, faces.row(i));
    if (!facialExpression.empty())
        expressions.Add(facialExpression); //for every face, add the recognized expression to the list
}
```

For each face detected and inferred using the model, the faceCount variable is incremented.

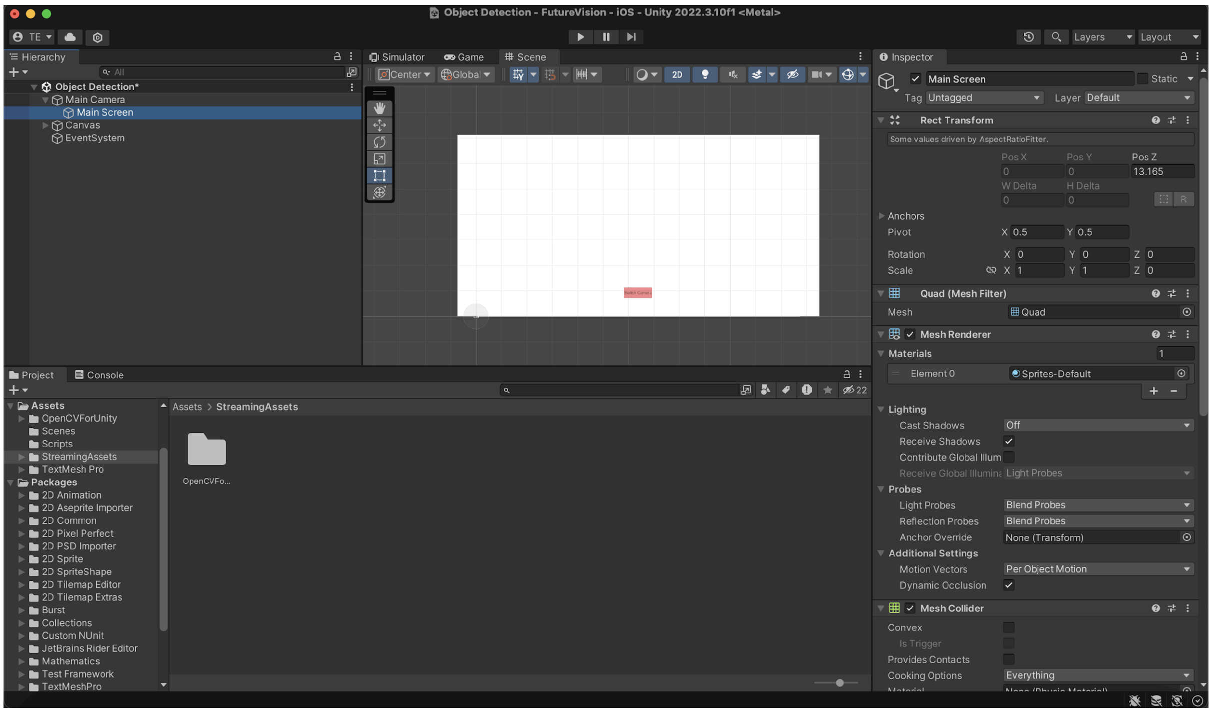
EmotionDetectionScript.cs



```
    // Convert the Mat back to a Texture2D for rendering
    Utils.matToTexture2D(rgbaMat, texture);
    uiImage.sprite = Sprite.Create(texture, new UnityEngine.Rect(0, 0, te

    // Log the number of detected faces on screen
    faceCountText.text = ("Faces Detected: " + faceCount);
}
```

At the end of the Update() loop, the faceCountText variable is updated to display “Faces detected: {faceCount}”, instead of debugging the face count in the console.

YoLo object detection page (C#)



Main Screen ✓ Static
Tag Untagged Layer Default

Rect Transform
Some values driven by AspectRatioFitter.
Pos X 0 Pos Y 0 Pos Z 13.165
W Delta 0 H Delta 0
Anchors
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 0

Quad (Mesh Filter)
Mesh Quad

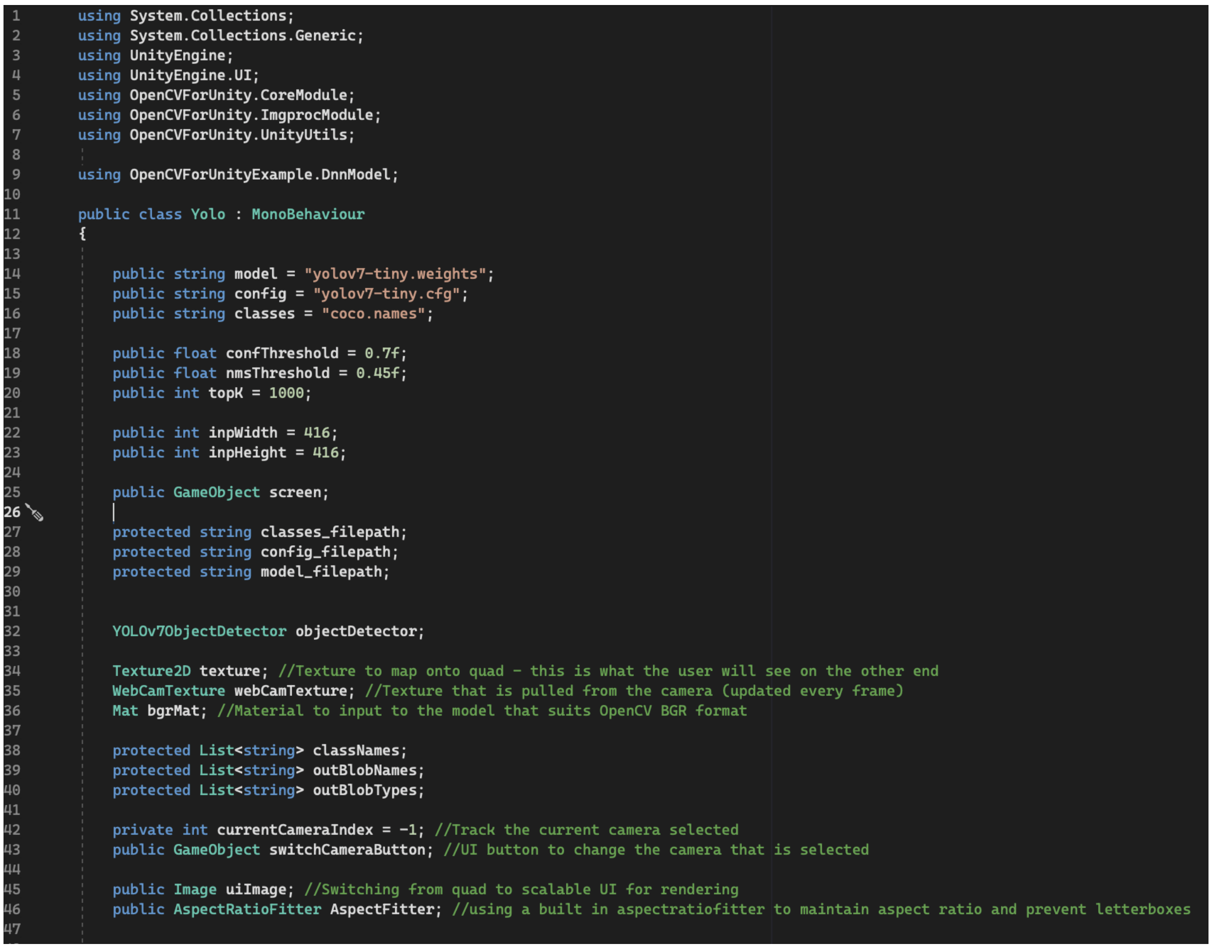
Mesh Renderer
Materials 1
Element 0 Sprites-Default
Lighting
Cast Shadows Off
Receive Shadows ✓
Contribute Global Illum
Receive Global Illumina Light Probes
Probes
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)
Additional Settings
Motion Vectors Per Object Motion
Dynamic Occlusion ✓

Mesh Collider
Convex
Is Trigger
Provides Contacts
Cooking Options Everything

For this part of the project, I want to create a new scene, dedicated to YoLo object detection.

I used the same template for the new scene, called ‘Object Detection’, that I used for ‘Emotion Detection’. This included the ‘Main Screen’ gameObject, however instead of using the *EmotionDetectionScript.cs*, I have created a script called *Yolo.cs* to manage object detection.

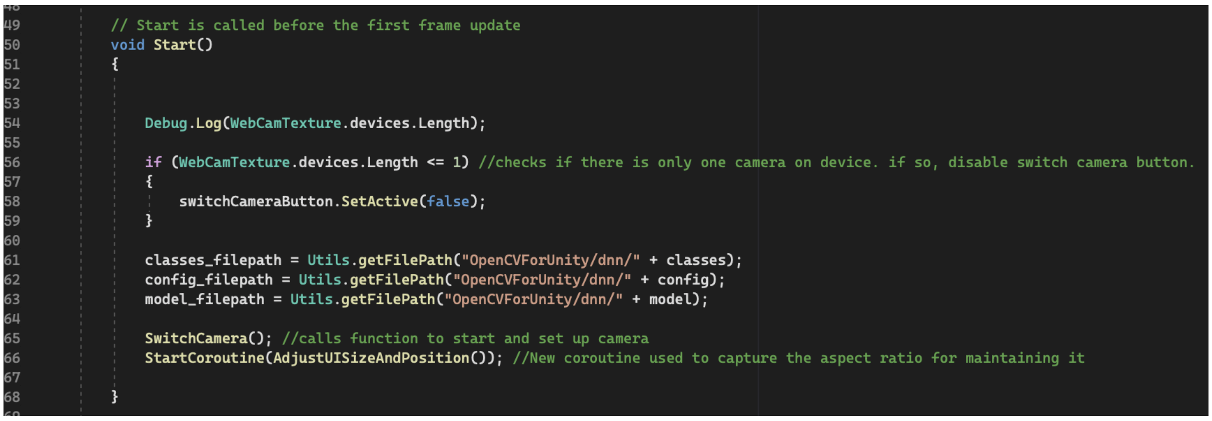
Yolo.cs



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using OpenCVForUnity.CoreModule;
using OpenCVForUnity.ImgprocModule;
using OpenCVForUnity.UnityUtils;

using OpenCVForUnityExample.DnnModel;

public class Yolo : MonoBehaviour
{

    public string model = "yolov7-tiny.weights";
    public string config = "yolov7-tiny.cfg";
    public string classes = "coco.names";

    public float confThreshold = 0.7f;
    public float nmsThreshold = 0.45f;
    public int topK = 1000;

    public int inpWidth = 416;
    public int inpHeight = 416;

    public GameObject screen;

    protected string classes_filepath;
    protected string config_filepath;
    protected string model_filepath;


    YOLOv7ObjectDetector objectDetector;

    Texture2D texture; //Texture to map onto quad - this is what the user will see on the other end
    WebCamTexture webCamTexture; //Texture that is pulled from the camera (updated every frame)
    Mat bgrMat; //Material to input to the model that suits OpenCV BGR format

    protected List<string> classNames;
    protected List<string> outBlobNames;
    protected List<string> outBlobTypes;

    private int currentCameraIndex = -1; //Track the current camera selected
    public GameObject switchCameraButton; //UI button to change the camera that is selected

    public Image uiImage; //Switching from quad to scalable UI for rendering
    public AspectRatioFitter AspectFitter; //using a built in aspectratiofitter to maintain aspect ratio and prevent letterboxes

```

The *yolo.cs* script starts off by initializing a variety of variables, such as the model to be used (yolov7), the config file and the classes associated with each object.
It also manages the set confidence threshold, which I may switch to a slider system where the user can set this themselves, along with other necessary variables.

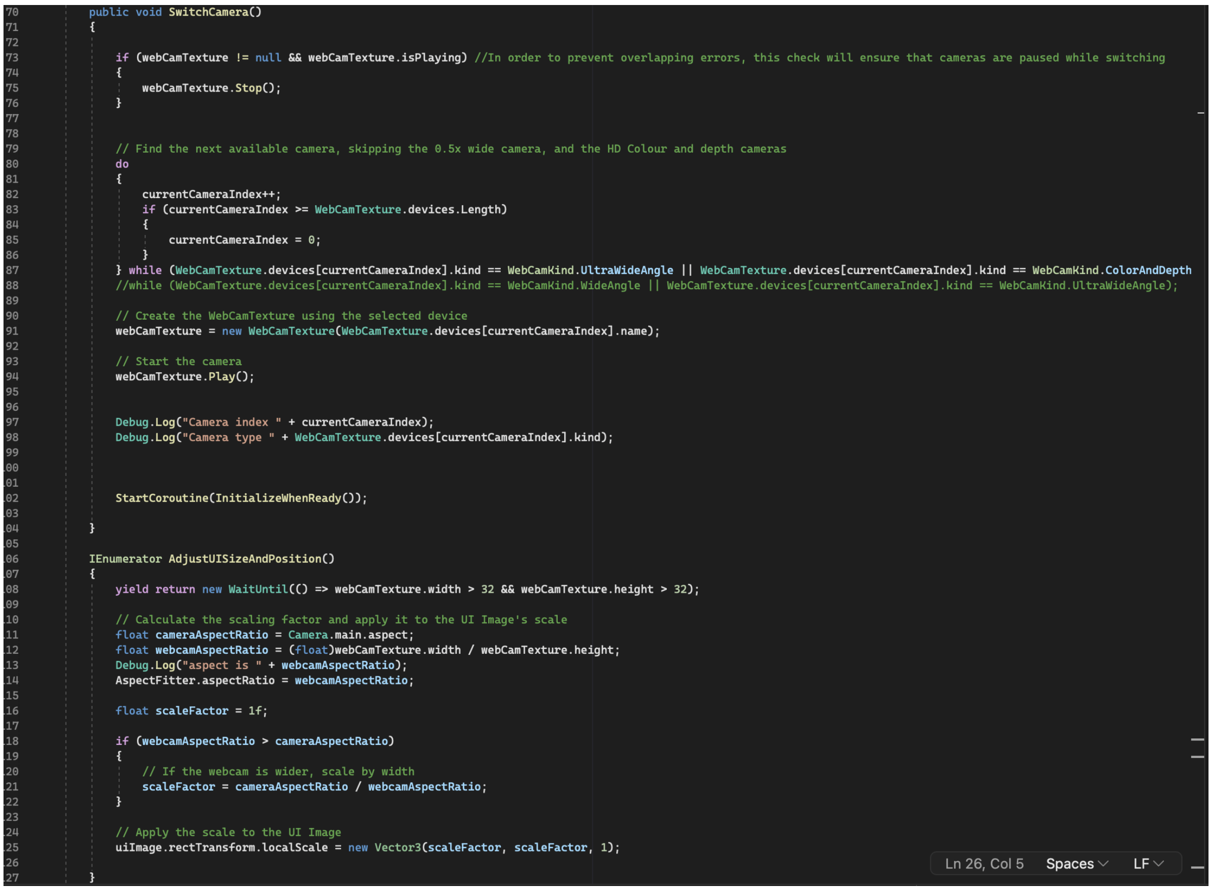
yolo.cs



```
    // Start is called before the first frame update
    void Start()
    {


        Debug.Log(WebCamTexture.devices.Length);

        if (WebCamTexture.devices.Length <= 1) //checks if there is only one camera on device. if so, disable switch camera button.
        {
            switchCameraButton.SetActive(false);
        }

        classes_filepath = Utils.getFilePath("OpenCVForUnity/dnn/" + classes);
        config_filepath = Utils.getFilePath("OpenCVForUnity/dnn/" + config);
        model_filepath = Utils.getFilePath("OpenCVForUnity/dnn/" + model);

        SwitchCamera(); //calls function to start and set up camera
        StartCoroutine(AdjustUISizeAndPosition()); //New coroutine used to capture the aspect ratio for maintaining it

    }
```

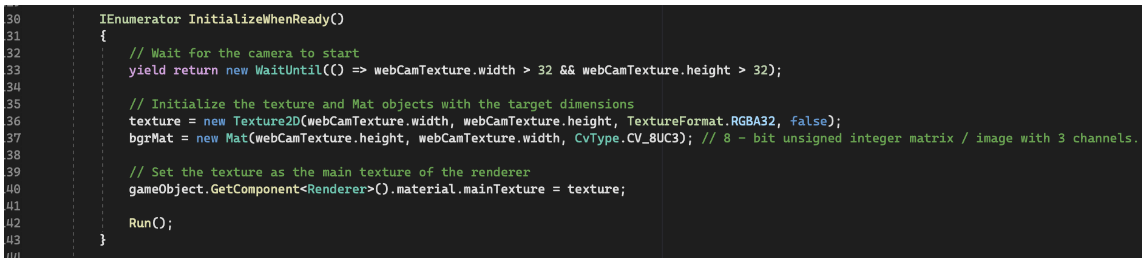
The start() function is almost identical to that of *EmotionDetectionScript.cs* This is because I have aimed to make parts of the scripts as re-useable as possible to be used by different models using the same structure and layout of code. The only difference in *yolo.cs* is that the protected strings ‘classes_filepath’, ‘config_filepath’, and ‘model_filepath’ have to be set to the associated filepaths of their respective files, all of which are stored in “OpenCVForUnity/dnn/..”

yolo.cs



```
    public void SwitchCamera()
    {

        if (webCamTexture != null && webCamTexture.isPlaying) //In order to prevent overlapping errors, this check will ensure that cameras are paused while switching
        {
            webCamTexture.Stop();
        }


        // Find the next available camera, skipping the 0.5x wide camera, and the HD Colour and depth cameras
        do
        {
            currentCameraIndex++;
            if (currentCameraIndex >= WebCamTexture.devices.Length)
            {
                currentCameraIndex = 0;
            }
        } while (WebCamTexture.devices[currentCameraIndex].kind == WebCamKind.UltraWideAngle || WebCamTexture.devices[currentCameraIndex].kind == WebCamKind.ColorAndDepth
        //while (WebCamTexture.devices[currentCameraIndex].kind == WebCamKind.WideAngle || WebCamTexture.devices[currentCameraIndex].kind == WebCamKind.UltraWideAngle);

        // Create the WebCamTexture using the selected device
        webCamTexture = new WebCamTexture(WebCamTexture.devices[currentCameraIndex].name);

        // Start the camera
        webCamTexture.Play();


        Debug.Log("Camera index " + currentCameraIndex);
        Debug.Log("Camera type " + WebCamTexture.devices[currentCameraIndex].kind);



        StartCoroutine(InitializeWhenReady());

    }

    IEnumerator AdjustUISizeAndPosition()
    {
        yield return new WaitUntil(() => webCamTexture.width > 32 && webCamTexture.height > 32);

        // Calculate the scaling factor and apply it to the UI Image's scale
        float cameraAspectRatio = Camera.main.aspect;
        float webcamAspectRatio = (float)webCamTexture.width / webCamTexture.height;
        Debug.Log("aspect is " + webcamAspectRatio);
        AspectFitter.aspectRatio = webcamAspectRatio;

        float scaleFactor = 1f;

        if (webcamAspectRatio > cameraAspectRatio)
        {
            // If the webcam is wider, scale by width
            scaleFactor = cameraAspectRatio / webcamAspectRatio;
        }

        // Apply the scale to the UI Image
        uiImage.rectTransform.localScale = new Vector3(scaleFactor, scaleFactor, 1);

    }
```

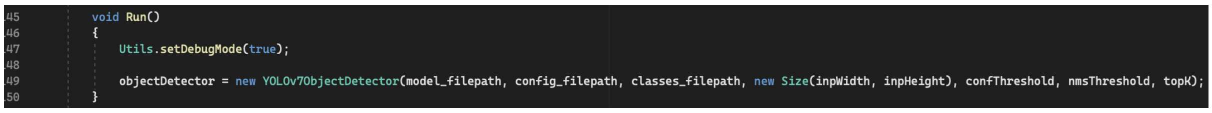
Ln 26, Col 5 Spaces LF



```
    IEnumerator InitializeWhenReady()
    {
        // Wait for the camera to start
        yield return new WaitUntil(() => webCamTexture.width > 32 && webCamTexture.height > 32);

        // Initialize the texture and Mat objects with the target dimensions
        texture = new Texture2D(webCamTexture.width, webCamTexture.height, TextureFormat.RGBA32, false);
        bgrMat = new Mat(webCamTexture.height, webCamTexture.width, CvType.CV_8UC3); // 8 - bit unsigned integer matrix / image with 3 channels.

        // Set the texture as the main texture of the renderer
        gameObject.GetComponent<Renderer>().material.mainTexture = texture;

        Run();
    }
```

The SwitchCamera(), AdjustUISizeAndPosition() and InitializeWhenReady() functions are identical to *EmotionDetectionScript.cs* as the method of camera capturing and rendering is identical and therefore no changes need to be made.

yolo.cs

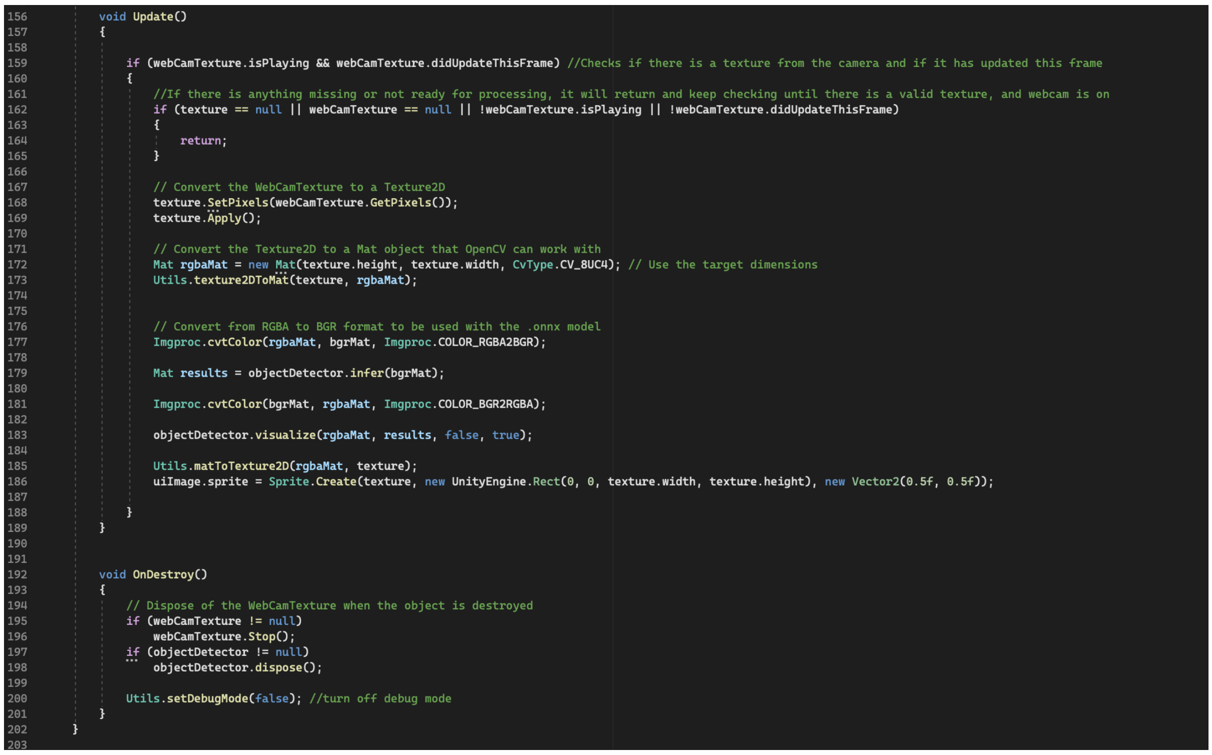


```
    void Run()
    {
        Utils.setDebugMode(true);

        objectDetector = new YOLOv7ObjectDetector(model_filepath, config_filepath, classes_filepath, new Size(inpWidth, inpHeight), confThreshold, nmsThreshold, topK);
    }
```

The Run() function is in charge of initializing the object detector model to be used throughout running and inputs parameters such as the model, config, and classes filepaths, along with the input size, confidence threshold and more.

If I want to actively alter the confidence threshold, I will have to re-initialize the model each time so I will keep this in consideration for the future (this includes the EmotionRecognition model)

yolo.cs



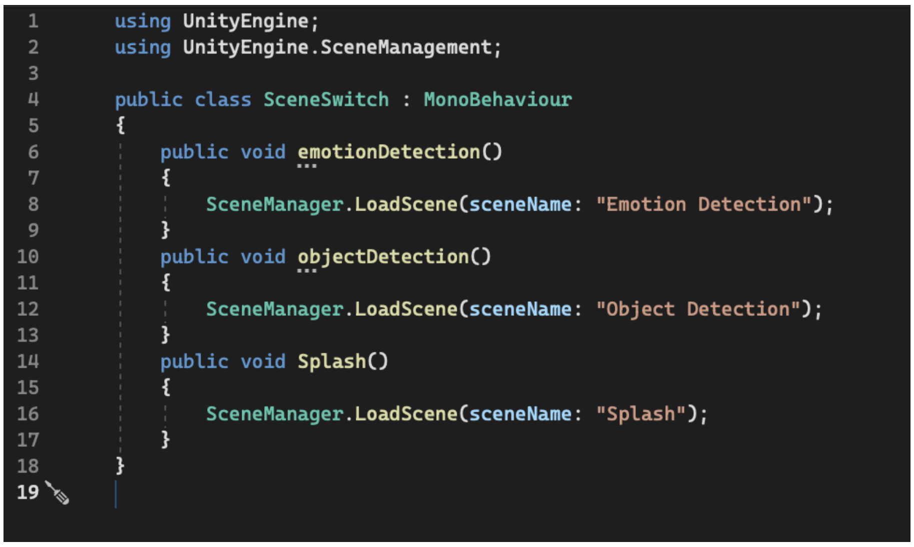
```
    void Update()
    {

        if (webCamTexture.isPlaying && webCamTexture.didUpdateThisFrame) //Checks if there is a texture from the camera and if it has updated this frame
        {
            //If there is anything missing or not ready for processing, it will return and keep checking until there is a valid texture, and webcam is on
            if (texture == null || webCamTexture == null || !webCamTexture.isPlaying || !webCamTexture.didUpdateThisFrame)
            {
                return;
            }

            // Convert the WebCamTexture to a Texture2D
            texture.SetPixels(webCamTexture.GetPixels());
            texture.Apply();

            // Convert the Texture2D to a Mat object that OpenCV can work with
            Mat rgbaMat = new Mat(texture.height, texture.width, CvType.CV_8UC4); // Use the target dimensions
            Utils.texture2DToMat(texture, rgbaMat);


            // Convert from RGBA to BGR format to be used with the .onnx model
            Imgproc.cvtColor(rgbaMat, bgrMat, Imgproc.COLOR_RGBA2BGR);

            Mat results = objectDetector.infer(bgrMat);

            Imgproc.cvtColor(bgrMat, rgbaMat, Imgproc.COLOR_BGR2RGBA);

            objectDetector.visualize(rgbaMat, results, false, true);

            Utils.matToTexture2D(rgbaMat, texture);
            uiImage.sprite = Sprite.Create(texture, new UnityEngine.Rect(0, 0, texture.width, texture.height), new Vector2(0.5f, 0.5f));

        }
    }


    void OnDestroy()
    {
        // Dispose of the WebCamTexture when the object is destroyed
        if (webCamTexture != null)
            webCamTexture.Stop();
        if (objectDetector != null)
            objectDetector.dispose();

        Utils.setDebugMode(false); //turn off debug mode
    }
}
```

The remainder of this script is much more simple than the *EmotionDetectionScript.cs* this is because I have removed the functionality to auto-rotate. I have done this temporarily as I will be changing the project to be completely portrait-based so I have no need for any extra code. Furthermore, I have not included a ‘detected faces’ text box as this scene aims to focus more on inanimate objects in the user’s surroundings rather than a real person, therefore I can remove the unnecessary logic from the script.

Splash Screen (C#)

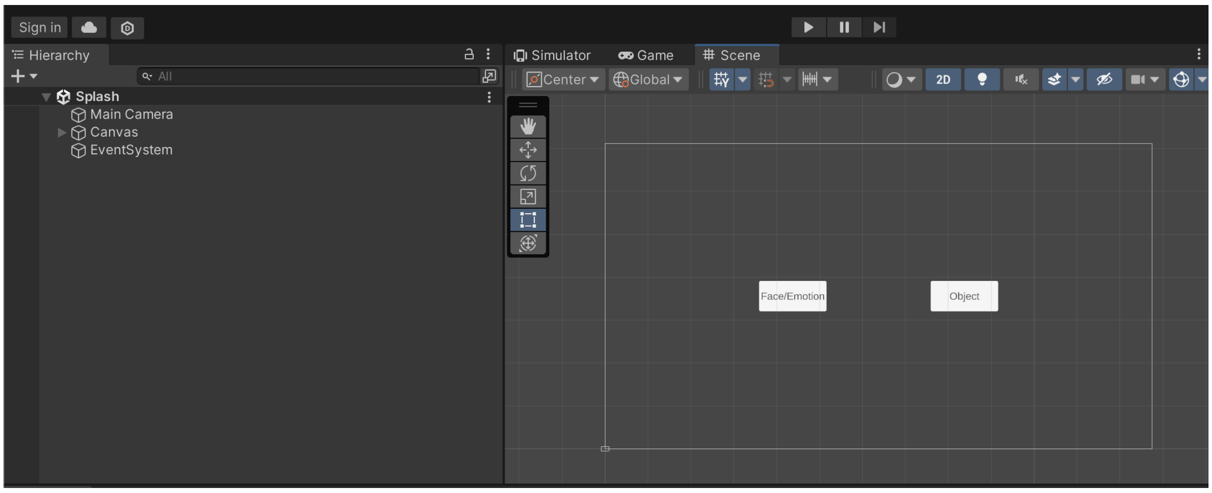
For this part of the project, I wanted to implement a splash screen which the user is greeted to when opening the app, or when wanting to switch scenes. I created the script below that set up 3 public functions that, when a button assigned to it was pressed, would load the desired scene

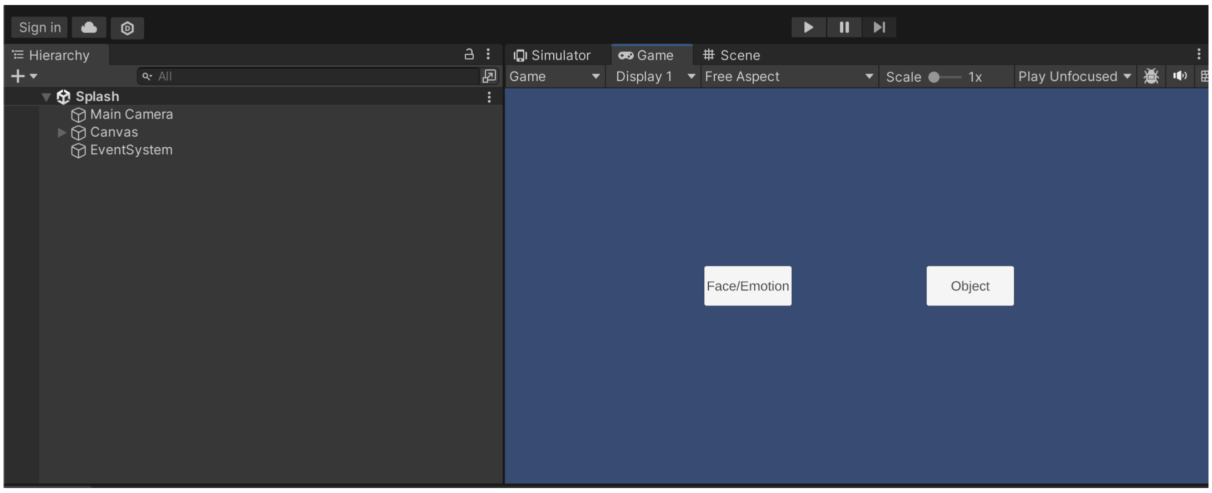
SceneSwitch.cs



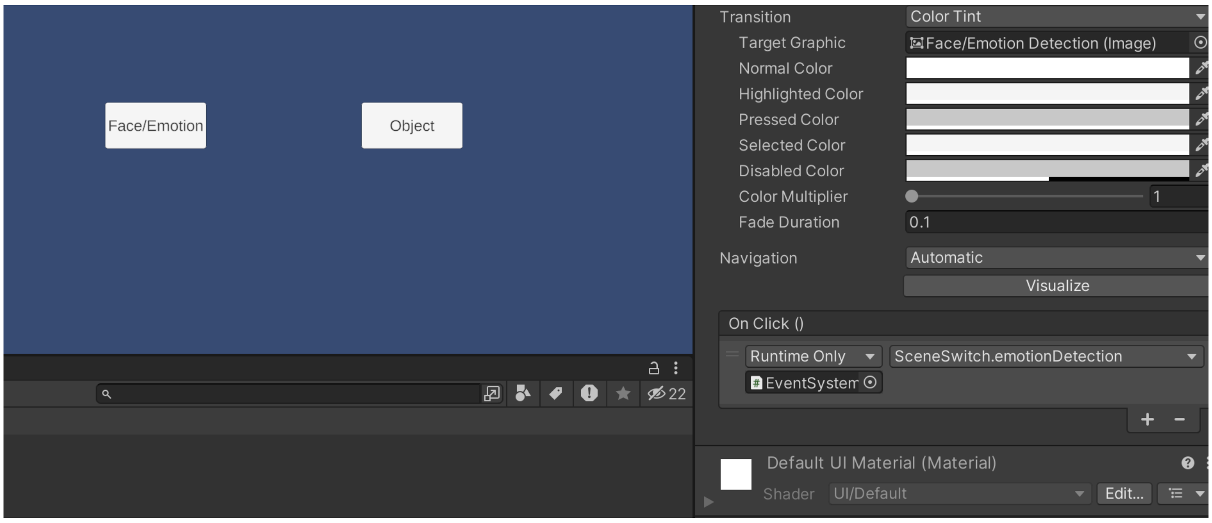
```
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneSwitch : MonoBehaviour
{
    public void emotionDetection()
    {
        SceneManager.LoadScene(sceneName: "Emotion Detection");
    }
    public void objectDetection()
    {
        SceneManager.LoadScene(sceneName: "Object Detection");
    }
    public void Splash()
    {
        SceneManager.LoadScene(sceneName: "Splash");
    }
}

```

I set up a basic splash screen, with no advanced UI for now, that contained 2 buttons, one to take the user to the object detection scene, and one to take them to the emotion detection scene.



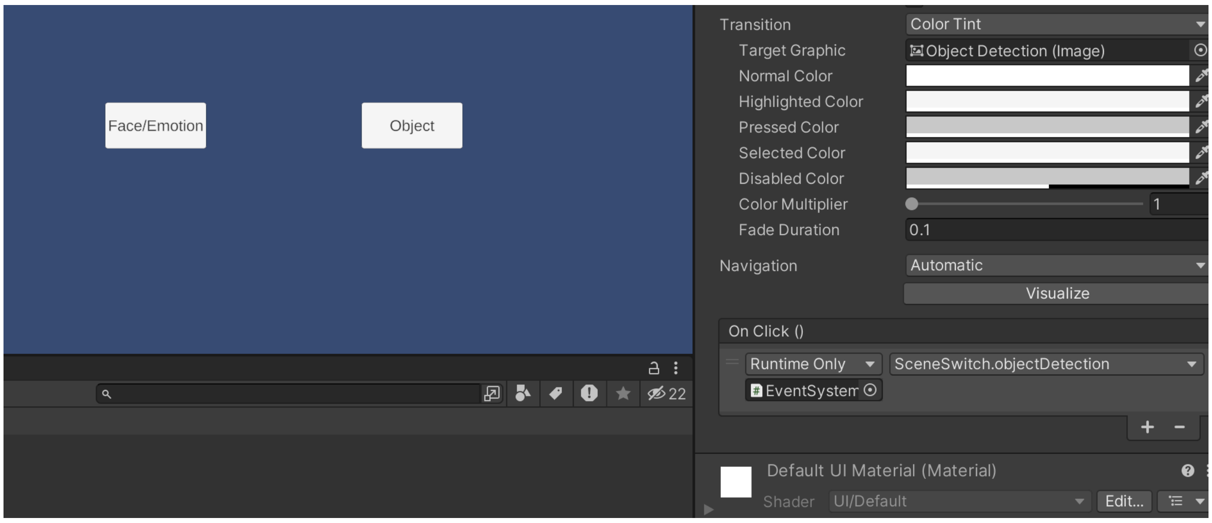


For the face/emotion detection button, it is set to call the emotionDetection() function from SceneSwitch.cs when pressed, taking the user to the emotion detection scene.



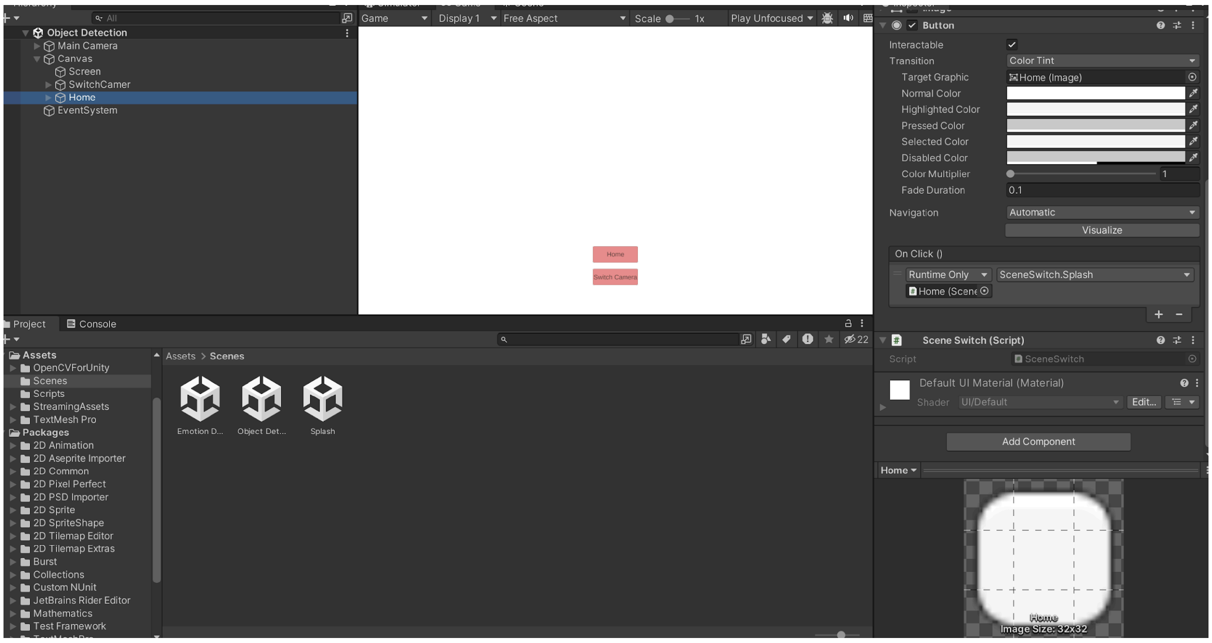
Transition: Color Tint
Target Graphic: Face/Emotion Detection (Image)
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier: 1
Fade Duration: 0.1
Navigation: Automatic
On Click ()
Runtime Only — SceneSwitch.emotionDetection
EventSystem
Default UI Material (Material)
Shader: UI/Default

For the object detection button, it is set to call the objectDetection() function from SceneSwitch.cs when pressed, taking the user to the object detection scene.



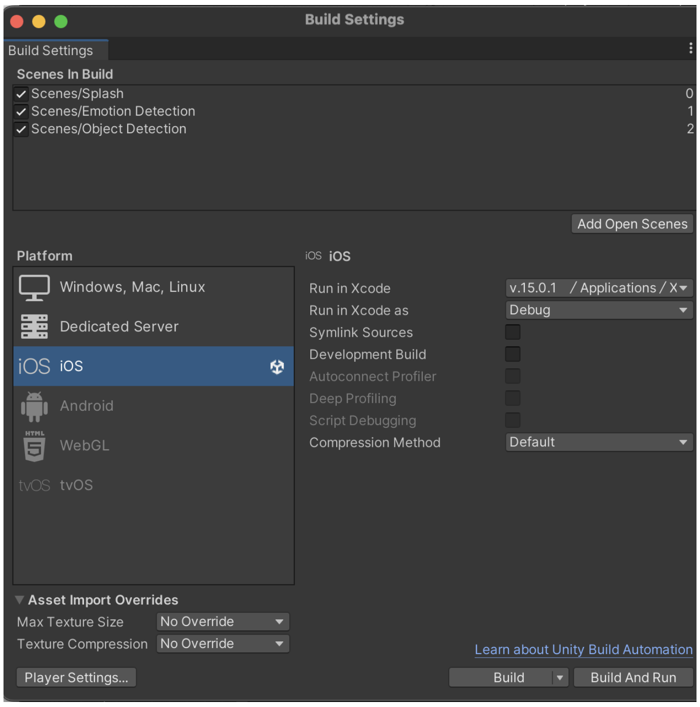
Transition: Color Tint
Target Graphic: Object Detection (Image)
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier: 1
Fade Duration: 0.1
Navigation: Automatic
On Click ()
Runtime Only — SceneSwitch.objectDetection
EventSystem
Default UI Material (Material)
Shader: UI/Default

In the object detection scene and the face detection scene, I have added a button that takes the user back to the splash screen, where they will be able to switch scenes. I will add more functionality to the home screen such as settings, about, help etc. later.


Object Detection
Main Camera
Canvas
Screen
SwitchCamer
Home
EventSystem

Button
Interactable ✓
Transition: Color Tint
Target Graphic: Home (Image)
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier: 1
Fade Duration: 0.1
Navigation: Automatic
On Click ()
Runtime Only — SceneSwitch.Splash
Home (Scene)
Scene Switch (Script)
Script: SceneSwitch
Default UI Material (Material)
Shader: UI/Default

Assets > Scenes: Emotion D..., Object Det..., Splash


Build Settings

Scenes In Build
✓ Scenes/Splash 0
✓ Scenes/Emotion Detection 1
✓ Scenes/Object Detection 2

Platform: iOS
Run in Xcode: v.15.0.1 / Applications / X
Run in Xcode as: Debug
Symlink Sources
Development Build
Autoconnect Profiler
Deep Profiling
Script Debugging
Compression Method: Default

Asset Import Overrides
Max Texture Size: No Override
Texture Compression: No Override

I have added the object detection scene and the splash screen in the ‘Scenes in build’ tab. This is so that everything in these scenes is built when testing in the final build. I have set the splash screen scene at the top; this means that it will be the first to load when the user starts the application, which is my aim.

Image captioning model (Python)

I'll train and use a neural network in Python Tensorflow to best set up the inputs and outputs for my project in order to enable AI-generated image captioning. The dataset I'll be using is the Flickr 8K dataset, which has eight thousand images and sentence descriptions that the AI can use to train itself.

Here’s how the training AI often works:

1) Training Dataset: I will have a dataset that consists of input data (e.g., images, text, numerical values) and corresponding target or output values. This dataset is used to train the AI model.

2) Batches: In reality, the dataset is frequently too big to process all at once. As a result, it is split up into smaller groups called batches. A portion of the training data are contained in each batch.

3) Iterations: The model goes through each batch one at a time during each epoch. This indicates that it undergoes a number of iterations, with each iteration involving a single batch. One full epoch is deemed finished once all the batches have been completed.

4) Modifying Model Weights: The model makes predictions based on the input data in each iteration, calculates the error (the discrepancy between the predicted output and the actual target), and modifies its internal parameters (weights and biases) to minimise this error. The details of these updates depend on the optimization algorithm and loss function that are being applied.

5) Multiple Epochs: The training process typically entails going through several epochs. This enhances the model's capacity to learn from data over time. Typically, several epochs are run until the model's performance on the training set is satisfactory. To prevent overfitting, in which the model becomes overly dependent on the training data, it is crucial to keep track of the model's performance on a different validation dataset.

The number of training epochs is a hyperparameter that will be selected based on my particular problem. A model with too few epochs may not have learned the patterns in the training data and become underfit, whereas a model with too many epochs may become overfit and learn noise from the training data. Usually, this is established through experimentation and tracking the model's effectiveness with respect to validation data.

I used the github repo https://github.com/dabasajay/Image-Caption-Generator as a guide to train and evaluate my training model and created my own training script to accompany this. Firstly, I created 3 scripts called *load_data.py, preprocess.py,* and *model.py* *load_data.py* will load the data from the training images and captions from the flickr8K dataset

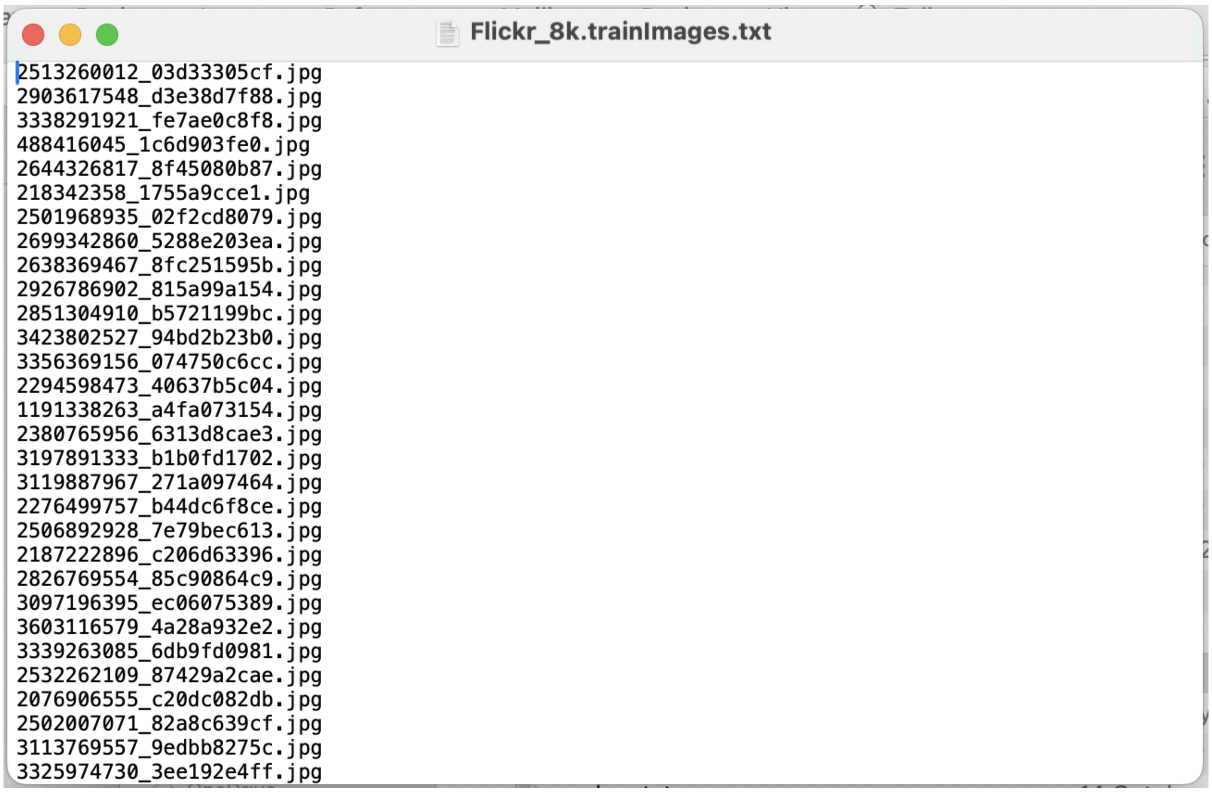
preprocess.py will be in charge of the preprocessing functions that will work alongside the *load_data.py* script and provide some preprocessed data to it.

model.py is used to define the CNN and RNN models and feeds them to the main *train.py*

load_data.py

We have Flickr_8k.trainImages.txt and Flickr_8k.devImages.txt files which consist of unique identifiers(id) which can be used to filter the images and their descriptions

Glimpse of file:



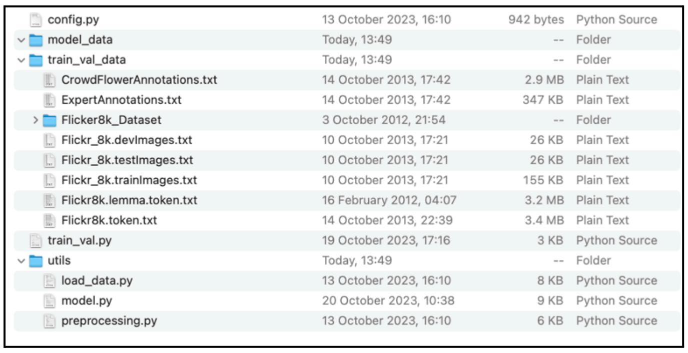
```
2513260012_03d33305cf.jpg
2903617548_d3e38d7f88.jpg
3338291921_fe7ae0c8f8.jpg
488416045_1c6d903fe0.jpg
2644326817_8f45080b87.jpg
218342358_1755a9cce1.jpg
2501968935_02f2cd8079.jpg
2699342860_5288e203ea.jpg
2638369467_8fc251595b.jpg
2926786902_815a99a154.jpg
2851304910_b5721199bc.jpg
3423802527_94bd2b23b0.jpg
3356369156_074750c6cc.jpg
2294598473_40637b5c04.jpg
1191338263_a4fa073154.jpg
2380765956_6313d8cae3.jpg
3197891333_b1b0fd1702.jpg
3119887967_271a097464.jpg
2276499757_b44dc6f8ce.jpg
2506892928_7e79bec613.jpg
2187222896_c206d63396.jpg
2826769554_85c90864c9.jpg
3097196395_ec06075389.jpg
3603116579_4a28a932e2.jpg
3339263085_6db9fd0981.jpg
2532262109_87429a2cae.jpg
2076906555_c20dc082db.jpg
2502007071_82a8c639cf.jpg
3113769557_9edbb8275c.jpg
3325974730_3ee192e4ff.jpg
```

load_data.py

```
import numpy as np
from utils.preprocessing import *
from pickle import load, dump
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
import random
```

The initial imports include utils.preprocessing, pickle, Keras tokenizer and pad sequence and the keras to_categorical function

- utils.preprocessing
  - This is a local script that is called, which includes functions for pre-processing the inputs, that is saved in the utils folder of the Python project as seen below, where the *train_val.py* script is the main script



config.py	13 October 2023, 16:10	942 bytes	Python Source
model_data	Today, 13:49	--	Folder
train_val_data	Today, 13:49	--	Folder
CrowdFlowerAnnotations.txt	14 October 2013, 17:42	2.9 MB	Plain Text
ExpertAnnotations.txt	14 October 2013, 17:42	347 KB	Plain Text
Flicker8k_Dataset	3 October 2012, 21:54	--	Folder
Flickr_8k.devImages.txt	10 October 2013, 17:21	26 KB	Plain Text
Flickr_8k.testImages.txt	10 October 2013, 17:21	26 KB	Plain Text
Flickr_8k.trainImages.txt	10 October 2013, 17:21	155 KB	Plain Text
Flickr8k.lemma.token.txt	16 February 2012, 04:07	3.2 MB	Plain Text
Flickr8k.token.txt	14 October 2013, 22:39	3.4 MB	Plain Text
train_val.py	19 October 2023, 17:16	3 KB	Python Source
utils	Today, 13:49	--	Folder
load_data.py	13 October 2023, 16:10	8 KB	Python Source
model.py	20 October 2023, 10:38	9 KB	Python Source
preprocessing.py	13 October 2023, 16:10	6 KB	Python Source

- pickle
  - Pickle is used to save and load Python objects, which is especially useful when working with machine learning models and data preprocessing, as it allows for the efficient storage and retrieval of complex data structures. It is also used to load preprocessed data from files.
  - ‘pickle.load’ is used to load Python objects (such as dictionaries or lists) that were previously saved to a file using pickle.dump. In the script, it is used to load preprocessed data, such as image features and the tokenizer, which have been saved to binary files.
  - ‘pickle.dump’ function is used to save the Keras Tokenizer object to a binary file using pickle. The tokenizer is a crucial component in natural language processing tasks, and it's saved to be reused in the future without having to train it again.
- Keras tokenizer
  - The Keras Tokenizer is used for text preprocessing and encoding. Its primary purpose is to prepare the textual data (captions) for consumption by a machine learning model.
  - It:
    - Accepts a dictionary of captions where each image id is associated with a list of captions.
    - Flattens these captions into a single list.
    - Uses the Keras Tokenizer to learn a consistent mapping from words to unique integer values. This means it assigns a unique integer to each word in the vocabulary.
- pad_sequence function
  - The ‘pad_sequences’ function from the Keras library is used to ensure that the input sequences (captions) are of the same length. This is a common

preprocessing step when working with sequence data and is important when training machine learning models, particularly neural networks.
    - The main purpose of ‘pad_sequences’ is to ensure that all input sequences (captions) have the same length as the ‘max_length’.
- to_categorical function
    - The ‘to_categorical’ function from the Keras library is used to perform one-hot encoding on the output words (target words) of the caption data. One-hot encoding is a technique that converts categorical data into a binary format, which is necessary when training neural networks for classification tasks, including sequence generation tasks like captioning.

load_data.py

```python
# Function to load a set of image identifiers from a file
def load_set(filename):
    # Open the file and read contents then close
    file = open(filename, 'r')
    doc = file.read()
    file.close()

    # Create an empty list called ids to store image identifiers
    ids = list()

    # Process the file content line by line
    for line in doc.split('\n'):
        # Skip empty lines
        if len(line) < 1:
            continue
        # Get the image identifier (id) by splitting the line at the first full stop (.)
        _id = line.split('.')[0]
        ids.append(_id)

    # Return a set of unique image identifiers
    return set(ids)
```

This function loads a set of image identifiers (ids) from a text file specified by ‘filename’. It reads the file, splits it by newline characters, and stores the ids in a Python set. The function returns the set of ids.

load_data.py

```python
def load_cleaned_captions(filename, ids):

        # Open the file, read the contents, then close
        file = open(filename, 'r')
        doc = file.read()
        file.close()

        # Create a dictionary called 'captions' that will hold the captions of each image
        captions = dict()
        _count = 0

        # Process the file line by line
        for line in doc.split('\n'):
                # Split line on white space
                tokens = line.split()

                # Split the id from caption
                image_id, image_caption = tokens[0], tokens[1:]

                # Skip the images if they are not in the ids set
                if image_id in ids:
                        # Create list
                        if image_id not in captions:
                                captions[image_id] = list()
                        # Wrap caption in start & end tokens
                        caption = '<start> ' + ' '.join(image_caption) + ' <end>'
                        # Store
                        captions[image_id].append(caption)
                        _count = _count+1 # Increment the counter of how many captions there are
        return captions, _count # Return the dictionary of cleaned captions and the count of captions loaded
```

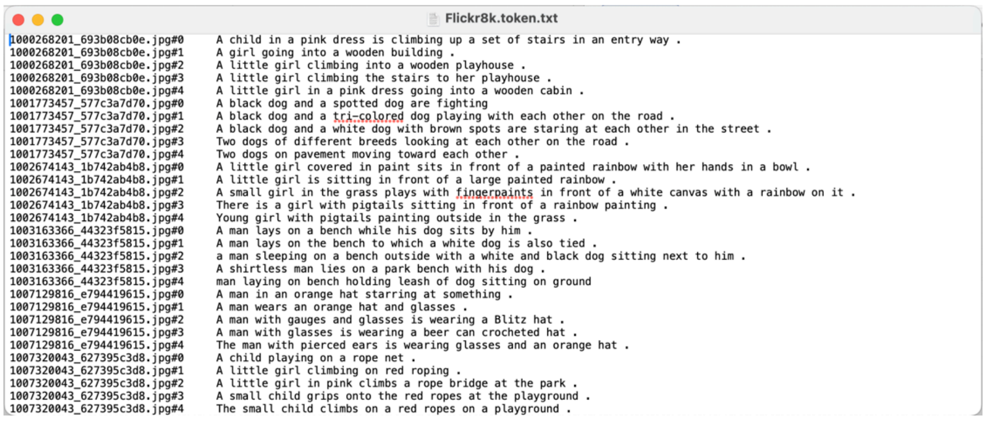
This function loads cleaned captions for images from a text file specified by ‘filename’, which is the txt file that contains all the names of the images (Flickr_8K.testImages.txt). It reads the file, processes it line by line, and pairs the image ids with their corresponding captions, which are read from a .txt file that contains the captions for each image (Flickr8K.token.txt). The ids argument is used to filter out images that are not present in the ids set. The function returns a dictionary where each image id is associated with a list of captions and the count of captions loaded.

The model that will be developed will generate a caption for any given image and the caption will be generated one word at a time. The sequence of previously generated words will be provided as input. Therefore, we will need a ‘first word’ to kick-off the generation process and a ‘last word‘ to signal the end of the caption. I will use the strings ‘<start>‘ and ‘<end>‘ for this purpose. These tokens are added to the captions as they are loaded.

- It is important to do this now before we encode the text so that the tokens are also encoded correctly.

This is a glimpse of the file where the captions will be cleaned and pre-processed:

Flickr8K.token.txt



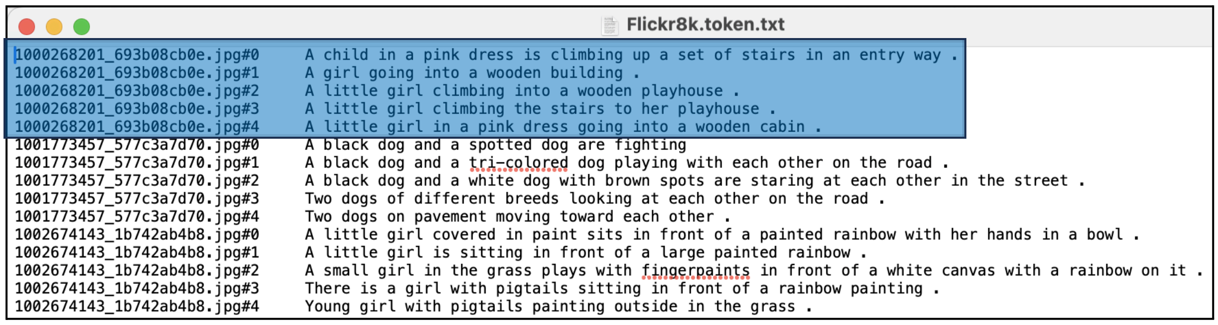
```
1000268201_693b08cb0e.jpg#0	A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg#1	A girl going into a wooden building .
1000268201_693b08cb0e.jpg#2	A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg#3	A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg#4	A little girl in a pink dress going into a wooden cabin .
1001773457_577c3a7d70.jpg#0	A black dog and a spotted dog are fighting
1001773457_577c3a7d70.jpg#1	A black dog and a tri-colored dog playing with each other on the road .
1001773457_577c3a7d70.jpg#2	A black dog and a white dog with brown spots are staring at each other in the street .
1001773457_577c3a7d70.jpg#3	Two dogs of different breeds looking at each other on the road .
1001773457_577c3a7d70.jpg#4	Two dogs on pavement moving toward each other .
1002674143_1b742ab4b8.jpg#0	A little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .
1002674143_1b742ab4b8.jpg#1	A little girl is sitting in front of a large painted rainbow .
1002674143_1b742ab4b8.jpg#2	A small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on it .
1002674143_1b742ab4b8.jpg#3	There is a girl with pigtails sitting in front of a rainbow painting .
1002674143_1b742ab4b8.jpg#4	Young girl with pigtails painting outside in the grass .
1003163366_44323f5815.jpg#0	A man lays on a bench while his dog sits by him .
1003163366_44323f5815.jpg#1	A man lays on the bench to which a white dog is also tied .
1003163366_44323f5815.jpg#2	a man sleeping on a bench outside with a white and black dog sitting next to him .
1003163366_44323f5815.jpg#3	A shirtless man lies on a park bench with his dog .
1003163366_44323f5815.jpg#4	man laying on bench holding leash of dog sitting on ground
1007129816_e794419615.jpg#0	A man in an orange hat starring at something .
1007129816_e794419615.jpg#1	A man wears an orange hat and glasses .
1007129816_e794419615.jpg#2	A man with gauges and glasses is wearing a Blitz hat .
1007129816_e794419615.jpg#3	A man with glasses is wearing a beer can crocheted hat .
1007129816_e794419615.jpg#4	The man with pierced ears is wearing glasses and an orange hat .
1007320043_627395c3d8.jpg#0	A child playing on a rope net .
1007320043_627395c3d8.jpg#1	A little girl climbing on red roping .
1007320043_627395c3d8.jpg#2	A little girl in pink climbs a rope bridge at the park .
1007320043_627395c3d8.jpg#3	A small child grips onto the red ropes at the playground .
1007320043_627395c3d8.jpg#4	The small child climbs on a red ropes on a playground .
```

load_data.py

```
# Function to load image features and filter based on provided ids
def load_image_features(filename, ids):

    # Load all the image features from a binary file
    all_features = load(open(filename, 'rb'))

    # Filter the features to include only those with ids present in the provided set
    features = {_id: all_features[_id] for _id in ids}

    # Return the filtered features
    return features


# Function to convert a dictionary of captions into a flat list of every caption
def to_lines(captions):
    everyCaption = list()  # Initialize a list to store all the captions
    for image_id in captions.keys():

        # Append each caption from the dictionary to the list
        [everyCaption.append(caption) for caption in captions[image_id]]
    # Return the list of every captions
    return everyCaption
```

The function ‘load_image_features()’ function loads the image features from the Flickr8K txt file and loads all the features and filters them to include only those whose ids match the ones in the ids set. It returns a dictionary where each image id is associated with its image features/caption.

After that, the ‘to_lines()’ function converts the dictionary of the captions, generated from ‘load_cleaned_captions()’ and connects all the captions of an associated image id into a single list. This is because each image consists of multiple captions, for better learning as seen below:


Flickr8k.token.txt

```
1000268201_693b08cb0e.jpg#0	A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg#1	A girl going into a wooden building .
1000268201_693b08cb0e.jpg#2	A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg#3	A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg#4	A little girl in a pink dress going into a wooden cabin .
1001773457_577c3a7d70.jpg#0	A black dog and a spotted dog are fighting
1001773457_577c3a7d70.jpg#1	A black dog and a tri-colored dog playing with each other on the road .
1001773457_577c3a7d70.jpg#2	A black dog and a white dog with brown spots are staring at each other in the street .
1001773457_577c3a7d70.jpg#3	Two dogs of different breeds looking at each other on the road .
1001773457_577c3a7d70.jpg#4	Two dogs on pavement moving toward each other .
1002674143_1b742ab4b8.jpg#0	A little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .
1002674143_1b742ab4b8.jpg#1	A little girl is sitting in front of a large painted rainbow .
1002674143_1b742ab4b8.jpg#2	A small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on it .
1002674143_1b742ab4b8.jpg#3	There is a girl with pigtails sitting in front of a rainbow painting .
1002674143_1b742ab4b8.jpg#4	Young girl with pigtails painting outside in the grass .
```

load_data.py

```
# Function to create and fit a Keras Tokenizer on captions
def create_tokenizer(captions):
    # Convert the dictionary of captions into a flat list
    lines = to_lines(captions)
    # Initialize a Keras Tokenizer
    tokenizer = Tokenizer()
    # Fit the tokenizer on the list of captions
    tokenizer.fit_on_texts(lines)
    # Return the fitted tokenizer
    return tokenizer


# Function to calculate the maximum length of captions with the most words
def calc_max_length(captions):

    # Convert the dictionary of captions into a flat list
    lines = to_lines(captions)

    # Calculate the maximum length by finding the length of the longest caption
    return max(len(line.split()) for line in lines)
```

The captions will need to be encoded to numbers before it can be presented to the model. The first step in encoding the captions is to create a consistent mapping from words to unique integer values.

Keras provides the Tokenizer class that can learn this mapping from the loaded captions.

Thus, the ‘create_tokenizer()’ function will fit a tokenizer on given captions.

Afterwards, the function ‘calc_max_length()’ function calculates the maximum length of captions (in terms of the number of words) in the provided captions. It's used to determine the maximum sequence length for the model.

An AI image captioning model typically needs a maximum length for generated captions for several reasons:

1) Practicality: Without a maximum length, the generated captions could potentially become extremely long and unwieldy, making them less useful for real-world applications.
2) Computational efficiency: Limiting the length of captions also helps with computational efficiency. It reduces the amount of time and resources required for caption generation, making the process faster and more scalable.
3) Consistency: Setting a maximum length helps in maintaining consistency in the length of generated captions across different images. This consistency can be valuable for design and presentation purposes.

This is done by simply evaluating the length of the longest caption in the list and setting that as the max value.

load_data.py

```python
# Create sequences of images, input sequences and output words for an image
def create_sequences(tokenizer, max_length, captions_list, image):

        # Initialize lists to store data

        # InpImage : input for image features
        # InpText : input for text features
        # OutWord  : output word
        InpImage = list()
        InpText = list()
        OutWord = list()


        # Determine the vocabulary size
        vocab_siz = len(tokenizer.word_index) + 1

        # Walk through each caption for the image
        for caption in captions_list:

                # Encode the sequence (this is done via keras' built-in function)
                sequence = tokenizer.texts_to_sequences([caption])[0]

                # Split one sequence into multiple Input, and Output pairs
                for i in range(1, len(sequence)):
                        # Split into input and output pair
                        in_seq, out_seq = sequence[:i], sequence[i]

                        # Pad input sequence
                        in_seq = pad_sequences([in_seq], maxlen=max_length)[0]

                        # Encode output sequence
                        out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]

                        # Store data to each list
                        InpImage.append(image)
                        InpText.append(in_seq)
                        OutWord.append(out_seq)
                        |
        return InpImage, InpText, OutWord
```

This function creates sequences of images, input sequences, and output words for a given image. It prepares the data for training the caption generation model.

Each caption will be split into words. The model will be provided one word & the image and it generates the next word. Then the first two words of the caption will be provided to the model as input with the image to generate the next word. This is how the model will be trained.

For example, the input sequence “little girl running in field” would be split into 6 input-output pairs to train the model:

```
InpImage                InpText(text sequence)                                   OutWord(word)
---------------------------------------------------------------------------------------------------------------
image                    <start>, ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ little
image                    <start>, little, ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ girl
image                    <start>, little, girl, ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ running
image                    <start>, little, girl, running, ~~~~~~~~~~~~~~~~~~~~~ in
image                    <start>, little, girl, running, in, ~~~~~~~~~~~~~~~~~ field
image                    <start>, little, girl, running, in, field, ~~~~~~~~~~ <end>
```

load_data.py

```
# Data generator, intended to be used in a call to model.fit_generator()
def data_generator(images, captions, tokenizer, max_length, batch_size, random_seed):
        # Setting random seed for reproducibility of results
        random.seed(random_seed)

        # Get a list of image ids
        image_ids = list(captions.keys())
        _count=0 # Initialize a private counter variable

        while True:
        if _count >= len(image_ids): #Checks if the generator as exceeded the number of images in training data
                        # Generator exceeded or reached the end so restart it
                        _count = 0

                # Lists to store data for batches
                input_img_batch = list()
                input_sequence_batch = list()
                output_word_batch = list()


        for i in range(_count, min(len(image_ids), _count+batch_size)):

                        # Retrieve the image id
                        image_id = image_ids[i]

                        # Retrieve the image features
                        image = images[image_id][0]

                        # Retrieve the captions list
                        captions_list = captions[image_id]

                        # Shuffle captions list
                        random.shuffle(captions_list)


                        # Create sequences for the batch and add to respective lists
                        input_img, input_sequence, output_word = create_sequences(tokenizer, max_length, captions_list, image)

                        for j in range(len(input_img)):
                                input_img_batch.append(input_img[j])
                                input_sequence_batch.append(input_sequence[j])
                                output_word_batch.append(output_word[j])
                _count = _count + batc_size
        yield [[np.array(input_img_batch), np.array(input_sequence_batch)], np.array(output_word_batch)]
```

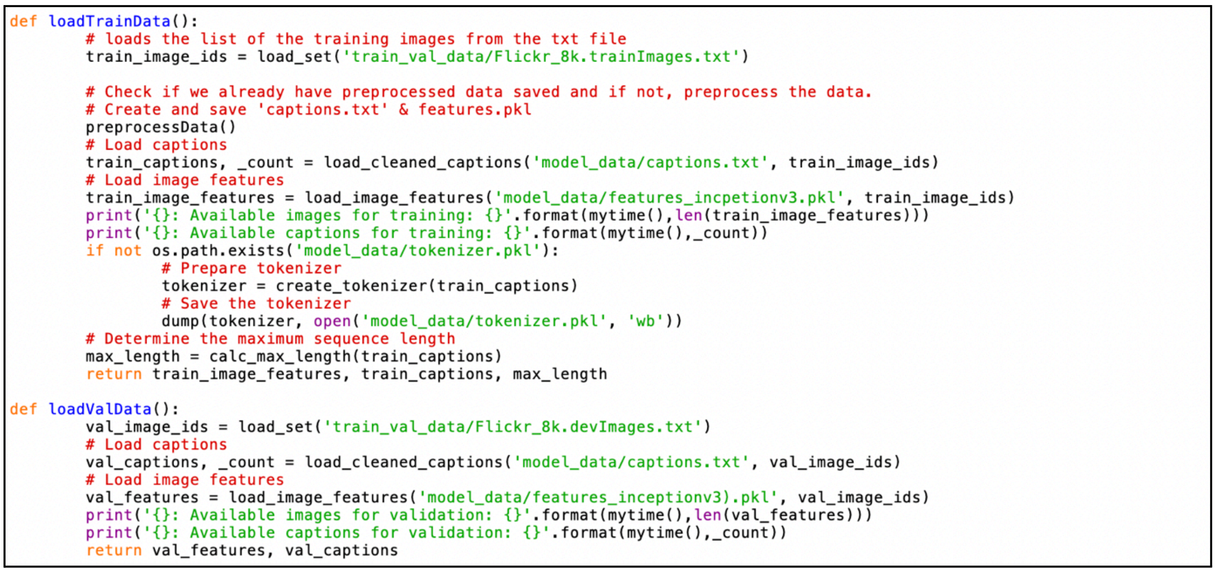
The ‘data_generator()’ defines a generator to be used with Keras’s model.fit_generator()’. It generates batches of data for the model. The generator then shuffles the captions for each image and then generates input and output sequences for training.

It does this by getting a list of all the image ids and loops through the batches, for each image, retrieving the image id , features and caption list. It then shuffles through the caption list. This step randomises the order in which the captions are used for training in each batch. Shuffling helps prevent the model from overfitting to a specific sequence of captions. creates input sequences and output words for the current image and its shuffled captions. The ‘create_sequences()’ function is called to generate these sequences, and the result is stored in ‘input_img’, ‘input_sequence’, and ‘output_word’. Then a nested loop iterates through each generated sequence within the current batch of images. Afterwards, each corresponding batch list is appended with the generated sequences.

At the end, the function yields a batch of training data as a generator output. The yield statement is used to return the batch of data. The batch is a list of two elements: a list of input data (image features and input sequences) and the corresponding output data (output words). This format is suitable for use with the model.fit_generator() function in Keras. The data is converted to NumPy arrays to ensure that it's in the correct format for training deep learning models.

A private ‘count’ variable is used throughout this to keep track of the position in the dataset for the next batch

load_data.py



```
def loadTrainData():
        # loads the list of the training images from the txt file
        train_image_ids = load_set('train_val_data/Flickr_8k.trainImages.txt')

        # Check if we already have preprocessed data saved and if not, preprocess the data.
        # Create and save 'captions.txt' & features.pkl
        preprocessData()
        # Load captions
        train_captions, _count = load_cleaned_captions('model_data/captions.txt', train_image_ids)
        # Load image features
        train_image_features = load_image_features('model_data/features_incpetionv3.pkl', train_image_ids)
        print('{}: Available images for training: {}'.format(mytime(),len(train_image_features)))
        print('{}: Available captions for training: {}'.format(mytime(),_count))
        if not os.path.exists('model_data/tokenizer.pkl'):
                # Prepare tokenizer
                tokenizer = create_tokenizer(train_captions)
                # Save the tokenizer
                dump(tokenizer, open('model_data/tokenizer.pkl', 'wb'))
        # Determine the maximum sequence length
        max_length = calc_max_length(train_captions)
        return train_image_features, train_captions, max_length

def loadValData():
        val_image_ids = load_set('train_val_data/Flickr_8k.devImages.txt')
        # Load captions
        val_captions, _count = load_cleaned_captions('model_data/captions.txt', val_image_ids)
        # Load image features
        val_features = load_image_features('model_data/features_inceptionv3).pkl', val_image_ids)
        print('{}: Available images for validation: {}'.format(mytime(),len(val_features)))
        print('{}: Available captions for validation: {}'.format(mytime(),_count))
        return val_features, val_captions
```

loadTrainData()

This function is responsible for loading and preparing the training datasets. This loads a set of image identifiers from the training using keras's 'load_set' function. This set of image identifiers will be used to filter images and their corresponding captions.

It then calls the 'preprocessData()' function to run the preprocessing of the data. This function is defined and called from the *preprocess.py* script.

The cleaned captions are then loaded from a file named *captions.txt* – the load_cleaned_captions() function was defined earlier in the script and its purpose is to read the captions from the file and filter them to match the provided image identifiers.

The image features are then loaded from the .pkl file that is generated after the image features are extracted (*preprocessing.py).*

The code checks whether a tokenizer file ('tokenizer.pkl') exists in the directory specified by 'model_data_path'. If it doesn't exist, it proceeds to create a tokenizer using the 'create_tokenizer' function, which learns a mapping from words to unique integer values based on the provided captions. The tokenizer is then saved to the 'tokenizer.pkl' file using the 'dump' function.

It finally calculates the maximum sequence length of captions in the training data and returns the filtered training image features, captions, and the maximum sequence length.

loadValData()

Firstly, the function loads a set of image identifiers from the validation data set using the ‘load_set’ function. Similar to the ‘loadTrainData()’ function, this set of image identifiers will be used to filter images and their corresponding captions for validation.

It is then followed by loading the cleaned captions similar to the loadTrainData() function.

The function then loads image features from a binary file named ‘model_data/features_inceptionv3.pkl'. features are loaded for the images specified in the ‘val_image_ids’ set. The function ‘load_image_features()’ reads the features and filters them based on the provided image identifiers.

Finally, the function returns the filtered validation image features and captions.

preprocessing.py

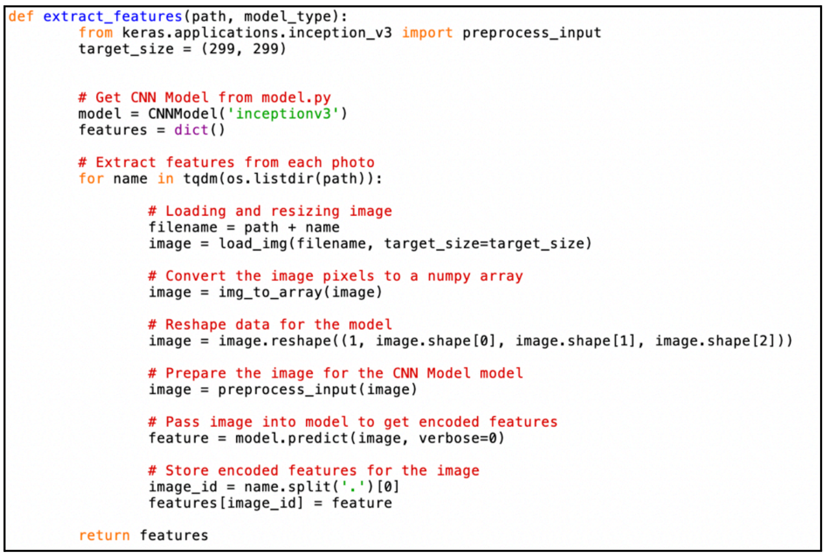
preprocessing.py

```
import numpy as np
import os
from pickle import dump
import string
from tqdm import tqdm
from model import CNNModel
from keras.preprocessing.image import load_img, img_to_array
from datetime import datetime as dt


#Function used for displaying time besides the printed output in the console
def mytime():
        _str = ''
        _str = str(dt.now().hour)+':'+str(dt.now().minute)+':'+str(dt.now().second)
        return _str

        #E.g. output looks like:
            #12:56:18: Preprocessing complete
            #12:56:19: Parsed and completed successfully
```

The preprocessing.py script is in charge of preprocessing the data; this involves, loading and resizing the images, converting the image pixels to a NumPy array, and extracting the features from each image using a pre-built incpetionv3 model, where the features are encoded and saved to ‘inceptionv3.pkl’. It also manages the cleaning of captions which is done by reading the captions file, splitting each caption via the white space, and removing unnecessary details within the caption (e.g. punctuation, special characters, uppercase characters etc.)

The myTime() function as seen above is used throughout this program. Its functionality is to display the current time before a printed output. This is so that I can see timestamps of when each function was executed as the training may take a few hours, therefore it will be left unattended. In order to see how long training takes and other functions, it will be beneficial to log the timestamps.

preprocessing.py



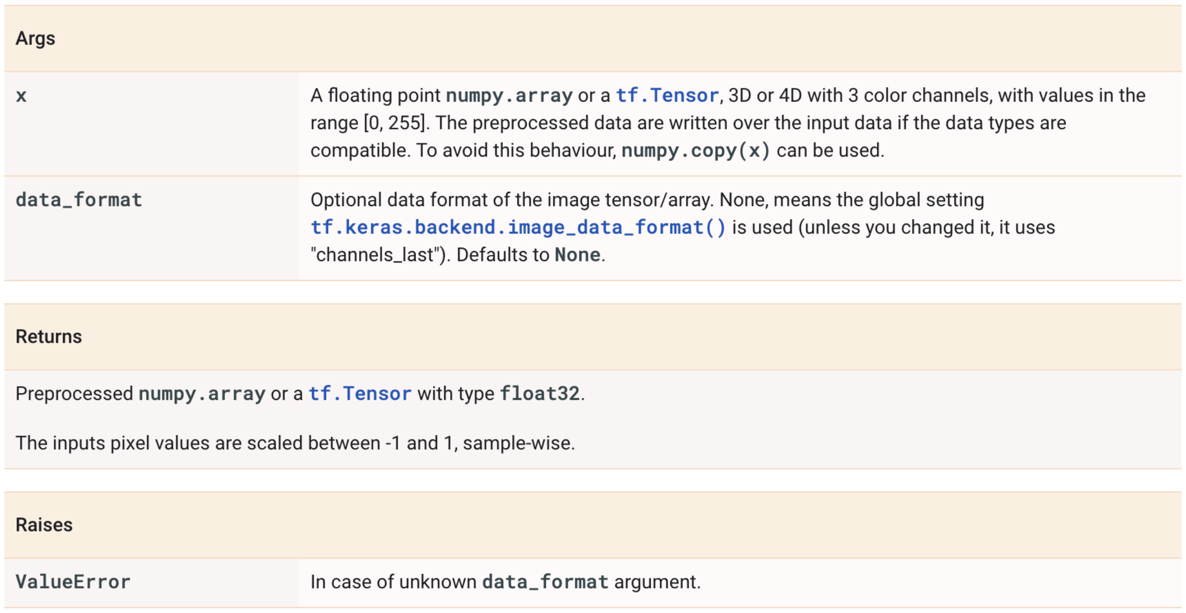
```python
def extract_features(path, model_type):
        from keras.applications.inception_v3 import preprocess_input
        target_size = (299, 299)


        # Get CNN Model from model.py
        model = CNNModel('inceptionv3')
        features = dict()

        # Extract features from each photo
        for name in tqdm(os.listdir(path)):

                # Loading and resizing image
                filename = path + name
                image = load_img(filename, target_size=target_size)

                # Convert the image pixels to a numpy array
                image = img_to_array(image)

                # Reshape data for the model
                image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))

                # Prepare the image for the CNN Model model
                image = preprocess_input(image)

                # Pass image into model to get encoded features
                feature = model.predict(image, verbose=0)

                # Store encoded features for the image
                image_id = name.split('.')[0]
                features[image_id] = feature

        return features
```

This function uses the inceptionv3 model to extract the features from each image. Before it does this, it must load the image, convert the pixels into an array that the model will be able to decode and work on, it must also reshape the data; this means that the organization of rows and columns in the array must be reshaped to fit the model. Finally, this reshaped array is sent through the preprocess_input() function which is imported from the inceptionv3 model as tf.keras.applications.inception_v3.preprocess_input

This function preprocesses a tensor or Numpy array encoding a batch of images.

preprocess_input() takes in 2 args that will return a preprocessed NumPy or tf.Tensor with type float32 where The inputs pixel values are scaled between -1 and 1, sample-wise.

Source:
(https://www.tensorflow.org/api_docs/python/tf/keras/applications/inception_v3/preprocess_input)



Args	
`x`	A floating point `numpy.array` or a `tf.Tensor`, 3D or 4D with 3 color channels, with values in the range [0, 255]. The preprocessed data are written over the input data if the data types are compatible. To avoid this behaviour, `numpy.copy(x)` can be used.
`data_format`	Optional data format of the image tensor/array. None, means the global setting `tf.keras.backend.image_data_format()` is used (unless you changed it, it uses "channels_last"). Defaults to `None`.

Returns
Preprocessed `numpy.array` or a `tf.Tensor` with type `float32`. The inputs pixel values are scaled between -1 and 1, sample-wise.

Raises	
`ValueError`	In case of unknown `data_format` argument.

The new preprocessed image is passed through the model which then returns a dictionary of form:

```
{
        image_id1 :  image_features 1
        image_id2 :  image_features 2
}
```

preprocess.py

```
def load_captions(filename):
        file = open(filename, 'r')
        doc = file.read()
        file.close()


        captions = dict()

        # Process lines by line
        _count = 0
        for line in doc.split('\n'):
                # Split line on white space
                tokens = line.split()
                if len(line) < 2:
                        continue

                # Take the first token as the image id, the rest as the caption
                image_id, image_caption = tokens[0], tokens[1:]

                # Extract filename from image id
                image_id = image_id.split('.')[0]

                # Convert caption tokens back to caption string
                image_caption = ' '.join(image_caption)

                # Create the list if needed
                if image_id not in captions:
                        captions[image_id] = list()

                # Store caption
                captions[image_id].append(image_caption)
                _count = _count+1

        return captions
```

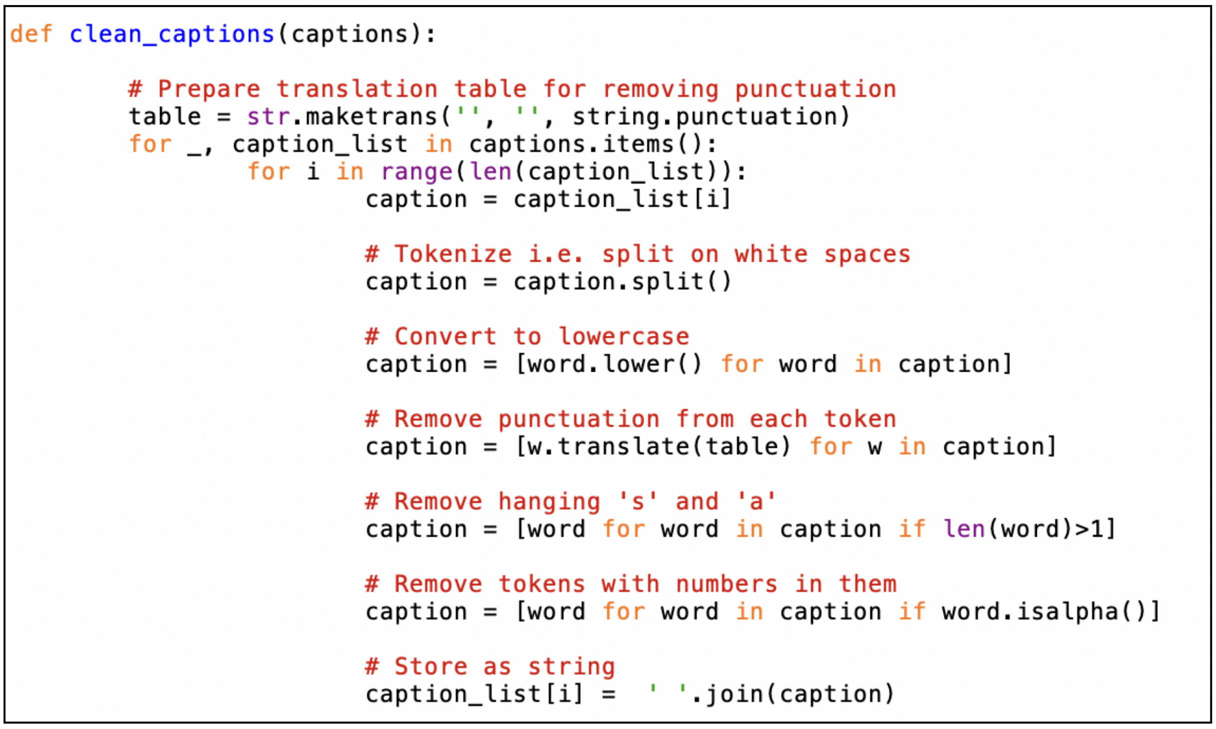
The load_captions() function loads the captions from the inputted text file. This file is read and the contents are organised into a dictionary of form

```
{
        image_id1 : [caption1, caption2, etc],
        image_id2 : [caption1, caption2, etc],
        ...
}
```

The dictionary is processed line by line, extracting the image id and caption from the tokens. The filename of each image is extracted from the image id by separating the filename from the '.png'

It returns a dictionary where the keys are image IDs, and the values are lists of captions for each image.

preprocess.py



```python
def clean_captions(captions):

        # Prepare translation table for removing punctuation
        table = str.maketrans('', '', string.punctuation)
        for _, caption_list in captions.items():
                for i in range(len(caption_list)):
                        caption = caption_list[i]

                        # Tokenize i.e. split on white spaces
                        caption = caption.split()

                        # Convert to lowercase
                        caption = [word.lower() for word in caption]

                        # Remove punctuation from each token
                        caption = [w.translate(table) for w in caption]

                        # Remove hanging 's' and 'a'
                        caption = [word for word in caption if len(word)>1]

                        # Remove tokens with numbers in them
                        caption = [word for word in caption if word.isalpha()]

                        # Store as string
                        caption_list[i] =  ' '.join(caption)
```

The clean_captions() function cleans and preprocesses the loaded captions. It performs several text preprocessing steps, including:

- Tokenization (splitting captions into words).
- Conversion to lowercase.
- Removal of punctuation.
- Removal of single characters like 's' and 'a'.
- Removal of tokens containing numbers.

Then, It updates the input 'captions' dictionary in place by modifying the captions to be clean and preprocessed.

preprocess.py

```
def save_captions(captions, filename):
        lines = list()
        for key, captions_list in captions.items():
                for caption in captions_list:
                        lines.append(key + ' ' + caption)
        data = '\n'.join(lines)
        file = open(filename, 'w')
        file.write(data)
        file.close()
```

This function saves the cleaned captions to captions.txt

After saving, captions.txt is of form: `id` `caption`

Example: 2252123185_487f21e336 stadium full of people watch game

2252123185_487f21e336.png

preprocess.py

```python
def preprocessData():
    # Print the current time and specify the model being used
    print('{}: Using {} model'.format(mytime(), 'inceptionv3'.title()))

    # Check if image features have already been generated and saved in a file
    if os.path.exists('model_data/features_inceptionv3.pkl'):
        # If the file exists, print a message indicating it's already available
        print('{}: Image features already generated at {}'.format(mytime(), 'model_data/features_inceptionv3.pkl'))
    else:
        # If the file does not exist, generate image features using the InceptionV3 model
        print('{}: Generating image features using inceptionv3 model...'.format(mytime()))
        features = extract_features('train_val_data/Flicker8k_Dataset/')

        # Save the generated features to a file
        dump(features, open('model_data/features_inceptionv3.pkl', 'wb'))
        # Print a message indicating the completion of feature extraction and saving
        print('{}: Completed & Saved features for {} images successfully'.format(mytime(), len(features)))

    # Check if captions have already been parsed and saved in a file
    if os.path.exists('model_data/captions.txt'):
        # If the file exists, print a message indicating it's already available
        print('{}: Parsed caption file already generated at {}'.format(mytime(), 'model_data/captions.txt'))
    else:
        # If the file does not exist, parse the captions file
        print('{}: Parsing captions file...'.format(mytime()))

        # Load captions from the source file 'train_val_data/Flickr8k.token.txt'
        captions = load_captions('train_val_data/Flickr8k.token.txt')

        # Save the parsed captions to a file named 'model_data/captions.txt'
        save_captions(captions, 'model_data/captions.txt')
        # Print a message indicating the completion of parsing and saving the captions
        print('{}: Parsed & Saved successfully'.format(mytime()))
```

The final function in *preprocess.py* is the preprocessData() function. This is in charge of orchestrating the preprocessing of both image features and captions.

It checks if preprocessed data files already exist (image features and parsed captions) and, if not, generates them. If image features are not generated, it uses the InceptionV3 model to extract features from images and saves them to a file by calling the extract_features() functions, passing the folder of images (‘train_val_data/Flicker8K_Dataset/’) as the path to the images.

If captions are not parsed, it loads and organizes captions and saves them in a specific format by calling the load_captions() and save_captions() functions, inputting the Flickr8k.token.txt file as the ‘captions’ parameter and ‘captions.txt’ as the ‘filename’ parameter

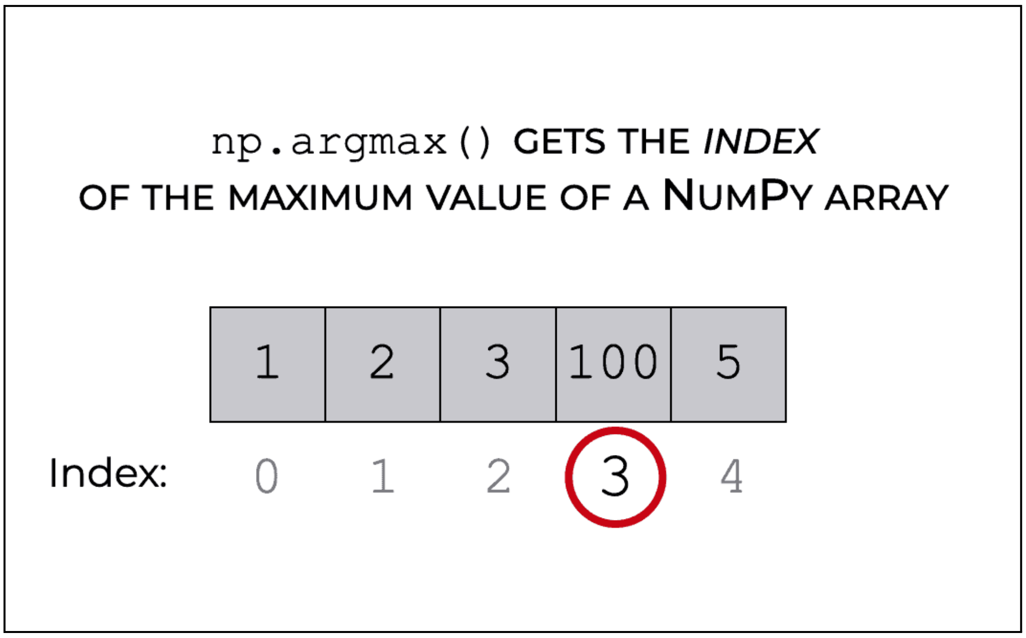
model.py

model.py is a predefined script that is in charge of defining CNN and RNN models. It first defines a CNN model for image feature extraction using InceptionV3. Then defines two RNN models for caption generation, one with a standard architecture and another with an alternative architecture. For this project, I will be using the '*alternativeRNNModel*' as this updated architecture seems to have a higher accuracy score with others.

The script provides functions to generate captions for images using both argmax and beam search methods and to evaluate the generated captions against ground truth captions using BLEU scores.

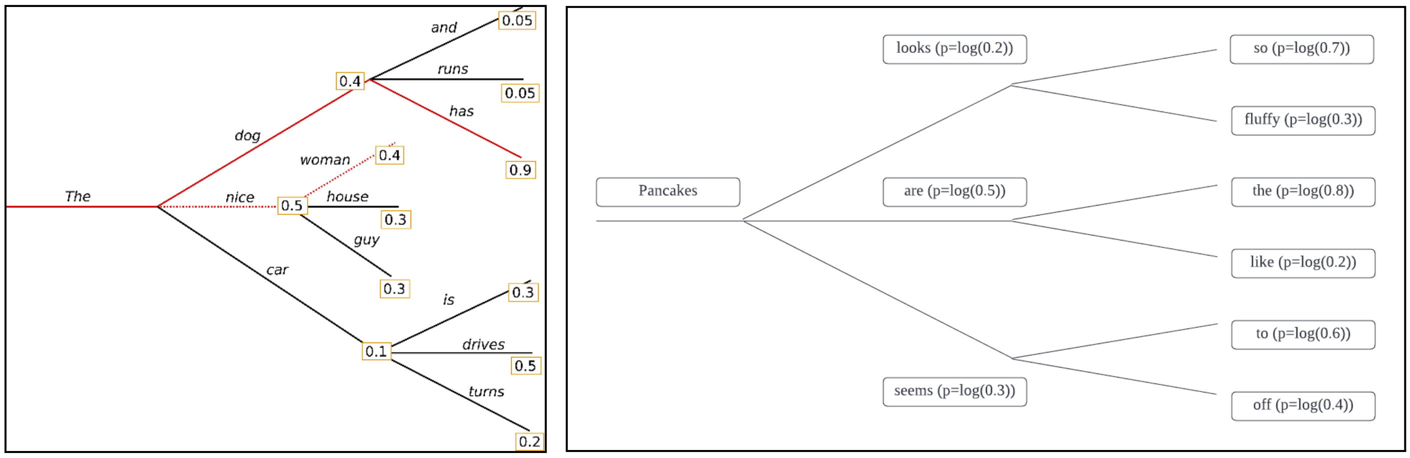
Argmax is a method that selects the output with the highest probability at each step of a sequence. In the context of text generation, it chooses the word with the maximum predicted probability as the next word in the sequence. This method is straightforward but may result in overly deterministic and less diverse output.

Argmax


np.argmax() GETS THE INDEX OF THE MAXIMUM VALUE OF A NUMPY ARRAY
1
2
3
100
5
Index:
0
1
2
3
4

Beam Search, on the other hand, is a more sophisticated method for text generation. It considers multiple potential next words at each step (controlled by a parameter called "beam width"). Instead of selecting the single word with the highest probability, it maintains a list of top candidates based on their cumulative probabilities. The beam search algorithm explores multiple paths, and when the sequence is complete or "<end>" is encountered, it selects the path with the highest overall probability.

Visualised beam search algorithm



model.py

```
import numpy as np
from keras.applications.inception_v3 import InceptionV3
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Input, Dense, Dropout, LSTM, Embedding, concatenate, RepeatVector, TimeDistributed, Bidirectional
from keras.preprocessing.sequence import pad_sequences
from tqdm import tqdm

from nltk.translate.bleu_score import corpus_bleu


def CNNModel():
        model = InceptionV3()
        model.layers.pop()
        model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
        return model


def RNNModel(vocab_size, max_len):

        embedding_size = 300
        LSTM_units = 256
        dense_units = 256
        dropout = 0.3


        # InceptionV3 outputs a 2048 dimensional vector for each image, which we'll feed to RNN Model
        image_input = Input(shape=(2048,))

        image_model_1 = Dropout(dropout)(image_input)
        image_model = Dense(embedding_size, activation='relu')(image_model_1)

        caption_input = Input(shape=(max_len,))
        # mask_zero: We zero pad inputs to the same length, the zero mask ignores those inputs. E.g. it is an efficiency.
        caption_model_1 = Embedding(vocab_size, embedding_size, mask_zero=True)(caption_input)
        caption_model_2 = Dropout(dropout)(caption_model_1)
        caption_model = LSTM(LSTM_units)(caption_model_2)

        # Merging the models and creating a softmax classifier
        final_model_1 = concatenate([image_model, caption_model])
        final_model_2 = Dense(dense_units, activation='relu')(final_model_1)
        final_model = Dense(vocab_size, activation='softmax')(final_model_2)

        model = Model(inputs=[image_input, caption_input], outputs=final_model)
        model.compile(loss='categorical_crossentropy', optimizer='adam')
        return model
```

The script begins with importing numpy and keras libraries and functions, followed by nltk.translate.bleu_score

- BLEU, or the Bilingual Evaluation Understudy, is a score for comparing a candidate translation of text to one or more reference translations.
- Although developed for translation, it can be used to evaluate text generated for a suite of natural language processing tasks.
    - Source: (‘https://machinelearningmastery.com/calculate-bleu-score-for-text-python/’)

A perfect match between generated and reference text results in a BLEU score of 1.0, while a perfect mismatch yields a score of 0.0. BLEU scores are used to quantitatively assess the quality, accuracy, and fluency of generated text, providing a numerical measure of how well the generated text aligns with the reference text. Higher BLEU scores indicate better performance.

CNNModel()

This function defines a Convolutional Neural Network (CNN) model for image feature extraction. It uses the InceptionV3 model from Keras, which is a pre-trained deep-learning model designed for image classification and feature extraction. It removes the last layer of the InceptionV3 model and creates a new model that takes image inputs and outputs a 2048-dimensional vector.

RNNModel()

This function defines a Recurrent Neural Network (RNN) model for generating captions for images. It takes three arguments: vocab_size (size of the vocabulary) and max_len (maximum caption length). It creates two input layers for the image and the caption sequences. It applies dropout to the image input to prevent overfitting and a dense layer to reduce the dimension of the image features to the specified embedding_size. For the caption input, it uses an embedding layer to convert word indices to dense vectors, applies dropout, and then uses an LSTM layer for the sequential processing of the captions. The image and caption models are merged using concatenation and passed through dense layers to make predictions. The model is compiled with a categorical cross-entropy loss function and the Adam optimizer. The resulting model is returned for generating image captions.

model.py

```
def AlternativeRNNModel(vocab_size, max_len):

        embedding_size = 300
        LSTM_units = 256
        dense_units = 256
        dropout = 0.3

        # InceptionV3 outputs a 2048 dimensional vector for each image, which we'll feed to RNN Model
        image_input = Input(shape=(2048,))

        image_model_1 = Dense(embedding_size, activation='relu')(image_input)
        image_model = RepeatVector(max_len)(image_model_1)

        caption_input = Input(shape=(max_len,))
        # mask_zero: We zero pad inputs to the same length, the zero mask ignores those inputs. E.g. it is an efficiency.
        caption_model_1 = Embedding(vocab_size, embedding_size, mask_zero=True)(caption_input)
        # Since we are going to predict the next word using the previous words
        # (length of previous words changes with every iteration over the caption), we have to set return_sequences = True.
        caption_model_2 = LSTM(LSTM_units, return_sequences=True)(caption_model_1)
        caption_model = TimeDistributed(Dense(embedding_size))(caption_model_2)

        # Merging the models and creating a softmax classifier
        final_model_1 = concatenate([image_model, caption_model])
        final_model_2 = Bidirectional(LSTM(LSTM_units, return_sequences=False))(final_model_1)
        final_model = Dense(vocab_size, activation='softmax')(final_model_2)

        model = Model(inputs=[image_input, caption_input], outputs=final_model)
        model.compile(loss='categorical_crossentropy', optimizer='adam')

        return model
```

This function defines an alternative RNN model with a different architecture for generating captions. It takes the same arguments as the previous function. This model uses a different approach for handling captions: it predicts the next word based on the previous words and image features.

model.py

```
def int_to_word(integer, tokenizer):
        for word, index in tokenizer.word_index.items():
                if index == integer:
                        return word
        return None
```

the ‘int_to_word’ function is a simple lookup function that converts an integer (typically a word index) back into its corresponding word in the vocabulary based on the tokenizer. This is useful as I want to convert the numerical output of a model into a human-readable text description.

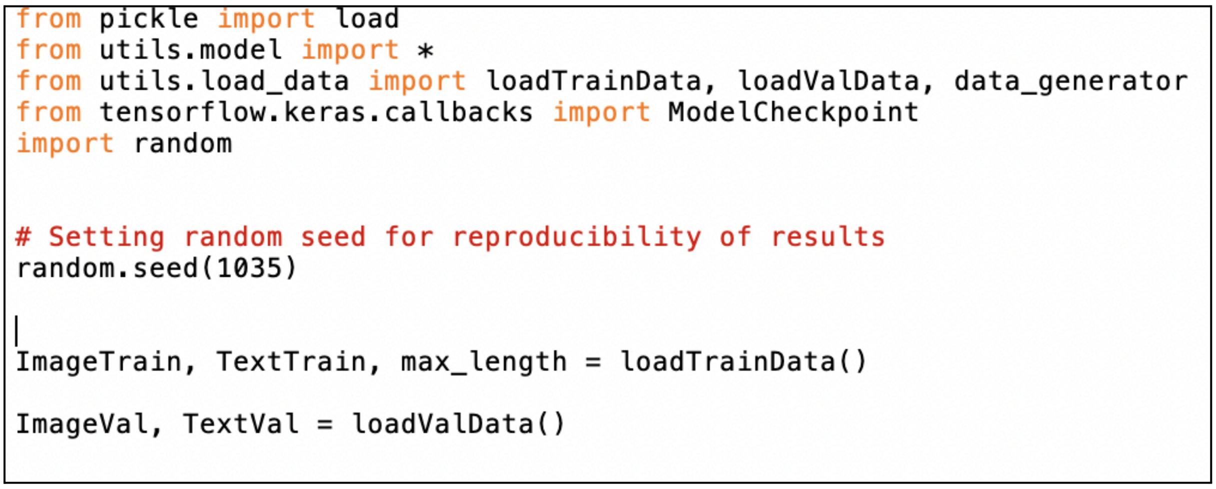
model.py

```
def generate_caption_beam_search(model, tokenizer, image, max_length, beam_index=3):
        # in_text --> [[idx,prob]] ;prob=0 initially
        in_text = [[tokenizer.texts_to_sequences(['<start>'])[0], 0.0]]
        while len(in_text[0][0]) < max_length:
                tempList = []
                for seq in in_text:
                        padded_seq = pad_sequences([seq[0]], maxlen=max_length)
                        preds = model.predict([image,padded_seq], verbose=0)
                        # Take top (i.e. which have highest probailities) `beam_index` predictions
                        top_preds = np.argsort(preds[0])[-beam_index:]
                        # Getting the top `beam_index` predictions and
                        for word in top_preds:
                                next_seq, prob = seq[0][:], seq[1]
                                next_seq.append(word)
                                # Update probability
                                prob += preds[0][word]
                                # Append as input for generating the next word
                                tempList.append([next_seq, prob])
                in_text = tempList
                # Sorting according to the probabilities
                in_text = sorted(in_text, reverse=False, key=lambda l: l[1])
                # Take the top words
                in_text = in_text[-beam_index:]
        in_text = in_text[-1][0]
        final_caption_raw = [int_to_word(i,tokenizer) for i in in_text]
        final_caption = []
        for word in final_caption_raw:
                if word=='<end>':
                        break
                else:
                        final_caption.append(word)
        final_caption.append('<end>')
        return ' '.join(final_caption)



"""
        *Evaluate the model on BLEU Score using beam search predictions
"""
def evaluate_model_beam_search(model, images, captions, tokenizer, max_length, beam_index=3):
        actual, predicted = list(), list()
        for image_id, caption_list in tqdm(captions.items()):
                yhat = generate_caption_beam_search(model, tokenizer, images[image_id], max_length, beam_index=beam_index)
                ground_truth = [caption.split() for caption in caption_list]
                actual.append(ground_truth)
                predicted.append(yhat.split())
        print('BLEU Scores :')
        print('A perfect match results in a score of 1.0, whereas a perfect mismatch results in a score of 0.0.')
        print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
        print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
        print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0)))
        print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))
```

These 2 functions are called at the end of training the new model. They use a beam-search algorithm which considers multiple possible next words (beam_index) at each step. It maintains a list of potential captions along with their probabilities and selects the top candidates based on the highest probability. This process continues until the caption reaches the maximum length or "<end>" is encountered, where the max_length refers to the maximum length of the captions in the training data. This model uses the newly trained RNN model and is evaluated using the BLEU score.

train.py

train.py



```
from pickle import load
from utils.model import *
from utils.load_data import loadTrainData, loadValData, data_generator
from tensorflow.keras.callbacks import ModelCheckpoint
import random


# Setting random seed for reproducibility of results
random.seed(1035)


ImageTrain, TextTrain, max_length = loadTrainData()

ImageVal, TextVal = loadValData()
```

These lines import necessary modules and functions. It imports modules like pickle, custom utility functions and configuration parameters from *config.py* and *rnnConfig* from *config.py*. It also imports the random module and sets the random seed for reproducibility.

A random seed is a starting point for generating random numbers or sequences in a pseudo-random number generator (PRNG). A random seed is used to initialize the random number generator in the context of AI training and machine learning in order to produce repeatable results. In machine learning and AI, experiments need to be reproducible. When training models, one must be able to run the same training process again with the same hyperparameters and data and get the same results. This is crucial for debugging, model comparisons, and sharing research. When something goes wrong during training, having a fixed random seed allows the isolation of issues. The experiment can be rerun with the same seed to see if the problem persists. If it does, it's likely not due to randomness but to a problem in the code or data.

Then, the script loads **training** data, which includes image features (ImageTrain), text features (captions) (TextTrain), and the maximum caption length (max_length). This data is loaded and generated by the loadTrainData() function in the load_data.py script

load_data.py

```
def loadTrainData():
        # loads the list of the training images from the txt file
        train_image_ids = load_set('train_val_data/Flickr_8k.trainImages.txt')

        # Check if we already have preprocessed data saved and if not, preprocess the data.
        # Create and save 'captions.txt' & features.pkl
        preprocessData()
        # Load captions
        train_captions, _count = load_cleaned_captions('model_data/captions.txt', train_image_ids)
        # Load image features
        train_image_features = load_image_features('model_data/features_incpetionv3.pkl', train_image_ids)
        print('{}: Available images for training: {}'.format(mytime(),len(train_image_features)))
        print('{}: Available captions for training: {}'.format(mytime(),_count))
        if not os.path.exists('model_data/tokenizer.pkl'):
                # Prepare tokenizer
                tokenizer = create_tokenizer(train_captions)
                # Save the tokenizer
                dump(tokenizer, open('model_data/tokenizer.pkl', 'wb'))
        # Determine the maximum sequence length
        max_length = calc_max_length(train_captions)
        return train_image_features, train_captions, max_length
```

(function returns image features, image captions, and max caption length in training images)

After loading the training data, the next line loads the validation data (ImageVal) and (TextVal) from the same script, using the loadValData() function.

load_data.py

```
def loadValData():
        val_image_ids = load_set('train_val_data/Flickr_8k.devImages.txt')
        # Load captions
        val_captions, _count = load_cleaned_captions('model_data/captions.txt', val_image_ids)
        # Load image features
        val_features = load_image_features('model_data/features_inceptionv3).pkl', val_image_ids)
        print('{}: Available images for validation: {}'.format(mytime(),len(val_features)))
        print('{}: Available captions for validation: {}'.format(mytime(),_count))
        return val_features, val_captions
```

(function returns validation image features and their captions)

train.py

```
#load the tokenizer generated
tokenizer = load(open('model_data/tokenizer.pkl', 'rb'))
vocab_size = len(tokenizer.word_index) + 1



#Now that we have the image features from CNN model, we need to feed them to a RNN Model.


#Define the RNN model

model = AlternativeRNNModel(vocab_size, max_length)
print('RNN Model (Decoder) Summary : ')
print(model.summary())
```

Here, the code loads a tokenizer using the pickle.load() function. This tokenizer is created during the create_tokenizer() function which is called from the loadTrainData() function in the load_data.py script.

It then calculates the vocabulary size based on the tokenizer's word index.

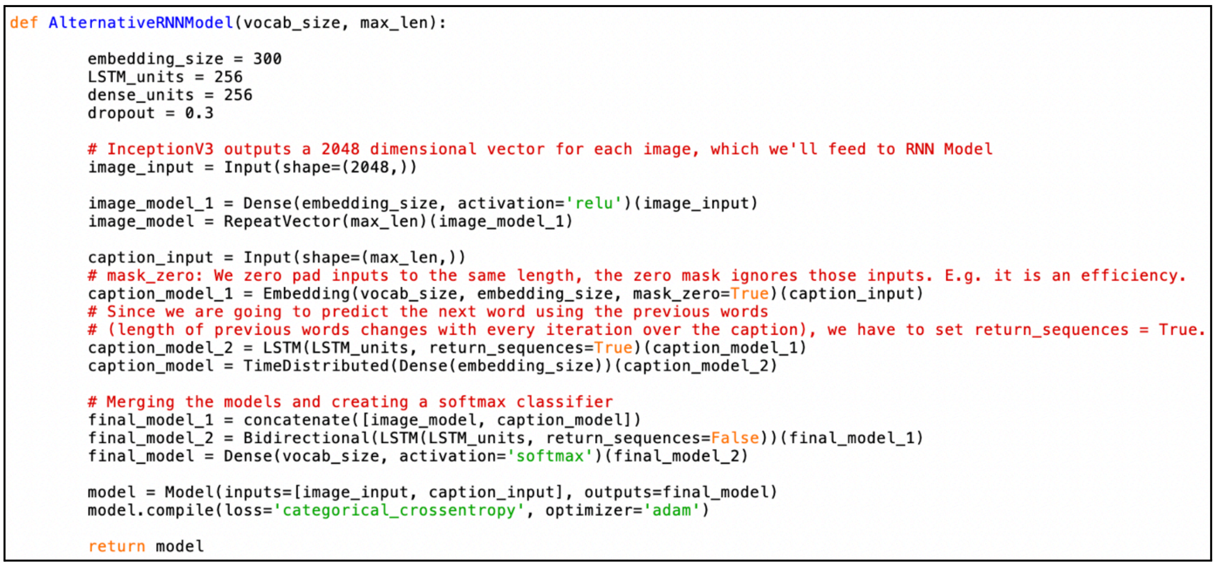
In the context of AI training and natural language processing, a tokenizer is a tool or component that divides text into smaller units, most frequently words or subwords. It is a crucial component of text preprocessing and is employed in AI training for a variety of tasks, particularly those involving language understanding and creation.

The next line creates an instance of the AlternativeRNNModel class with specific parameters like vocabulary size and maximum caption length. This is the model that will be used for the image captioning.

In this training model, 2 potential models can be used, ‘RNNModel’ or ‘AlternativeRNNModel’. For the best results, according to the guide I am following, the AlternativeRNNModel architecture provides more accurate results, although it may use more memory to train, which isn’t an issue for me, thus, I opted to use it for my project.

This model is defined and created from the model.py:

model.py



```
def AlternativeRNNModel(vocab_size, max_len):

        embedding_size = 300
        LSTM_units = 256
        dense_units = 256
        dropout = 0.3

        # InceptionV3 outputs a 2048 dimensional vector for each image, which we'll feed to RNN Model
        image_input = Input(shape=(2048,))

        image_model_1 = Dense(embedding_size, activation='relu')(image_input)
        image_model = RepeatVector(max_len)(image_model_1)

        caption_input = Input(shape=(max_len,))
        # mask_zero: We zero pad inputs to the same length, the zero mask ignores those inputs. E.g. it is an efficiency.
        caption_model_1 = Embedding(vocab_size, embedding_size, mask_zero=True)(caption_input)
        # Since we are going to predict the next word using the previous words
        # (length of previous words changes with every iteration over the caption), we have to set return_sequences = True.
        caption_model_2 = LSTM(LSTM_units, return_sequences=True)(caption_model_1)
        caption_model = TimeDistributed(Dense(embedding_size))(caption_model_2)

        # Merging the models and creating a softmax classifier
        final_model_1 = concatenate([image_model, caption_model])
        final_model_2 = Bidirectional(LSTM(LSTM_units, return_sequences=False))(final_model_1)
        final_model = Dense(vocab_size, activation='softmax')(final_model_2)

        model = Model(inputs=[image_input, caption_input], outputs=final_model)
        model.compile(loss='categorical_crossentropy', optimizer='adam')

        return model
```

The difference between ‘RNNModel’ and ‘AlternativeRNNModel’ is that ‘RNNModel’ uses a simpler architecture with a single LSTM for caption generation, while ‘AlternativeRNNModel’ uses a more complex architecture with Bidirectional LSTM and TimeDistributed layers. The choice between these models depends on the specific requirements of the image captioning task and the trade-off between model complexity and performance. During testing, I will analyse the performance of the ‘AlternativeRNNModel’ and if necessary, I will switch and re-train to ‘RNNModel’ which will provide a simpler and less performance-demanding model, paired with an optimal epoch size of 11.

Train.py

```
#Train the model save after each epoch

num_of_epochs = 20
batch_size = 64
steps_train = len(X2train)//batch_size
if len(X2train)%batch_size!=0:
    steps_train = steps_train+1
steps_val = len(X2val)//batch_size
if len(X2val)%batch_size!=0:
    steps_val = steps_val+1
model_save_path = 'model_data/model_inceptionv3_epoch-{epoch:02d}_train_loss-{loss:.4f}_val_loss-{val_loss:.4f}.hdf5"
checkpoint = ModelCheckpoint(model_save_path, monitor='val_loss', verbose=1, save_best_only=True, mode='min')
callbacks = [checkpoint]

print('steps_train: {}, steps_val: {}'.format(steps_train,steps_val))
print('Batch Size: {}'.format(batch_size))
print('Total Number of Epochs = {}'.format(num_of_epochs))
```

This section of the code sets up training-related parameters and configurations. It specifies the number of training epochs (num_of_epochs), and batch size (batch_size). According to the guide I am referencing for this part of the project, the most optimal number of epochs is 20, paired with a batch size of 64.

It then calculates the number of training and validation steps based on the dataset size and batch size. It also defines the path for saving model checkpoints, including placeholders for epoch number, training loss, and validation loss in the file name. A ModelCheckpoint callback is created to save the best models during training, and a list of callbacks is defined.

Train.py

```

# Shuffle train data
ids_train = list(TextTrain.keys())
random.shuffle(ids_train)
TextTrain_shuffled = {_id: TextTrain[_id] for _id in ids_train}
TextTrain = TextTrain_shuffled



# Create the train data generator
# returns [[img_features, text_features], out_word]
generator_train = data_generator(ImageTrain, TextTrain, tokenizer, max_length, batch_size, 1035)


# Create the validation data generator
# returns [[img_features, text_features], out_word]
generator_val = data_generator(ImageVal, TextVal, tokenizer, max_length, batch_size, 1035)
```

This part of the code shuffles the training data to introduce randomness during training. It first retrieves the keys (identifiers) of the training data, shuffles them using random.shuffle, and creates a shuffled version of the training data dictionary. The shuffling ensures that the training data is presented to the model in a random order during each epoch.
The code then creates a data generator for training data using the data_generator function. This generator will yield batches of data in the format [[img_features, text_features], out_word]. It takes image features (ImageTrain), text data (TextTrain), tokenizer, maximum caption length, batch size, and the random seed (1035) as inputs.

It generates batches of data for the model. The generator then shuffles the captions for each image and generates input and output sequences for training, as discussed above when breaking down the load_data() function.

Similarly, a data generator is created for the validation data (generator_val) with the same structure as the training data generator.

load_data.py

```
# Data generator, intended to be used in a call to model.fit_generator()
def data_generator(images, captions, tokenizer, max_length, batch_size, random_seed):
        # Setting random seed for reproducibility of results
        random.seed(random_seed)

        # Get a list of image ids
        image_ids = list(captions.keys())
        _count=0 # Initialize a private counter variable

        while True:
        if _count >= len(image_ids): #Checks if the generator as exceeded the number of images in training data
                        # Generator exceeded or reached the end so restart it
                        _count = 0

                # Lists to store data for batches
                input_img_batch = list()
                input_sequence_batch = list()
                output_word_batch = list()


        for i in range(_count, min(len(image_ids), _count+batch_size)):

                        # Retrieve the image id
                        image_id = image_ids[i]

                        # Retrieve the image features
                        image = images[image_id][0]

                        # Retrieve the captions list
                        captions_list = captions[image_id]

                        # Shuffle captions list
                        random.shuffle(captions_list)


                        # Create sequences for the batch and add to respective lists
                        input_img, input_sequence, output_word = create_sequences(tokenizer, max_length, captions_list, image)

                        for j in range(len(input_img)):
                                input_img_batch.append(input_img[j])
                                input_sequence_batch.append(input_sequence[j])
                                output_word_batch.append(output_word[j])
                _count = _count + batc_size
        yield [[np.array(input_img_batch), np.array(input_sequence_batch)], np.array(output_word_batch)]
```

Train.py

```
# Fit for one epoch
model.fit_generator(generator_train,
            epochs=num_of_epochs,
            steps_per_epoch=steps_train,
            validation_data=generator_val,
            validation_steps=steps_val,
            callbacks=callbacks,
            verbose=1)
```

This part of the code initiates model training using the fit_generator method. It trains the model for the specified number of epochs (num_of_epochs) and follows the specified number of steps for both training and validation sets. It uses the previously defined callbacks, such as saving the best models.

train.py

```
#Evaluate the model on validation data and ouput BLEU score

print('Model trained successfully. Running model on validation set for calculating BLEU score using BEAM search with k=3')
evaluate_model_beam_search(model, ImageVal, TextVal, tokenizer, max_length, beam_index=3)
```

Finally, it will evaluate the trained model on the validation data and calculate BLEU scores using beam search with k=3. It then calls the evaluate_model_beam_search() function in the model.py script to perform the evaluation and display the results.

model.py

```
def evaluate_model_beam_search(model, images, captions, tokenizer, max_length, beam_index=3):
	actual, predicted = list(), list()
	for image_id, caption_list in tqdm(captions.items()):
		yhat = generate_caption_beam_search(model, tokenizer, images[image_id], max_length, beam_index=beam_index)
		ground_truth = [caption.split() for caption in caption_list]
		actual.append(ground_truth)
		predicted.append(yhat.split())
	print('BLEU Scores :')
	print('A perfect match results in a score of 1.0, whereas a perfect mismatch results in a score of 0.0.')
	print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
	print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
	print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0)))
	print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))
```

Model summary

Many parameters and variables for this program were referenced from this file to provide the best possible results;

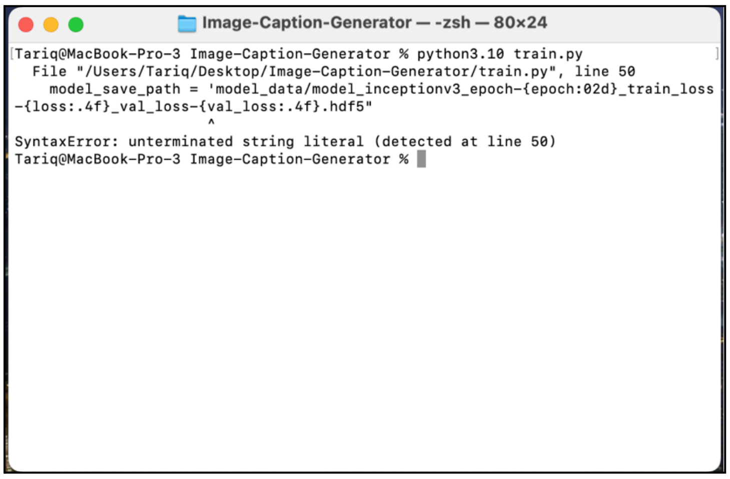
```
config = {
	'images_path': 'train_val_data/Flicker8k_Dataset/',
	'train_data_path': 'train_val_data/Flickr_8k.trainImages.txt',
	'val_data_path': 'train_val_data/Flickr_8k.devImages.txt',
	'captions_path': 'train_val_data/Flickr8k.token.txt',
	'tokenizer_path': 'model_data/tokenizer.pkl',
	'model_data_path': 'model_data/',
	'model_load_path': 'model_data/model_inceptionv3_epoch-20_train_loss-2.4050_val_loss-3.0527.hdf5',
	'num_of_epochs': 20,
	'max_length': 40,
	'batch_size': 64,
	'beam_search_k':3,
	'test_data_path': 'test_data/',
	'model_type': 'inceptionv3',
	'random_seed': 1035
}

rnnConfig = {
	'embedding_size': 300,
	'LSTM_units': 256,
	'dense_units': 256,
	'dropout': 0.3
}
```

When running *train.py* the program will initialize the training of the AI model to be used in my greater program. It will be saved as a weights file that can be used in other Python projects to seamlessly take an input and produce an output.

Training

Errors when running:

Error 1:



```
Tariq@MacBook-Pro-3 Image-Caption-Generator % python3.10 train.py
  File "/Users/Tariq/Desktop/Image-Caption-Generator/train.py", line 50
    model_save_path = 'model_data/model_inceptionv3_epoch-{epoch:02d}_train_loss
-{loss:.4f}_val_loss-{val_loss:.4f}.hdf5"
                      ^
SyntaxError: unterminated string literal (detected at line 50)
Tariq@MacBook-Pro-3 Image-Caption-Generator %
```

Before:

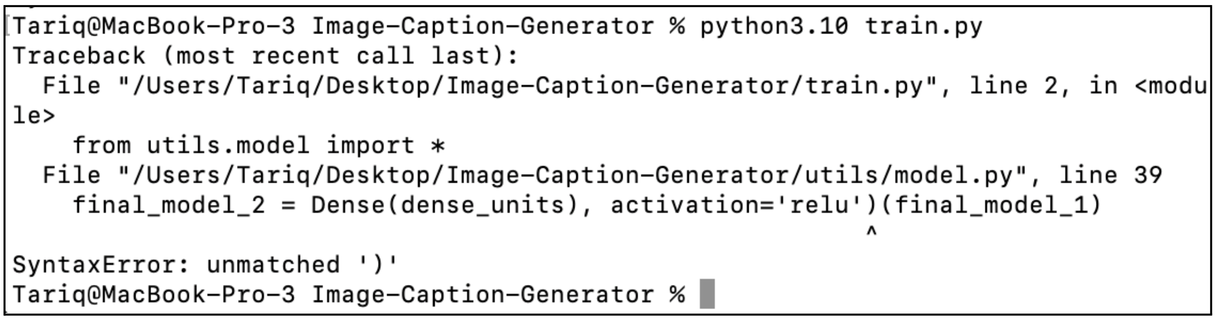
```
model_save_path = 'model_data/model_inceptio
```

After:

(I used the config file to determine the parameters used for the save path instead of doing it myself)

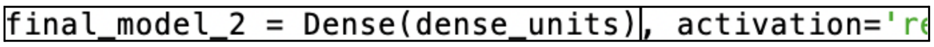
```
model_save_path = config['model_data_path']+"model_in
```

Error 2:



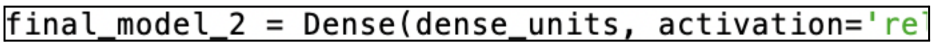
```
Tariq@MacBook-Pro-3 Image-Caption-Generator % python3.10 train.py
Traceback (most recent call last):
  File "/Users/Tariq/Desktop/Image-Caption-Generator/train.py", line 2, in <modu
le>
    from utils.model import *
  File "/Users/Tariq/Desktop/Image-Caption-Generator/utils/model.py", line 39
    final_model_2 = Dense(dense_units), activation='relu')(final_model_1)
                                                         ^
SyntaxError: unmatched ')'
Tariq@MacBook-Pro-3 Image-Caption-Generator %
```

Before:



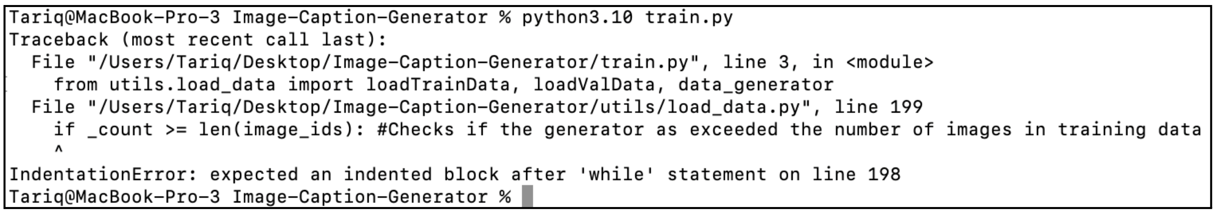
```
final_model_2 = Dense(dense_units), activation='r
```

After:



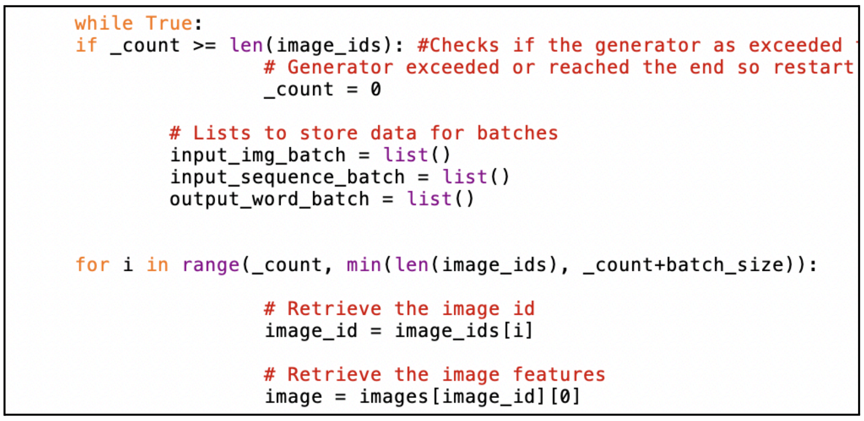
```
final_model_2 = Dense(dense_units, activation='re
```

Error 3:



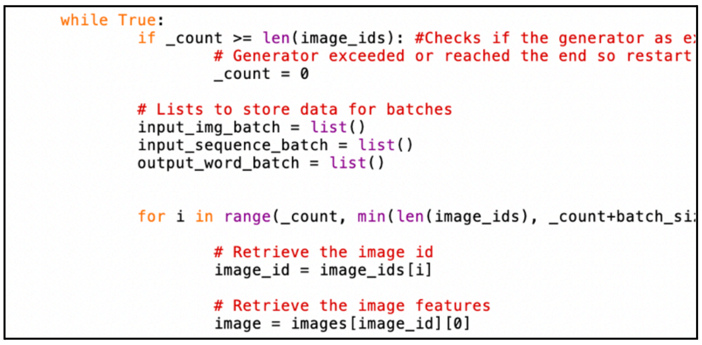
```
Tariq@MacBook-Pro-3 Image-Caption-Generator % python3.10 train.py
Traceback (most recent call last):
  File "/Users/Tariq/Desktop/Image-Caption-Generator/train.py", line 3, in <module>
    from utils.load_data import loadTrainData, loadValData, data_generator
  File "/Users/Tariq/Desktop/Image-Caption-Generator/utils/load_data.py", line 199
    if _count >= len(image_ids): #Checks if the generator as exceeded the number of images in training data
    ^
IndentationError: expected an indented block after 'while' statement on line 198
Tariq@MacBook-Pro-3 Image-Caption-Generator %
```

Before:



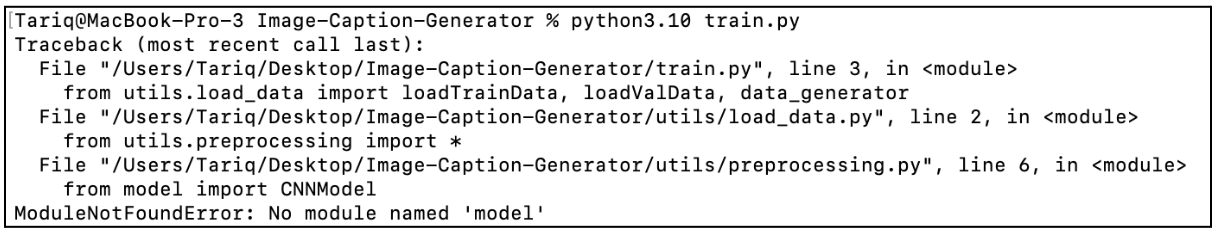
```
    while True:
    if _count >= len(image_ids): #Checks if the generator as exceeded
                    # Generator exceeded or reached the end so restart
                    _count = 0

            # Lists to store data for batches
            input_img_batch = list()
            input_sequence_batch = list()
            output_word_batch = list()


    for i in range(_count, min(len(image_ids), _count+batch_size)):

                    # Retrieve the image id
                    image_id = image_ids[i]

                    # Retrieve the image features
                    image = images[image_id][0]
```

After:



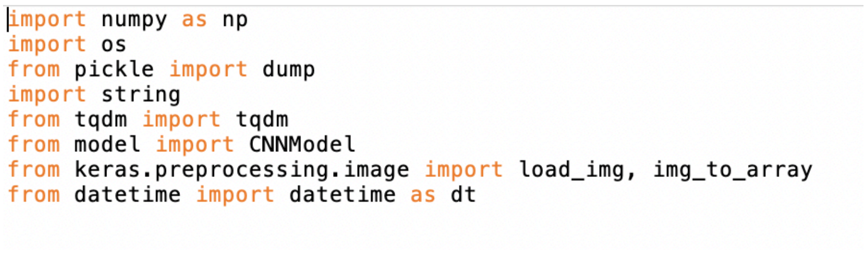
```
    while True:
            if _count >= len(image_ids): #Checks if the generator as e
                    # Generator exceeded or reached the end so restart
                    _count = 0

            # Lists to store data for batches
            input_img_batch = list()
            input_sequence_batch = list()
            output_word_batch = list()


            for i in range(_count, min(len(image_ids), _count+batch_si

                    # Retrieve the image id
                    image_id = image_ids[i]

                    # Retrieve the image features
                    image = images[image_id][0]
```

Error 4:



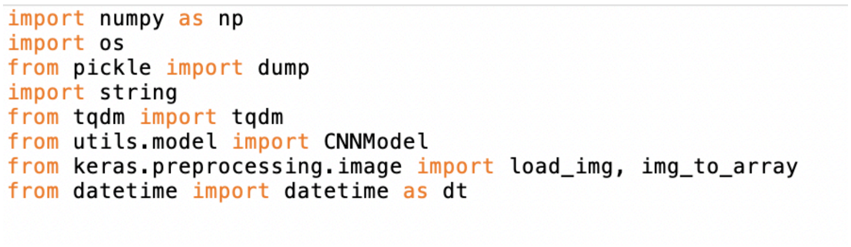
```
Tariq@MacBook-Pro-3 Image-Caption-Generator % python3.10 train.py
Traceback (most recent call last):
  File "/Users/Tariq/Desktop/Image-Caption-Generator/train.py", line 3, in <module>
    from utils.load_data import loadTrainData, loadValData, data_generator
  File "/Users/Tariq/Desktop/Image-Caption-Generator/utils/load_data.py", line 2, in <module>
    from utils.preprocessing import *
  File "/Users/Tariq/Desktop/Image-Caption-Generator/utils/preprocessing.py", line 6, in <module>
    from model import CNNModel
ModuleNotFoundError: No module named 'model'
```

Before:



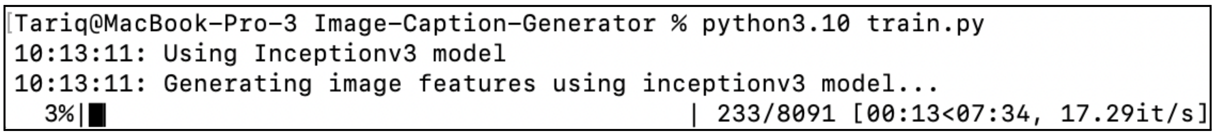
```
import numpy as np
import os
from pickle import dump
import string
from tqdm import tqdm
from model import CNNModel
from keras.preprocessing.image import load_img, img_to_array
from datetime import datetime as dt
```

After:

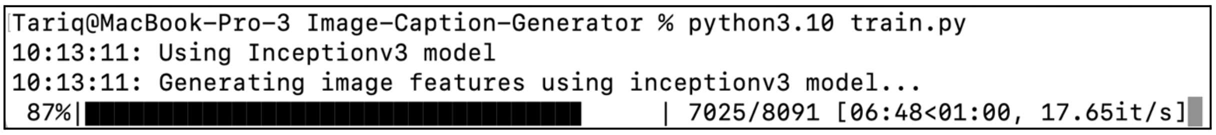


```
import numpy as np
import os
from pickle import dump
import string
from tqdm import tqdm
from utils.model import CNNModel
from keras.preprocessing.image import load_img, img_to_array
from datetime import datetime as dt
```

Running:

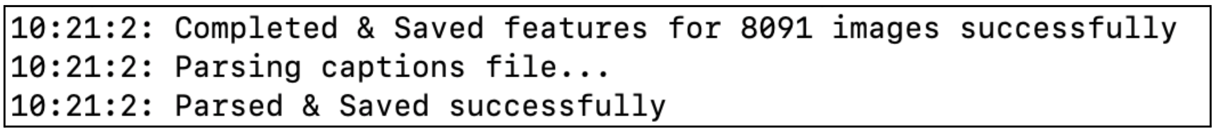


```
[Tariq@MacBook-Pro-3 Image-Caption-Generator % python3.10 train.py
10:13:11: Using Inceptionv3 model
10:13:11: Generating image features using inceptionv3 model...
  3%|█                                    | 233/8091 [00:13<07:34, 17.29it/s]
```



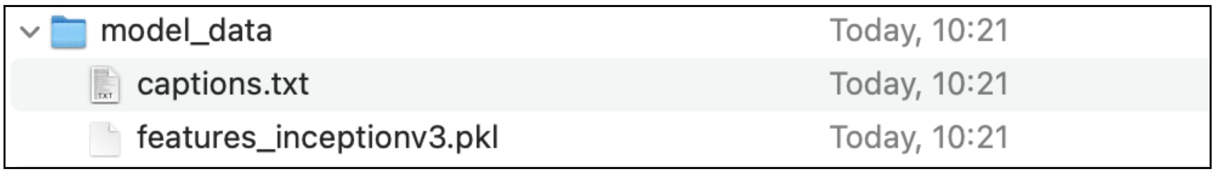
```
[Tariq@MacBook-Pro-3 Image-Caption-Generator % python3.10 train.py
10:13:11: Using Inceptionv3 model
10:13:11: Generating image features using inceptionv3 model...
 87%|██████████████████████████████████      | 7025/8091 [06:48<01:00, 17.65it/s]
```

The model has begun generating the image features from the list of images inputted to it. This process (for ~8000 images) takes around 7.5 minutes at an average of 17 images per second. This will vary for each device and will depend on the amount of memory, CPU, and GPU of a system.



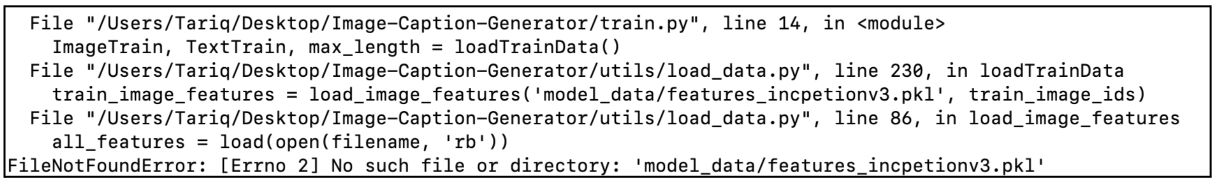
```
10:21:2: Completed & Saved features for 8091 images successfully
10:21:2: Parsing captions file...
10:21:2: Parsed & Saved successfully
```

The program has successfully saved the captions and features files from the images and has saved them in /model_data/



```
model_data                      Today, 10:21
    captions.txt                Today, 10:21
    features_inceptionv3.pkl    Today, 10:21
```

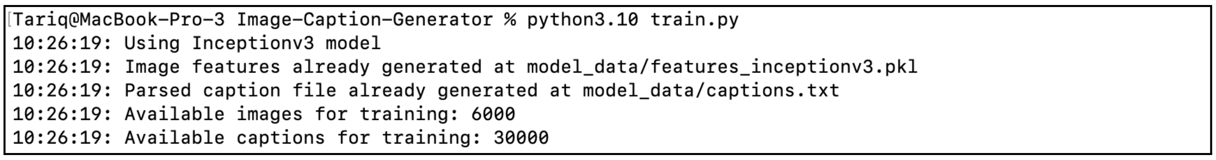
Another error has occurred where the script cannot find the .pkl file, but that is because it is looking for *features_incpetion.pkl*



```
  File "/Users/Tariq/Desktop/Image-Caption-Generator/train.py", line 14, in <module>
    ImageTrain, TextTrain, max_length = loadTrainData()
  File "/Users/Tariq/Desktop/Image-Caption-Generator/utils/load_data.py", line 230, in loadTrainData
    train_image_features = load_image_features('model_data/features_incpetionv3.pkl', train_image_ids)
  File "/Users/Tariq/Desktop/Image-Caption-Generator/utils/load_data.py", line 86, in load_image_features
    all_features = load(open(filename, 'rb'))
FileNotFoundError: [Errno 2] No such file or directory: 'model_data/features_incpetionv3.pkl'
```

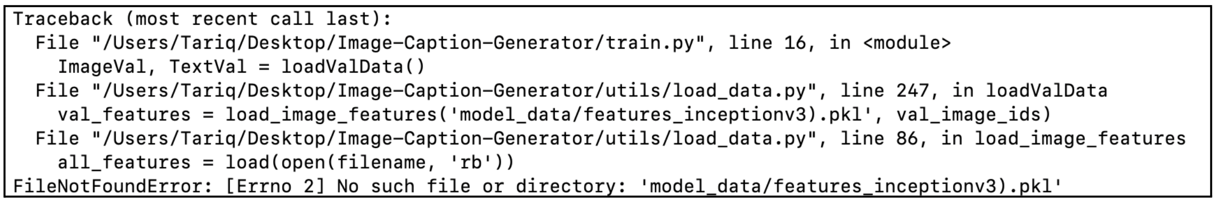
This is spelt wrong and should be *features_inception.pkl*

I will change this and re-run the script.



```
Tariq@MacBook-Pro-3 Image-Caption-Generator % python3.10 train.py
10:26:19: Using Inceptionv3 model
10:26:19: Image features already generated at model_data/features_inceptionv3.pkl
10:26:19: Parsed caption file already generated at model_data/captions.txt
10:26:19: Available images for training: 6000
10:26:19: Available captions for training: 30000
```

This time, the program detected 6000 images for training and 30000 captions for them, however, another error occurred, similar to the last, due to a spelling mistake in the code:

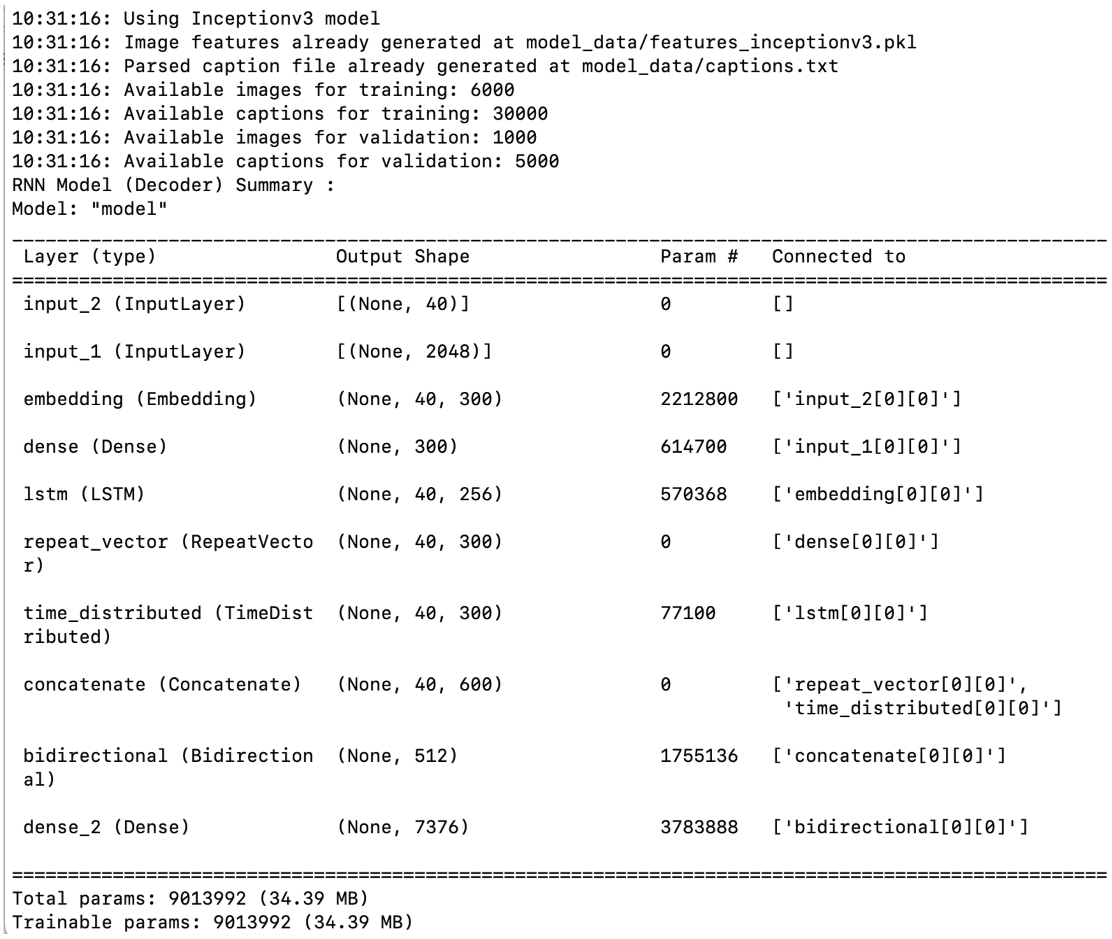


```
Traceback (most recent call last):
  File "/Users/Tariq/Desktop/Image-Caption-Generator/train.py", line 16, in <module>
    ImageVal, TextVal = loadValData()
  File "/Users/Tariq/Desktop/Image-Caption-Generator/utils/load_data.py", line 247, in loadValData
    val_features = load_image_features('model_data/features_inceptionv3).pkl', val_image_ids)
  File "/Users/Tariq/Desktop/Image-Caption-Generator/utils/load_data.py", line 86, in load_image_features
    all_features = load(open(filename, 'rb'))
FileNotFoundError: [Errno 2] No such file or directory: 'model_data/features_inceptionv3).pkl'
```

I have fixed this and re-run the code again.

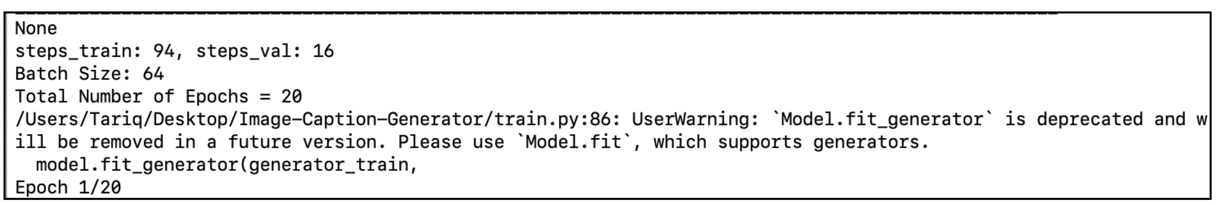
Below is the RNN model summary.

It consists of 2 input layers, 1 embedding layer, 1 dense layer, 1 LSTM layer, 1 repeat vector, 1 time distributed layer, 1 concatenate layer, 1 bidirectional layer and another final dense layer.

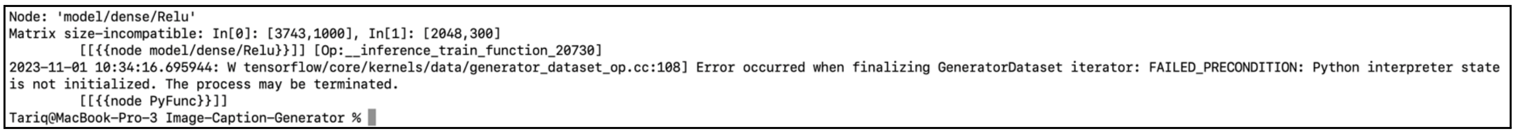


```
10:31:16: Using Inceptionv3 model
10:31:16: Image features already generated at model_data/features_inceptionv3.pkl
10:31:16: Parsed caption file already generated at model_data/captions.txt
10:31:16: Available images for training: 6000
10:31:16: Available captions for training: 30000
10:31:16: Available images for validation: 1000
10:31:16: Available captions for validation: 5000
RNN Model (Decoder) Summary :
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_2 (InputLayer)        [(None, 40)]                 0         []

 input_1 (InputLayer)        [(None, 2048)]               0         []

 embedding (Embedding)       (None, 40, 300)              2212800   ['input_2[0][0]']

 dense (Dense)               (None, 300)                  614700    ['input_1[0][0]']

 lstm (LSTM)                 (None, 40, 256)              570368    ['embedding[0][0]']

 repeat_vector (RepeatVecto  (None, 40, 300)              0         ['dense[0][0]']
 r)

 time_distributed (TimeDist  (None, 40, 300)              77100     ['lstm[0][0]']
 ributed)

 concatenate (Concatenate)   (None, 40, 600)              0         ['repeat_vector[0][0]',
                                                                     'time_distributed[0][0]']

 bidirectional (Bidirection  (None, 512)                  1755136   ['concatenate[0][0]']
 al)

 dense_2 (Dense)             (None, 7376)                 3783888   ['bidirectional[0][0]']

==================================================================================================
Total params: 9013992 (34.39 MB)
Trainable params: 9013992 (34.39 MB)
```

The program starts to iterate through and train the model but another error occurs.



```
None
steps_train: 94, steps_val: 16
Batch Size: 64
Total Number of Epochs = 20
/Users/Tariq/Desktop/Image-Caption-Generator/train.py:86: UserWarning: `Model.fit_generator` is deprecated and w
ill be removed in a future version. Please use `Model.fit`, which supports generators.
  model.fit_generator(generator_train,
Epoch 1/20
```



```
Node: 'model/dense/Relu'
Matrix size-incompatible: In[0]: [3743,1000], In[1]: [2048,300]
         [[{{node model/dense/Relu}}]] [Op:__inference_train_function_20730]
2023-11-01 10:34:16.695944: W tensorflow/core/kernels/data/generator_dataset_op.cc:108] Error occurred when finalizing GeneratorDataset iterator: FAILED_PRECONDITION: Python interpreter state
is not initialized. The process may be terminated.
         [[{{node PyFunc}}]]
Tariq@MacBook-Pro-3 Image-Caption-Generator %
```

This error suggests that there is a problem with one of the layers in the neural network model. In this case, it seems to be a dense layer with a ReLU activation. The specific error is "Matrix size-incompatible: In[0]: [3743,1000], In[1]: [2048,300]." This error indicates that there is a mismatch in the dimensions of matrices passed to the 'model/dense/Relu' operation. The error may also be associated with a Python interpreter state issue, which could potentially terminate the process.

To fix this, I will try re-running train.py from the Python interpreter and if that doesn't work, I will attempt to use a different model architecture.

I updated my code to attempt to debug any issues with the shapes for the tensors, and these were my results:

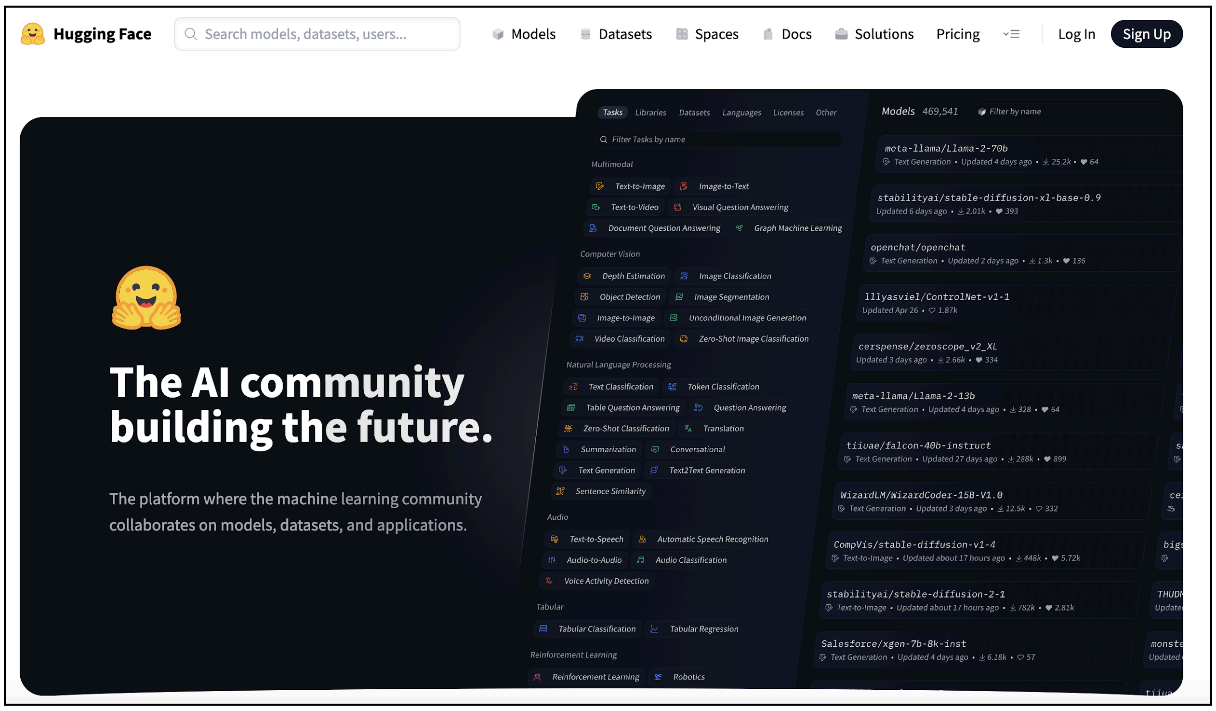
Image Input Shape: (None, 2048)
Image Model 1 Shape: (None, 300)
Caption Model Shape: (None, 40, 300)
Final Model 1 Shape after Concatenate: (None, 40, 600)
Final Model 2 Shape: (None, 512)

Based on the debugged outputs, the model construction seems fine. The shapes are as expected. These shapes are consistent with the design of the RNN model, so there doesn't seem to be a problem in constructing the model architecture, I even attempted a different model architecture, but I came across the same error.

After attempting various solutions and days of unsuccessful debugging, such as migrating the project to a Windows-based operating system, then a Linux-based operating system and downgrading to Python 3.6.7m which the guide was written in, I still could not find any solution to the error that I was encountering, which seemed to be a problem with the model architecture. As a result, I have decided to move forward with a different approach by utilising a pre-trained model, although I hope to be able to train my own image captioning model in the future, should the time constraints of this project allow.

I will be continuing this project by utilising a solution called Huggingface.co Huggingface is an open-source platform for machine learning and data research. It serves as a hub for AI specialists and fans, similar to GitHub but for AI.

Huggingface.co



The AI community building the future.

The platform where the machine learning community collaborates on models, datasets, and applications.

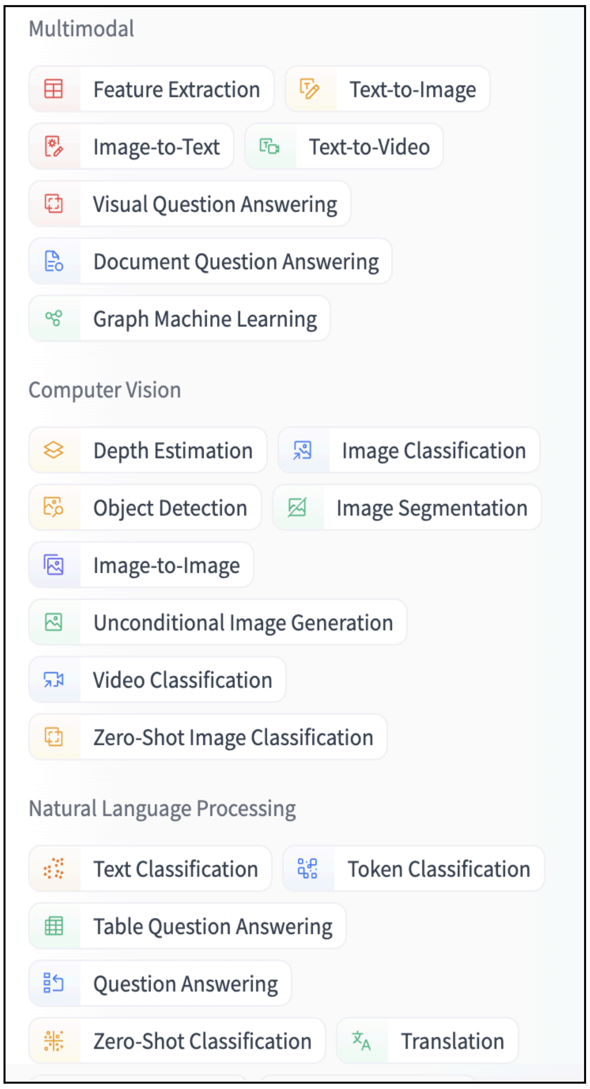
Huggingface is itself a company, but it is primarily made up of all the people contributing their AI models to it, these contributions help create a community of Natural Language Processing (NLP) models

Huggingface also makes use of a library called ‘transformers’ which allows anybody to have direct, offline use of their models, which is what I will be using for my project.

Essentially, Huggingface is a place where people are able to share their own models and projects, while also viewing other people’s.

The models that are featured on Huggingface are vast in number and are categorized by their type and functions.

Huggingface model browse tab


Multimodal

- Feature Extraction
- Text-to-Image
- Image-to-Text
- Text-to-Video
- Visual Question Answering
- Document Question Answering
- Graph Machine Learning

Computer Vision

- Depth Estimation
- Image Classification
- Object Detection
- Image Segmentation
- Image-to-Image
- Unconditional Image Generation
- Video Classification
- Zero-Shot Image Classification

Natural Language Processing

- Text Classification
- Token Classification
- Table Question Answering
- Question Answering
- Zero-Shot Classification
- Translation

Inside each category, there are more subcategories for more specialised models designed to undertake specific tasks.

For this project, I will be focusing on the image-to-text language model, from the multimodal category.

A multimodal AI model is an AI paradigm, in which various data types (image, text, speech, numerical data) are combined with multiple intelligence processing algorithms to achieve higher performances. Multimodal AI often outperforms single-modal AI in many real-world problems.

An image captioning AI model is classified as a multimodal model because it involves processing and understanding information from multiple modalities or sources of data. In the context of image captioning, the two main modalities are:

Image Modality: This refers to the visual information contained in an image. The AI model needs to analyse and comprehend the visual content of the image to generate a meaningful caption.

Text Modality: This refers to the linguistic or textual information. The model generates a caption, which is a textual description, based on its understanding of the visual content.

A multimodal model combines these different modalities to perform a task, such as image captioning. It typically consists of two main components:

Vision Model: This component processes the visual information from the image. Convolutional Neural Networks (CNNs) are commonly used for image processing tasks, as they are adept at capturing hierarchical features in visual data.

Language Model: This component processes the textual information. Recurrent Neural Networks (RNNs) or Transformer-based models, such as GPT (Generative Pre-trained Transformer), are often used for language understanding and generation tasks.

Specifically, I will be using the following model for my project:

nlpconnect/vit-gpt2-image-captioning

Image-to-Text • Updated Feb 27 • 2.48M • 602

This model makes use of gpt-2, a large language model (LLM) used for text generation. After extracting the image features, the gpt-2 model will caption the image and return a string in the console.

In order to implement this into my project, I will be creating a Flask server that will manage the processing of the image and within Unity, I will create a C# script that, when pressing a button, will take a screenshot of the screen and send the image to the flask server for caption processing, while waiting for a response.

Below is the Python script for image processing:

imageCaption.py

```
from flask import Flask, request, jsonify
from transformers import VisionEncoderDecoderModel, ViTImageProcessor, AutoTokenizer
import torch
from PIL import Image
import io
import base64

# Load pre-trained models and set up processing tools
model = VisionEncoderDecoderModel.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
feature_extractor = ViTImageProcessor.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
tokenizer = AutoTokenizer.from_pretrained("nlpconnect/vit-gpt2-image-captioning")


# Check if GPU is available and move the model to the appropriate device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)


# Set parameters for text generation
max_length = 16
num_beams = 4
gen_kwargs = {"max_length": max_length, "num_beams": num_beams}
```

The script starts by importing the necessary libraries and modules, such as Flask, transformers (from Huggingface), PyTorch, and Pillow. After the imports, the pre-trained model is imported. For image captioning, you need a model to caption the images, a feature extractor, and a tokenizer, all of which were included in the Huggingface model that I am using.

The script then moves the model to the system GPU if available. Using a GPU for deep learning tasks, including training and inference of machine learning models, offers several advantages over using a CPU (Central Processing Unit) such as:

- Parallel processing
    - GPUs are designed to handle parallel processing efficiently. Deep learning tasks, such as those involving neural networks, often involve matrix operations that can be parallelized.
- High memory bandwidth
    - Deep learning models often require large amounts of data to be processed quickly. GPUs have high memory bandwidth, allowing them to efficiently transfer and process large amounts of data in parallel.
- Availability of GPU-Accelerated Libraries
    - Many deep learning frameworks, including TensorFlow and PyTorch, provide GPU-accelerated implementations of key operations. This means that when running these frameworks on a GPU, the computations are offloaded to the GPU, resulting in faster execution.
- Real-time Inference
    - For applications like image captioning, where real-time responses are desired, the speed advantage of GPUs becomes crucial. GPUs can handle inference tasks quickly, making them suitable for applications that require low latency.

After setting the model to the GPU or CPU, the parameters for text generation are set, this includes the maximum length of the caption and the number of beams (beams define how far the model should look into the potential futures/next word combinations.)

imageCaption.py

```python
# Create a Flask web application
app = Flask(__name__)

# Define a route for handling file uploads via POST request
@app.route('/upload', methods=['POST'])
def upload():
    try:
        # Get the screenshot from the request
        screenshot_data = request.files['screenshot'].read()

        # Process the screenshot and get the caption
        caption = predict_caption(screenshot_data)

        # Return the caption as JSON response
        return jsonify(caption)
    except Exception as e:
        # Return an error message if an exception occurs
        return jsonify({'error': str(e)})
```

The script then creates a Flask web application then defines a route /upload that handles POST requests. When an image is uploaded, it reads the image data from the request, calls the predict_caption function to generate a caption, and returns the caption as a JSON response. If an exception occurs, it returns an error message.

imageCaption.py

```
# Define a function for predicting captions from the input screenshot
def predict_caption(screenshot_data):
    try:

        # Process the screenshot
        image = Image.open(io.BytesIO(screenshot_data)).convert("RGB")
        images = [image]

        # Extract pixel values and move to the specified device (CPU/GPU)
        pixel_values = feature_extractor(images=images, return_tensors="pt").pixel_values
        pixel_values = pixel_values.to(device)

        # Generate captions using the loaded model
        output_ids = model.generate(pixel_values, **gen_kwargs)

        # Decode the generated output and remove special tokens
        preds = tokenizer.batch_decode(output_ids, skip_special_tokens=True)
        preds = [pred.strip() for pred in preds]
        print("final caption: ", preds)
        return preds

    except Exception as e:
        # Print an error message if an exception occurs during prediction
        print("Error predicting caption:", e)
        return "Error predicting caption"

# Start the Flask application if this script is executed directly
if __name__ == '__main__':
    app.run(debug=True)
```

The predict_caption function takes the screenshot data as input, processes the image, generates captions using the pre-trained model, and returns the predicted captions. If an exception occurs during prediction, it prints an error message and returns an error string.

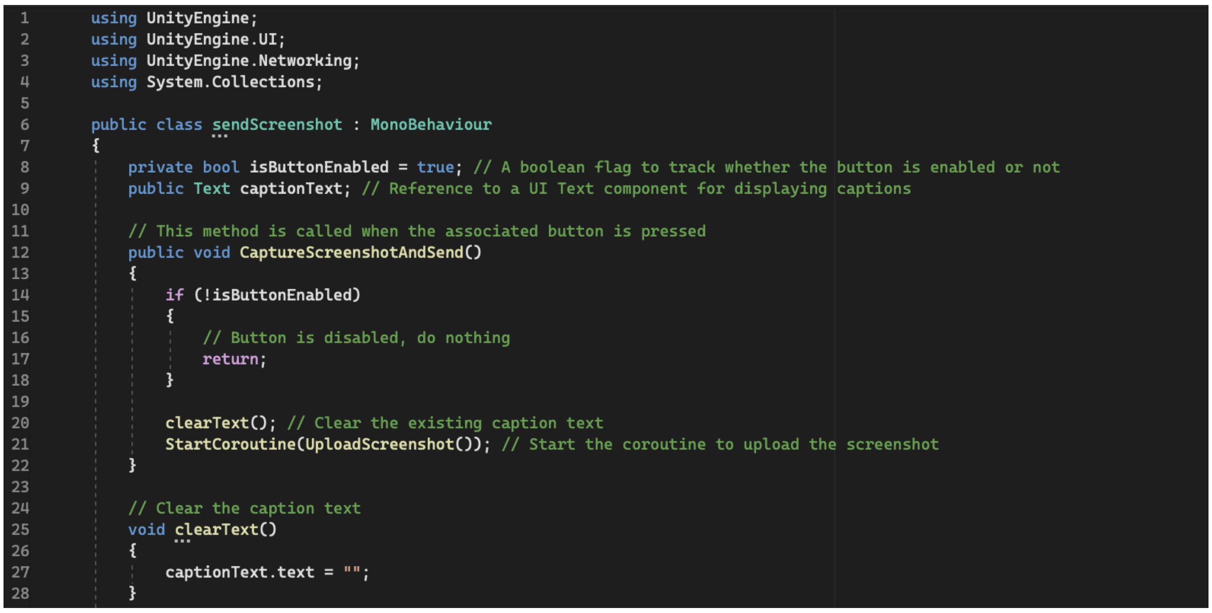
For now, the server will run on the default host and port (127.0.0.1:5000). The debug=True parameter enables debugging features, making it easier to identify and fix issues during development.

For the C# side of the image captioning, some preconditions must be met such as:

- When the button is pressed, it should be disabled until a response is returned. This is because I do not want the user sending many responses at once and overwhelming the server.
- When the screenshot is taken, it should disable any GUI temporarily and only capture the raw camera view, with no bounding boxes or buttons in view. When the screenshot is taken, the GUI can reappear.
- There should be a check that the device is connected to the internet. If it is not connected, the image captioning button should be disabled as there would be no way to send a request to the Flask server and this could cause some problematic errors for the user.

Below is the script that accomplishes this

sendScreenshot.cs

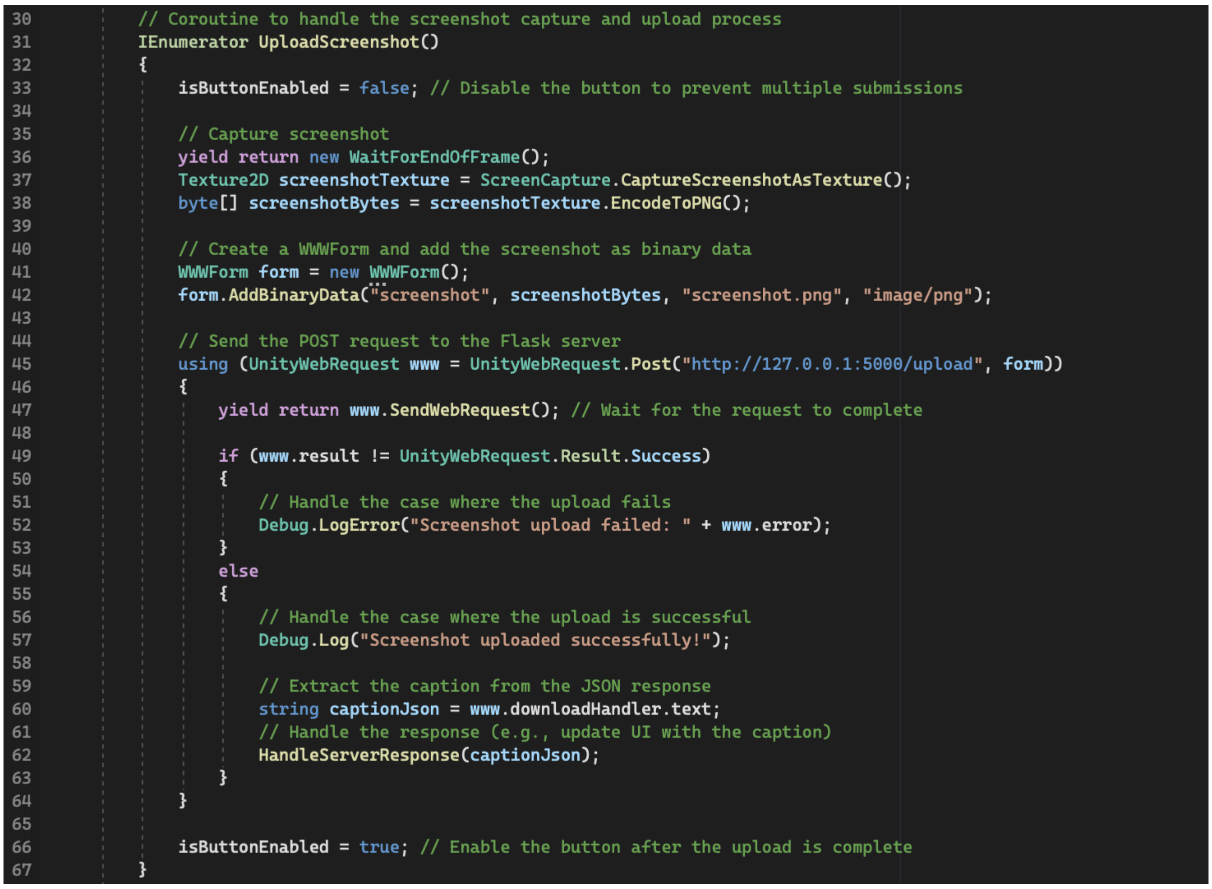


```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Networking;
using System.Collections;

public class sendScreenshot : MonoBehaviour
{
    private bool isButtonEnabled = true; // A boolean flag to track whether the button is enabled or not
    public Text captionText; // Reference to a UI Text component for displaying captions

    // This method is called when the associated button is pressed
    public void CaptureScreenshotAndSend()
    {
        if (!isButtonEnabled)
        {
            // Button is disabled, do nothing
            return;
        }

        clearText(); // Clear the existing caption text
        StartCoroutine(UploadScreenshot()); // Start the coroutine to upload the screenshot
    }

    // Clear the caption text
    void clearText()
    {
        captionText.text = "";
    }
```

A private Boolean ‘isButtonEnabled’ is set, this tracks whether the button is enabled. A public Text variable called ‘captionText’ is also initialized. This is in charge of updating and displaying the text that is shown on the UI.

‘CaptureScreenshotAndSend’ is a public method that is called when the button is pressed. It checks if the button is enabled before proceeding. If the button is enabled, it clears the text and starts the coroutine UploadScreenshot().

sendScreenshot.cs



```
    // Coroutine to handle the screenshot capture and upload process
    IEnumerator UploadScreenshot()
    {
        isButtonEnabled = false; // Disable the button to prevent multiple submissions

        // Capture screenshot
        yield return new WaitForEndOfFrame();
        Texture2D screenshotTexture = ScreenCapture.CaptureScreenshotAsTexture();
        byte[] screenshotBytes = screenshotTexture.EncodeToPNG();

        // Create a WWWForm and add the screenshot as binary data
        WWWForm form = new WWWForm();
        form.AddBinaryData("screenshot", screenshotBytes, "screenshot.png", "image/png");

        // Send the POST request to the Flask server
        using (UnityWebRequest www = UnityWebRequest.Post("http://127.0.0.1:5000/upload", form))
        {
            yield return www.SendWebRequest(); // Wait for the request to complete

            if (www.result != UnityWebRequest.Result.Success)
            {
                // Handle the case where the upload fails
                Debug.LogError("Screenshot upload failed: " + www.error);
            }
            else
            {
                // Handle the case where the upload is successful
                Debug.Log("Screenshot uploaded successfully!");

                // Extract the caption from the JSON response
                string captionJson = www.downloadHandler.text;
                // Handle the response (e.g., update UI with the caption)
                HandleServerResponse(captionJson);
            }
        }

        isButtonEnabled = true; // Enable the button after the upload is complete
    }
```

‘UploadScreenshot’ is a coroutine. It sets isButtonEnabled to false to prevent multiple submissions. ‘yield return new WaitForEndOfFrame()’ waits until the end of the current frame. ‘CaptureScreenshotAsTexture()’ captures the screenshot and converts it to a texture. ‘EncodeToPNG()’ converts the texture to PNG format and stores it in a byte array.

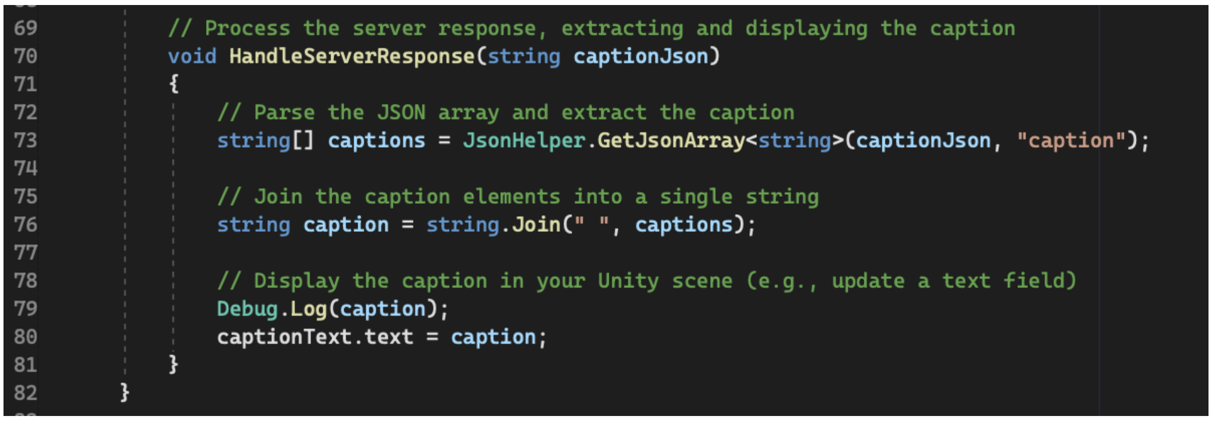
A ‘WWWForm’ is created to prepare the data for the HTTP request. The screenshot data is added to the form as binary data with a field name "screenshot", a file name "screenshot.png", and the MIME type "image/png".

A ‘UnityWebRequest’ is created for a POST request to the specified URL ("http://127.0.0.1:5000/upload") with the prepared form data. ‘www.SendWebRequest()’ sends the request and waits for it to complete.

After the web request is created, the function checks if the request was successful. If not, it logs an error. If successful, it logs success, extracts the caption JSON from the response, and calls ‘HandleServerResponse()’.

It finally re-enables the button after the upload process is complete.

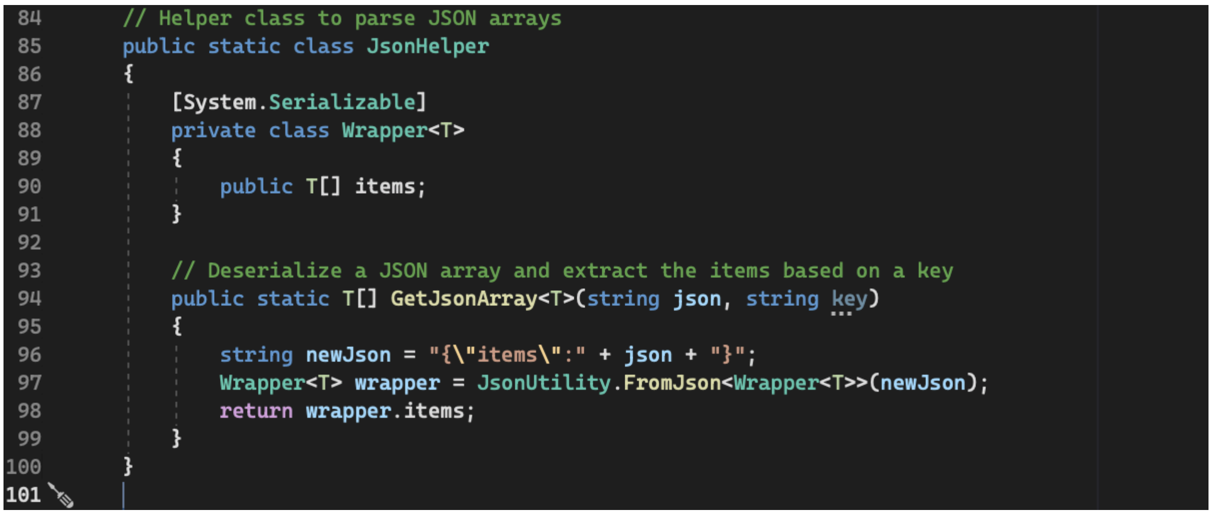
sendScreenshot.cs



```
        // Process the server response, extracting and displaying the caption
        void HandleServerResponse(string captionJson)
        {
            // Parse the JSON array and extract the caption
            string[] captions = JsonHelper.GetJsonArray<string>(captionJson, "caption");

            // Join the caption elements into a single string
            string caption = string.Join(" ", captions);

            // Display the caption in your Unity scene (e.g., update a text field)
            Debug.Log(caption);
            captionText.text = caption;
        }
    }
```

The ‘HandleServerResponse()’ function processes the server's JSON response.
It extracts the caption from the JSON using the ‘JsonHelper.GetJsonArray’ method.
The caption elements are joined into a single string and then logged and displayed in the UI.

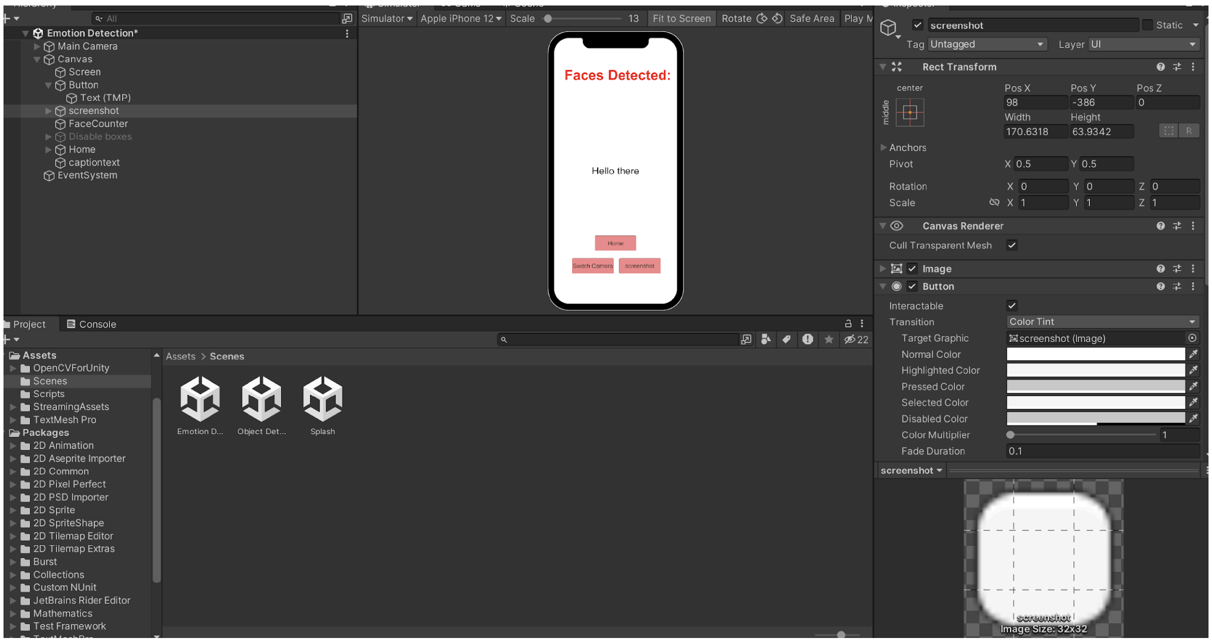
sendScreenshot.cs



```
    // Helper class to parse JSON arrays
    public static class JsonHelper
    {
        [System.Serializable]
        private class Wrapper<T>
        {
            public T[] items;
        }

        // Deserialize a JSON array and extract the items based on a key
        public static T[] GetJsonArray<T>(string json, string key)
        {
            string newJson = "{\"items\":" + json + "}";
            Wrapper<T> wrapper = JsonUtility.FromJson<Wrapper<T>>(newJson);
            return wrapper.items;
        }
    }
```

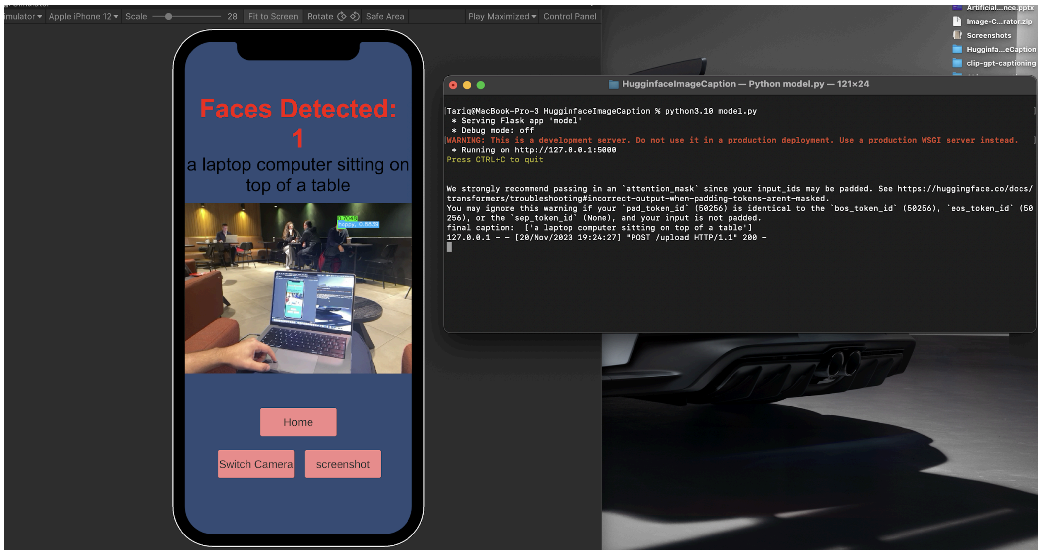
Finally, the class ‘JsonHelper’ is a utility class to parse JSON arrays. It uses generics to allow parsing arrays of different types. The Wrapper<T> class is a helper for deserializing JSON arrays. The ‘GetJsonArray’ method takes the JSON string and a key, wraps it in a new JSON structure, and then deserializes it using ‘JsonUtility’.

I have set up a textbox in the middle of the screen and a ‘screenshot’ button that calls the ‘CaptureScreenshotAndSend()’ function in the *sendScreenshot.cs* script. This will become disabled while waiting for a response and return the caption in the textbox set in the public variable.



```
screenshot                     Static
Tag Untagged   Layer UI
Rect Transform
center   Pos X 98   Pos Y -386   Pos Z 0
middle   Width 170.6318   Height 63.9342
Anchors
Pivot      X 0.5  Y 0.5
Rotation   X 0  Y 0  Z 0
Scale      X 1  Y 1  Z 1
Canvas Renderer
Cull Transparent Mesh ✓
Image
Button
Interactable ✓
Transition        Color Tint
Target Graphic    screenshot (Image)
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier  1
Fade Duration     0.1
screenshot
Image Size: 32x32
```

I ran the code and tested the functionality.



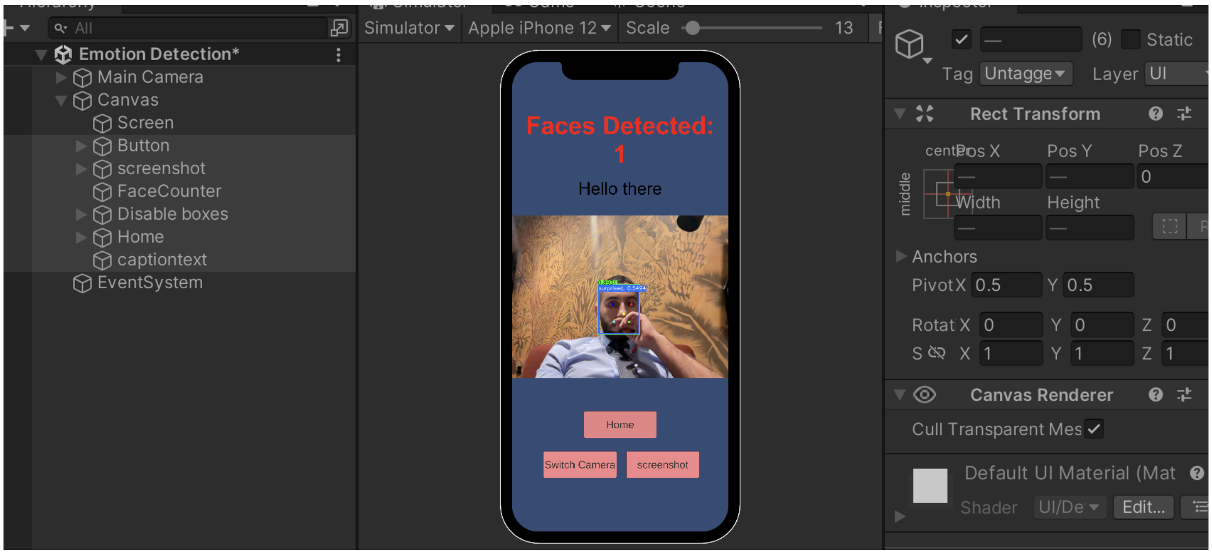
```
Tariq@MacBook-Pro-3 HugginfaceImageCaption % python3.10 model.py
 * Serving Flask app 'model'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit


We strongly recommend passing in an `attention_mask` since your input_ids may be padded. See https://huggingface.co/docs/
transformers/troubleshooting#incorrect-output-when-padding-tokens-arent-masked.
You may ignore this warning if your `pad_token_id` (50256) is identical to the `bos_token_id` (50256), `eos_token_id` (50
256), or the `sep_token_id` (None), and your input is not padded.
final caption:  ['a laptop computer sitting on top of a table']
127.0.0.1 - - [20/Nov/2023 19:24:27] "POST /upload HTTP/1.1" 200 -
```

The captioning works quite well and successfully sends a screenshot to the flask server for processing, however, unity will be sending a screenshot of the whole device screen, including the UI elements and text. This may have an effect on the accuracy of the generated caption and may confuse the AI model.

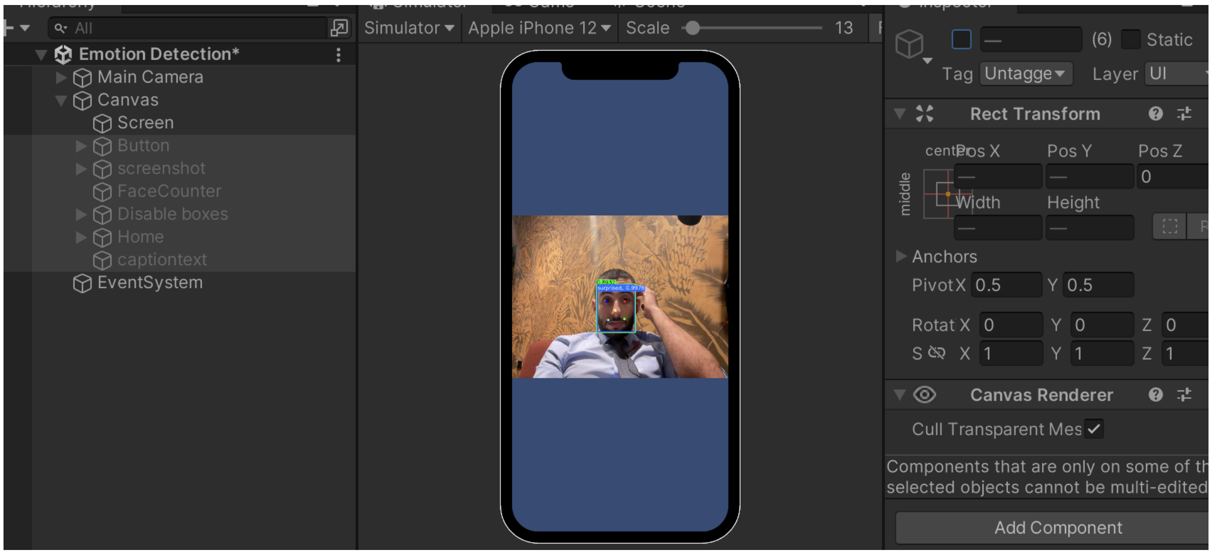
The best way to tackle this is by temporarily disabling UI elements while the screenshot is being taken.

One way to do this is to do this is by setting all the elements to enabled=False.

This is what it looks like when all the unecassary elements are active:

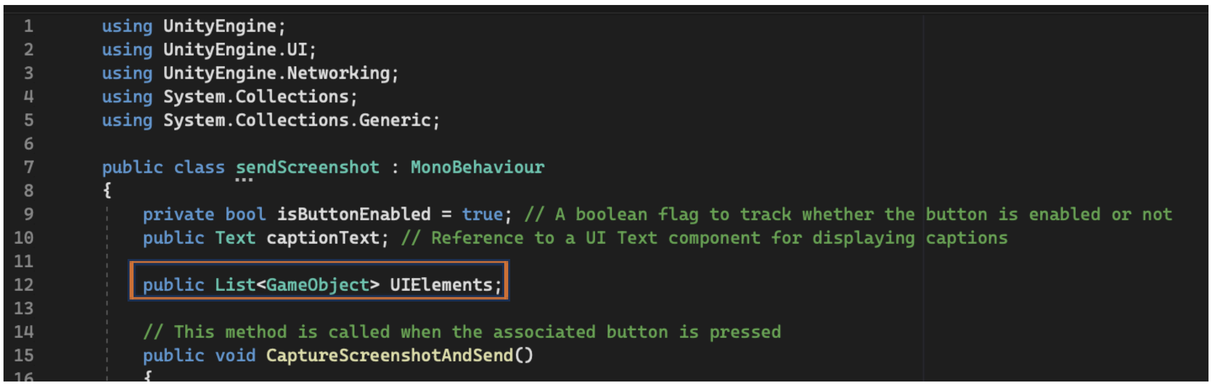


Here is what it looks like when we disable the UI elements (in the inspector)

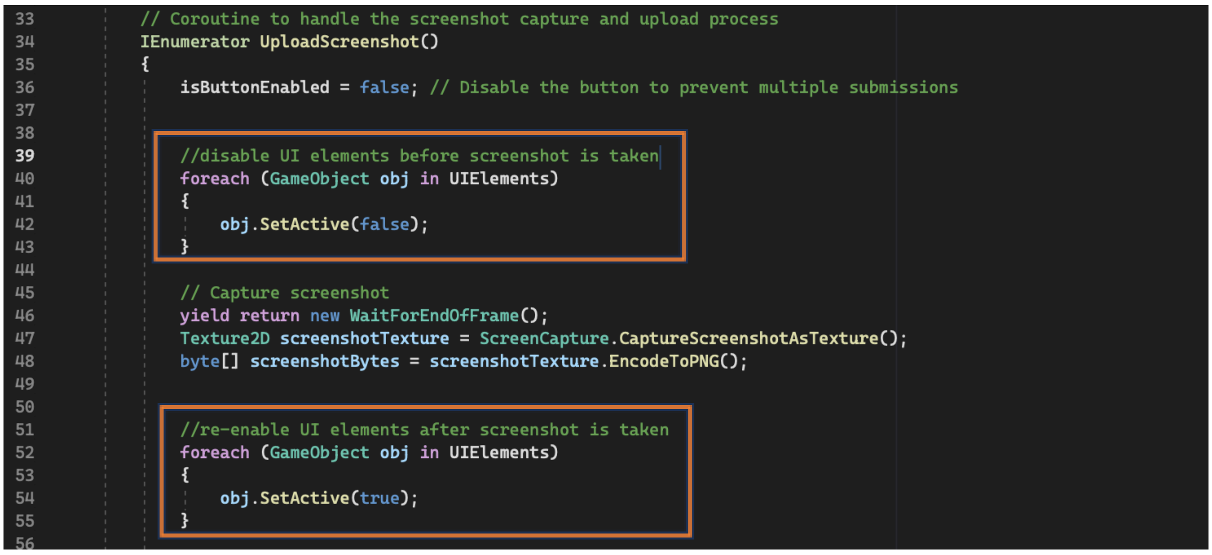


I will alter my *sendScreenshot.cs* script to include the logic needed to disable these UI elements.

sendScreenshot.cs



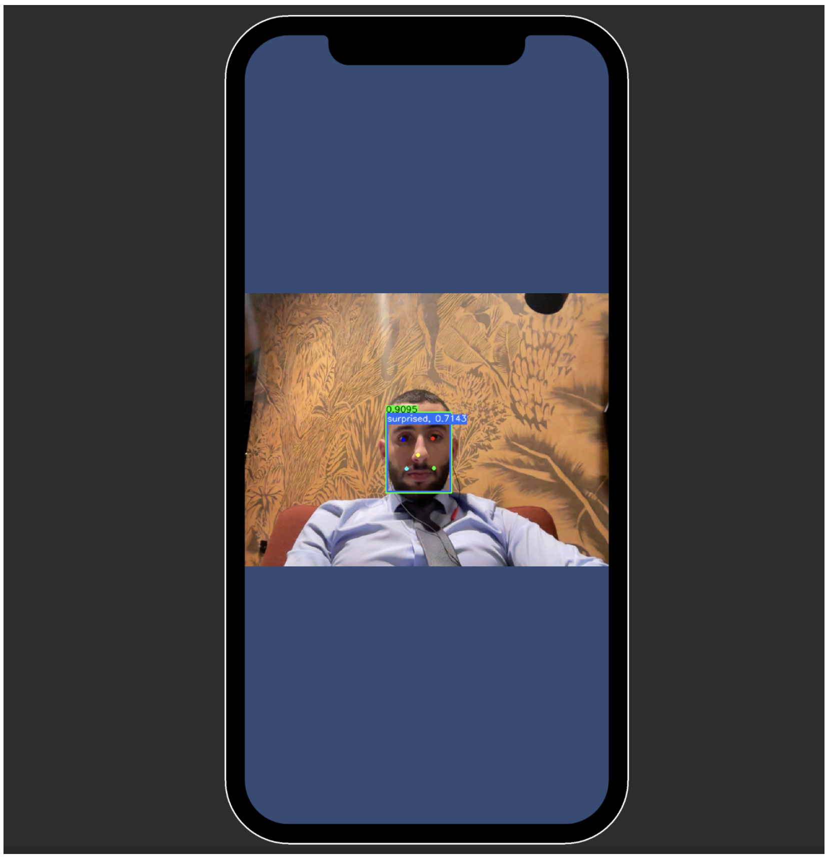
```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Networking;
using System.Collections;
using System.Collections.Generic;

public class sendScreenshot : MonoBehaviour
{
    private bool isButtonEnabled = true; // A boolean flag to track whether the button is enabled or not
    public Text captionText; // Reference to a UI Text component for displaying captions

    public List<GameObject> UIElements;

    // This method is called when the associated button is pressed
    public void CaptureScreenshotAndSend()
```



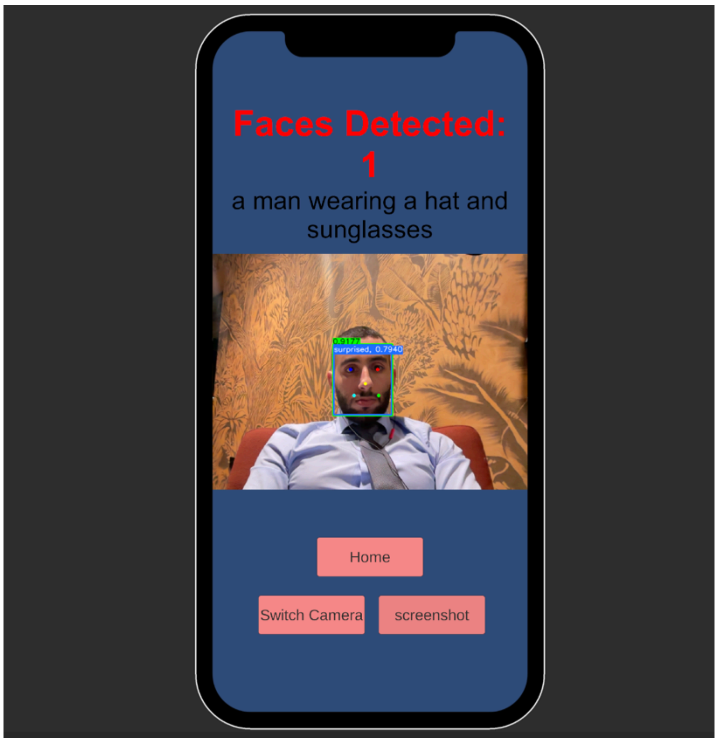
```
    // Coroutine to handle the screenshot capture and upload process
    IEnumerator UploadScreenshot()
    {
        isButtonEnabled = false; // Disable the button to prevent multiple submissions


        //disable UI elements before screenshot is taken
        foreach (GameObject obj in UIElements)
        {
            obj.SetActive(false);
        }

        // Capture screenshot
        yield return new WaitForEndOfFrame();
        Texture2D screenshotTexture = ScreenCapture.CaptureScreenshotAsTexture();
        byte[] screenshotBytes = screenshotTexture.EncodeToPNG();


        //re-enable UI elements after screenshot is taken
        foreach (GameObject obj in UIElements)
        {
            obj.SetActive(true);
        }
```

I created a list of GameObjects that can be disabled all together. I used a list as this would allow me to easily increase the number of items in the list and remove them, without having to create many different variables for each new UI element added. For example, if in later iterations I add a settings button or UI images, it will be much easier to add them to the list of GameObjects that will be disabled altogether. Once the screenshot is taken, the UI elements are re-enabled. This process happens usually within 1-2 frames and is almost unnoticeable but will have a massive impact on the image sent to the AI model as there are less things to process.

This is what the screenshots will look like now when captured:



Although this level of abstraction is great, the bounding box around the person's face is also a distracting feature. This is because the AI model only takes in a low-resolution image and therefore the box may blend and be mistaken for something unnecessary, such as a hat, seen below:

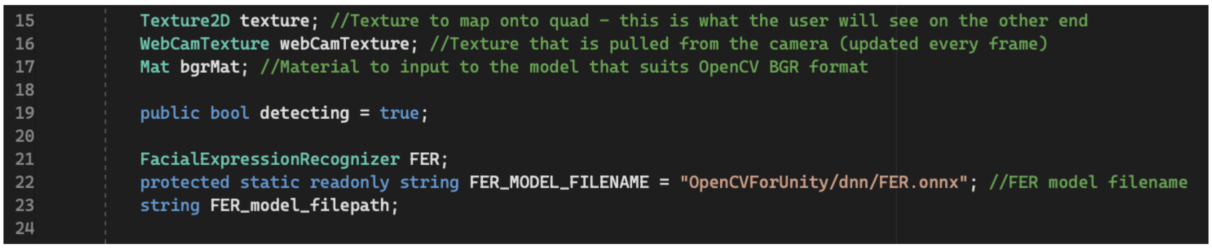


The caption that was detected with this image stated "A man wearing a hat and sunglasses". The bounding box from the face detection model is mistaken for a hat and sunglasses, this is a result of the thicker boxes above the head and the coloured spots on top of the eyes.

I will need to disable this while the screenshot is being taken so that the processing is more accurate when getting captions of people's faces.

The best way to disable the bounding boxes temporarily is to set a Boolean value before the update() function in the *EmotionDetectionScript.cs*, where before the inference of images begins, the function checks if a screenshot is being taken, and if so, it will stop detecting faces temporarily.

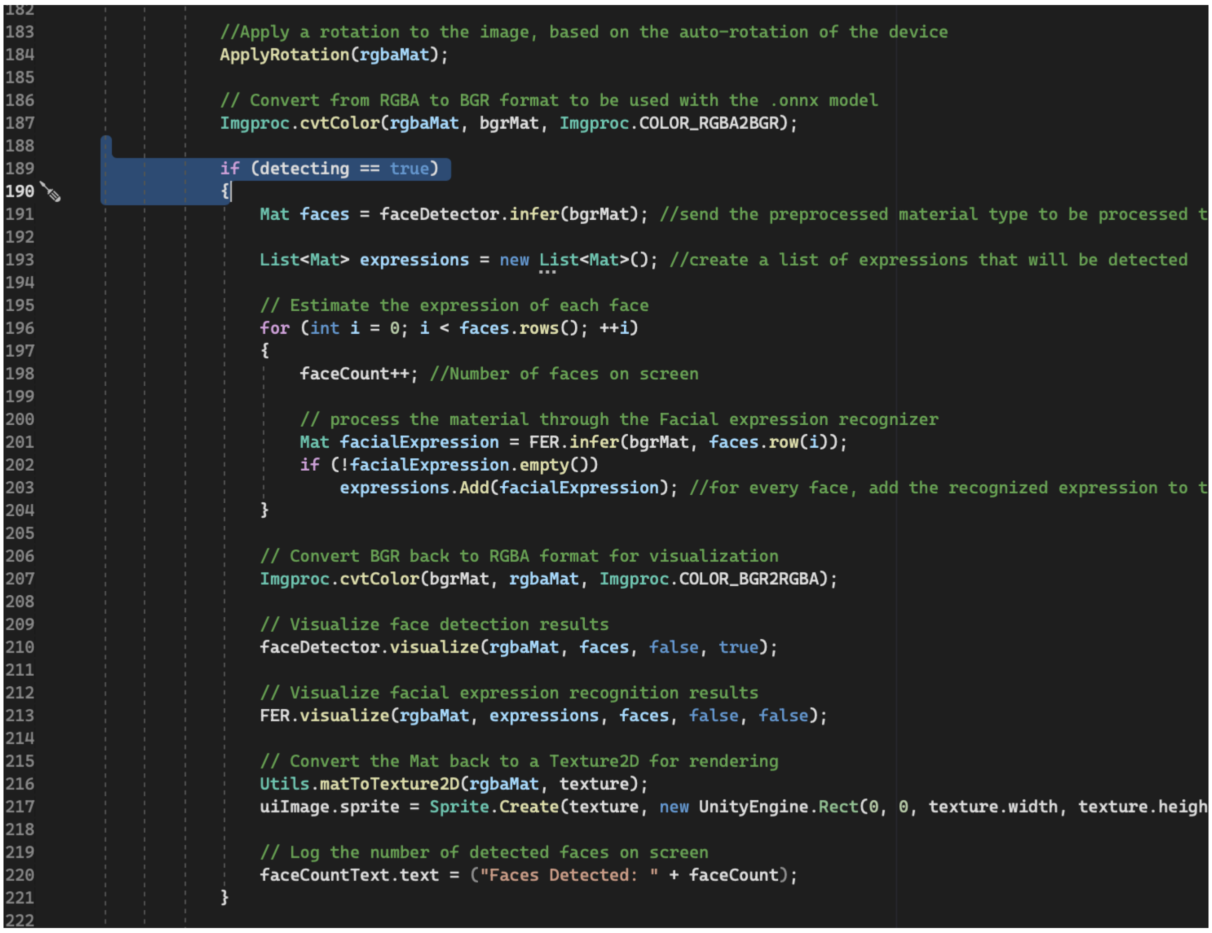
EmotionDetectionScript.cs



```
    Texture2D texture; //Texture to map onto quad - this is what the user will see on the other end
    WebCamTexture webCamTexture; //Texture that is pulled from the camera (updated every frame)
    Mat bgrMat; //Material to input to the model that suits OpenCV BGR format

    public bool detecting = true;

    FacialExpressionRecognizer FER;
    protected static readonly string FER_MODEL_FILENAME = "OpenCVForUnity/dnn/FER.onnx"; //FER model filename
    string FER_model_filepath;

```

I created a public boolean variable called detecting. This is by default set to true. When a screenshot is being taken, I will set 'detecting' to false and thus the bounding box will disappear.

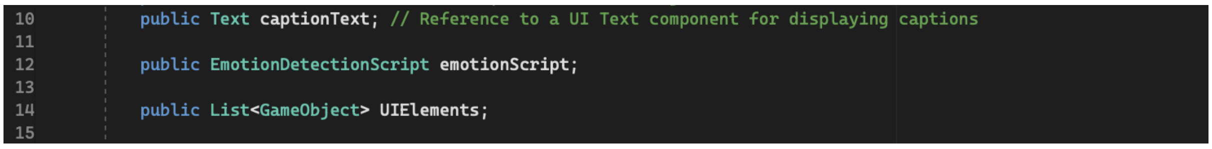
EmotionDetectionScript.cs



```
            //Apply a rotation to the image, based on the auto-rotation of the device
            ApplyRotation(rgbaMat);

            // Convert from RGBA to BGR format to be used with the .onnx model
            Imgproc.cvtColor(rgbaMat, bgrMat, Imgproc.COLOR_RGBA2BGR);

            if (detecting == true)
            {
                Mat faces = faceDetector.infer(bgrMat); //send the preprocessed material type to be processed t

                List<Mat> expressions = new List<Mat>(); //create a list of expressions that will be detected

                // Estimate the expression of each face
                for (int i = 0; i < faces.rows(); ++i)
                {
                    faceCount++; //Number of faces on screen

                    // process the material through the Facial expression recognizer
                    Mat facialExpression = FER.infer(bgrMat, faces.row(i));
                    if (!facialExpression.empty())
                        expressions.Add(facialExpression); //for every face, add the recognized expression to t
                }

                // Convert BGR back to RGBA format for visualization
                Imgproc.cvtColor(bgrMat, rgbaMat, Imgproc.COLOR_BGR2RGBA);

                // Visualize face detection results
                faceDetector.visualize(rgbaMat, faces, false, true);

                // Visualize facial expression recognition results
                FER.visualize(rgbaMat, expressions, faces, false, false);

                // Convert the Mat back to a Texture2D for rendering
                Utils.matToTexture2D(rgbaMat, texture);
                uiImage.sprite = Sprite.Create(texture, new UnityEngine.Rect(0, 0, texture.width, texture.heigh

                // Log the number of detected faces on screen
                faceCountText.text = ("Faces Detected: " + faceCount);
            }

```

I have added an if statement that checks if the program is actively detecting, which is managed by whether or not a screenshot is being taken. I have chosen to place this if

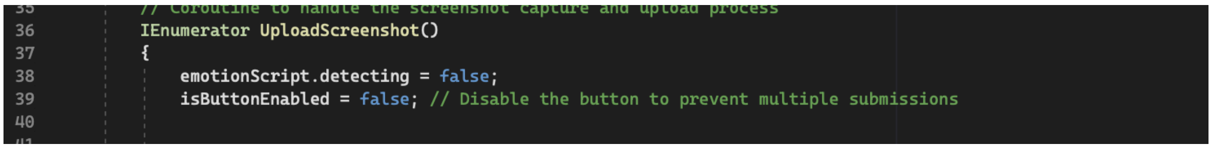
statement after the rendering of the camera on screen. This is so that the user is able to still see the camera output in real-time, regardless of whether a screenshot is being taken or not.
sendScreenshot.cs



```
10      public Text captionText; // Reference to a UI Text component for displaying captions
11
12      public EmotionDetectionScript emotionScript;
13
14      public List<GameObject> UIElements;
15
```

Referencing *EmotionDetectionScript.cs* in order to access the public variable 'detecting'

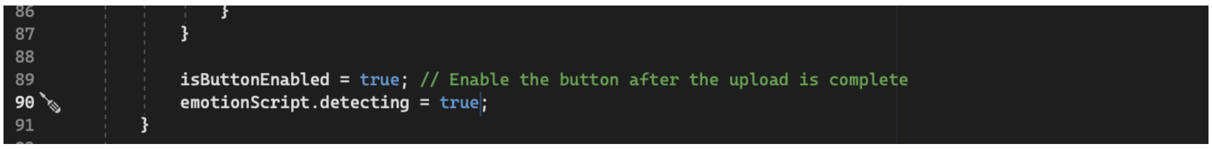
sendScreenshot.cs



```
35      // Coroutine to handle the screenshot capture and upload process
36      IEnumerator UploadScreenshot()
37      {
38          emotionScript.detecting = false;
39          isButtonEnabled = false; // Disable the button to prevent multiple submissions
40
```

At the start of the main screenshot function, turn 'detecting' to false. This will turn off the bounding boxes and emotion detection.

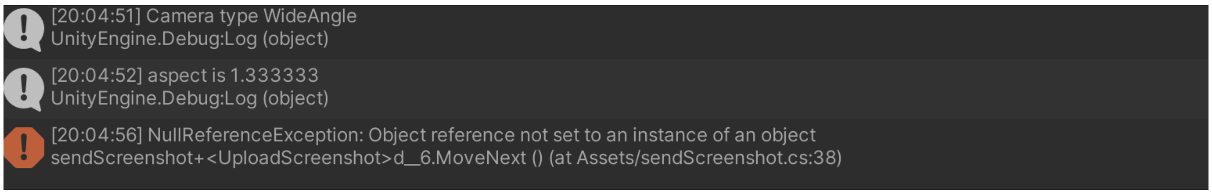
sendScreenshot.cs



```
86              }
87          }
88
89          isButtonEnabled = true; // Enable the button after the upload is complete
90          emotionScript.detecting = true;
91      }
```

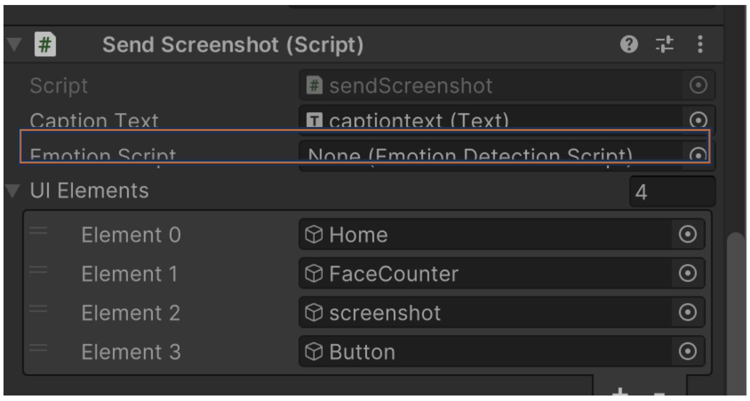
At the end of the main screenshot function, turn 'detecting' to true. This will turn on the bounding boxes again and emotion detection for normal functionality.

When attempting to take a screenshot, I received this error:



```
[20:04:51] Camera type WideAngle
UnityEngine.Debug:Log (object)

[20:04:52] aspect is 1.333333
UnityEngine.Debug:Log (object)

[20:04:56] NullReferenceException: Object reference not set to an instance of an object
sendScreenshot+<UploadScreenshot>d__6.MoveNext () (at Assets/sendScreenshot.cs:38)
```

This indicated that a public object was not set in the inspector.
Upon further debugging, I realised that I hadn't set the emotion detection script in the screenshot script inspector.

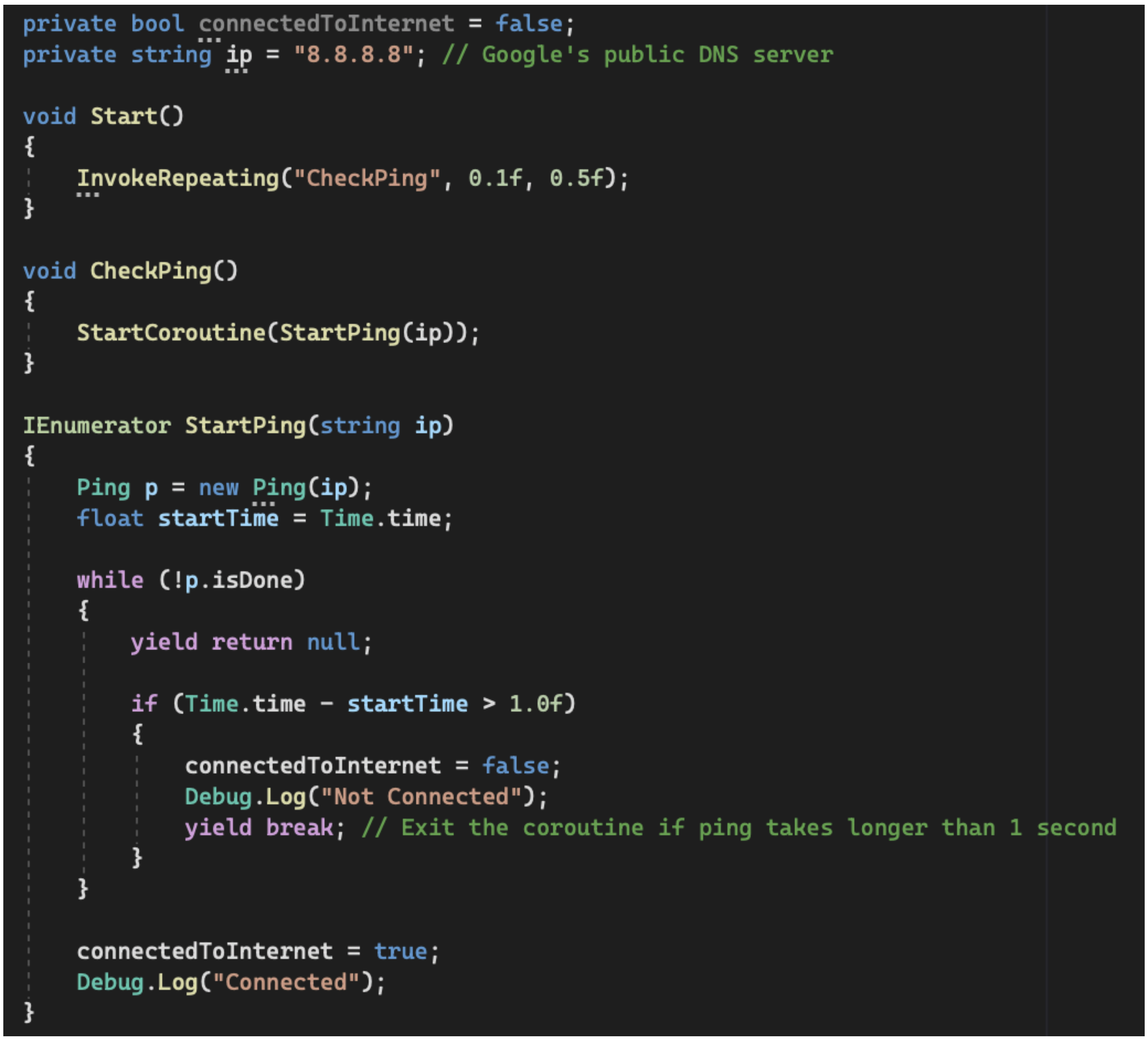

Send Screenshot (Script)

Script	sendScreenshot
Caption Text	captiontext (Text)
Emotion Script	None (Emotion Detection Script)
UI Elements	4
Element 0	Home
Element 1	FaceCounter
Element 2	screenshot
Element 3	Button

After fixing this and setting the emotion script correctly, the functionality now works perfectly, where the only image being screenshotted and sent to the flask server is the camera output. This means that the AI will be able to parse more accurate responses and captions to the user.

Now, all that I need to do is introduce the logic to check if the user is connected to the internet. This is important because if they were not, the image captioning wouldn't work properly.

sendScreenshot.cs



```
private bool connectedToInternet = false;
private string ip = "8.8.8.8"; // Google's public DNS server

void Start()
{
    InvokeRepeating("CheckPing", 0.1f, 0.5f);
}

void CheckPing()
{
    StartCoroutine(StartPing(ip));
}

IEnumerator StartPing(string ip)
{
    Ping p = new Ping(ip);
    float startTime = Time.time;

    while (!p.isDone)
    {
        yield return null;

        if (Time.time - startTime > 1.0f)
        {
            connectedToInternet = false;
            Debug.Log("Not Connected");
            yield break; // Exit the coroutine if ping takes longer than 1 second
        }
    }

    connectedToInternet = true;
    Debug.Log("Connected");
}
```

I have edited the *sendScreenshot.cs* to continuously check for a connection to the internet by pinging Google's public DNS server. This function will be checked every 0.5 seconds. If the time taken to ping is longer than 1 second, assume there is no connection and disable the screenshot functionality.

Test Plan for this version

Due to the nature of my project being dependent on many different functions and inputs, I will need to test that each of the desired outputs are displayed when the expected input is used. To track the test data, I will be using the table below, and providing each test data with a level of priority, (1 being high priority, 2 being mediocre priority, and 3 being low priority)

Test num.	What is being tested	Priority	Input	Expected output	Justification for this data
1	On-screen face counter	1	Device webcam + detected faces	Integer displaying number of faces in real-time	This will aid in accessibility and allow the user to gain a knowledge of the estimated number of people in the image
2	YoLo object detection	3	Device webcam	Coloured boxes are drawn around different objects, and mapped by calculating the coordinates of each object and assigning a label to them.	The YoLo object detection is an extra feature within this project. It is important that this functionality works well, however not critical to the application as this is not going to be the primary focus.
3	Splash screen/scene switcher	1	N/A	The splash screen will allow the user to choose which functionality they would like to use in the app, whether it be emotion/face detection, or a general-purpose object detection AI.	This is quite important as in order for the user to navigate the application, they need a splash screen that will allow it. The splash screen should be the one the loads up when the user opens the application.
4	Check internet connection	2	N/A	If there is an internet connection, debug “connected” if there is no connection, the screenshot button, when pressed will show a “not connected to the internet” text instead of the caption.	This is needed as the image captioning part of the project is hosted on a server and therefore an internet connection is needed. If there is no connection, pressing the button will try to send outbound connections and return errors, therefore by disabling this, we are able to reduce any unnecessary code execution, thus optimizing the code.
5	Image Captioning	1	Send Screenshot button	A small delay, then a message in the middle of the screen	This is the main part of iteration 2 that took me the most time to implement. This is

				displaying an AI generated caption of the camera feed	one of the key aspects in my project. Someone perhaps with visual problems, is able to have the image in front of them described, and then, in a future iteration, this can be coupled with a text-to-speech system so that the caption is spoken aloud to the user.
6	Home screen button functionality	2	‘home’ button	Takes the user back to the splash screen	This is so that, if they wanted, the user can return to the splash screen and choose a different functionality to use for the application
7	Image resolution	3	Device camera	High resolution, but doesn’t make the program slow	This is so that the user can have a reasonable experience when using this app and also be able to distinguish the camera feed at a high resolution.
8	Aspect ratio/rotation	2	Device orientation	Correctly displayed output when using the application	This is so that the image is consistently upright when using the program
9	iPhone deployment	1	n/a	The application runs and works when built on an iphone using xCode for deployment	This will be tested at each iteration in order to ensure that during development, no build errors occur which may cause the application to fail when deploying to an iPhone.

Test Results / Evidence

Test num.	What is being tested	Result	Input	Expected output	Comments	Evidence
1	On-screen face counter	🟩	Device webcam + detected faces	Integer displaying number of faces in real-time	N/A	Figure 2.2
2	YoLo object detection	🟧	Device webcam	Coloured boxes are drawn around different objects, and mapped by calculating the coordinates of each object and assigning a label to them.	Can be very laggy and sometimes crash the application due to the heavy processing required. I will need to optimize this.	Figure Figure 2.5
3	Splash screen/scene switcher	🟧	N/A	The splash screen will allow the user to choose which functionality they would like to use in the app, whether it be emotion/face detection, or a general-purpose object detection AI.	N/A	Figure 2.1
4	Check internet connection	🟧	N/A	If there is an internet connection, debug "connected" if there is no connection, the screenshot button, when pressed will show a "not connected to the internet" text instead of the caption.	N/A	Figure 2.3
5	Image Captioning	🟩	Send Screenshot button	A small delay, then a message in the middle of the screen displaying an AI generated caption of the camera feed	Captions can occasionally be quite inaccurate, this would be due to the model itself.	Figure 2.4
6	Home screen button functionality	🟩	'home' button	Takes the user back to the splash screen	N/A	

7	Image resolution	🟧	Device Camera	High resolution, but doesn’t make the program slow	Object detection has far too many boxes and due to the high resolution, causes large amounts of memory usage and lag. Perhaps I will reduce image quality for Object Detection	All Figures
8	Aspect ratio/rotation	🟧	Device orientation	Correctly displayed output when using the application	Ocassionally, I have found some issues with the Object detection when using in different orientations, it can become super laggy sometimes. Also, when taking a screenshot for image captioning, the image is sometimes flipped. I will need to maintain the ApplyRotation() function while the image is being processed	Figure 2.6
9	iPhone Deployment	🟩	n/a	The application runs and works when built on an iphone using xCode for deployment	n/a	Figures 2.1 2.2 2.4.1 2.6.4

Figure 2.1
Starting up the application takes the user to the home screen

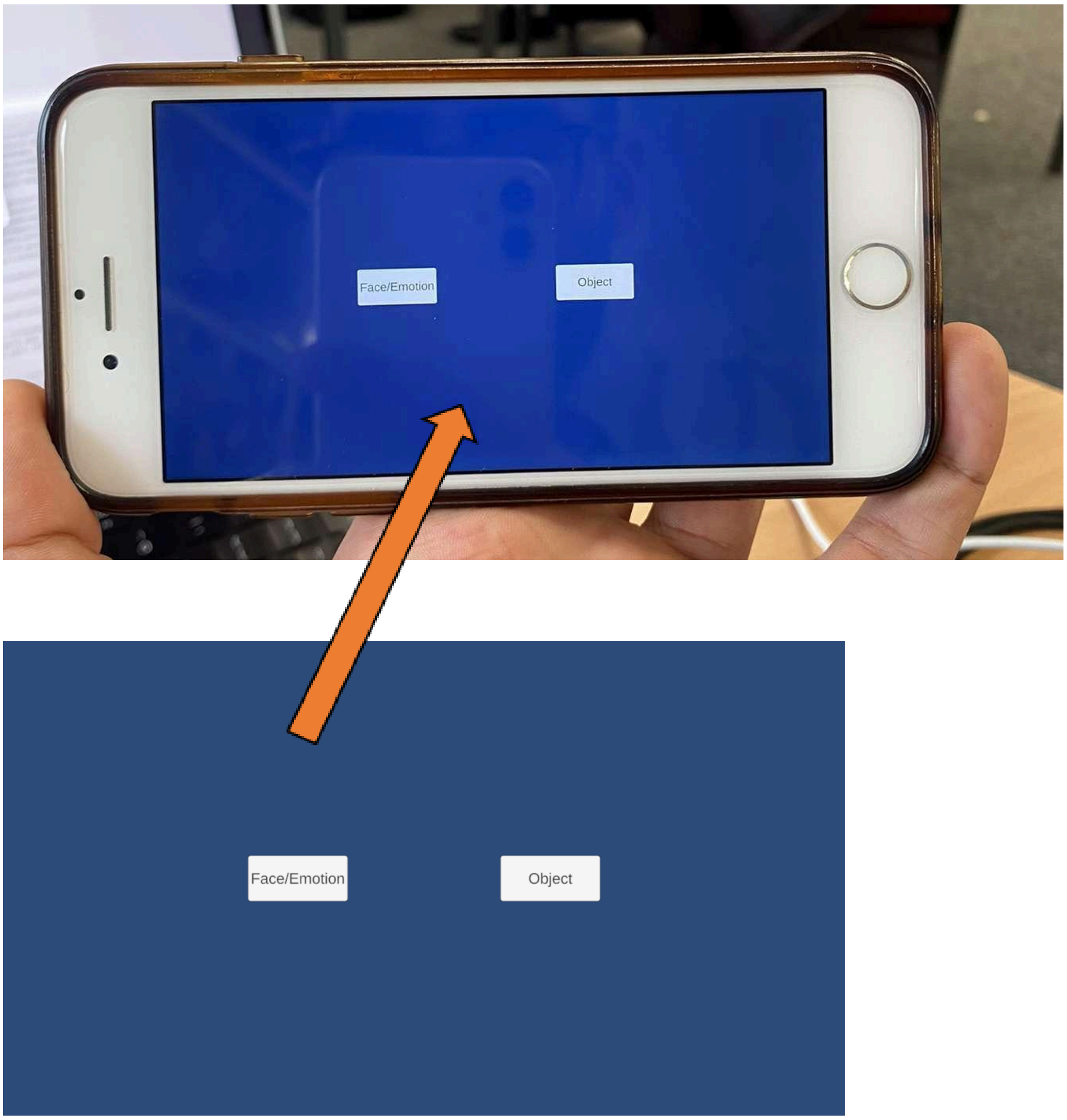


Figure 2.2
Detected faces are updated in real-time

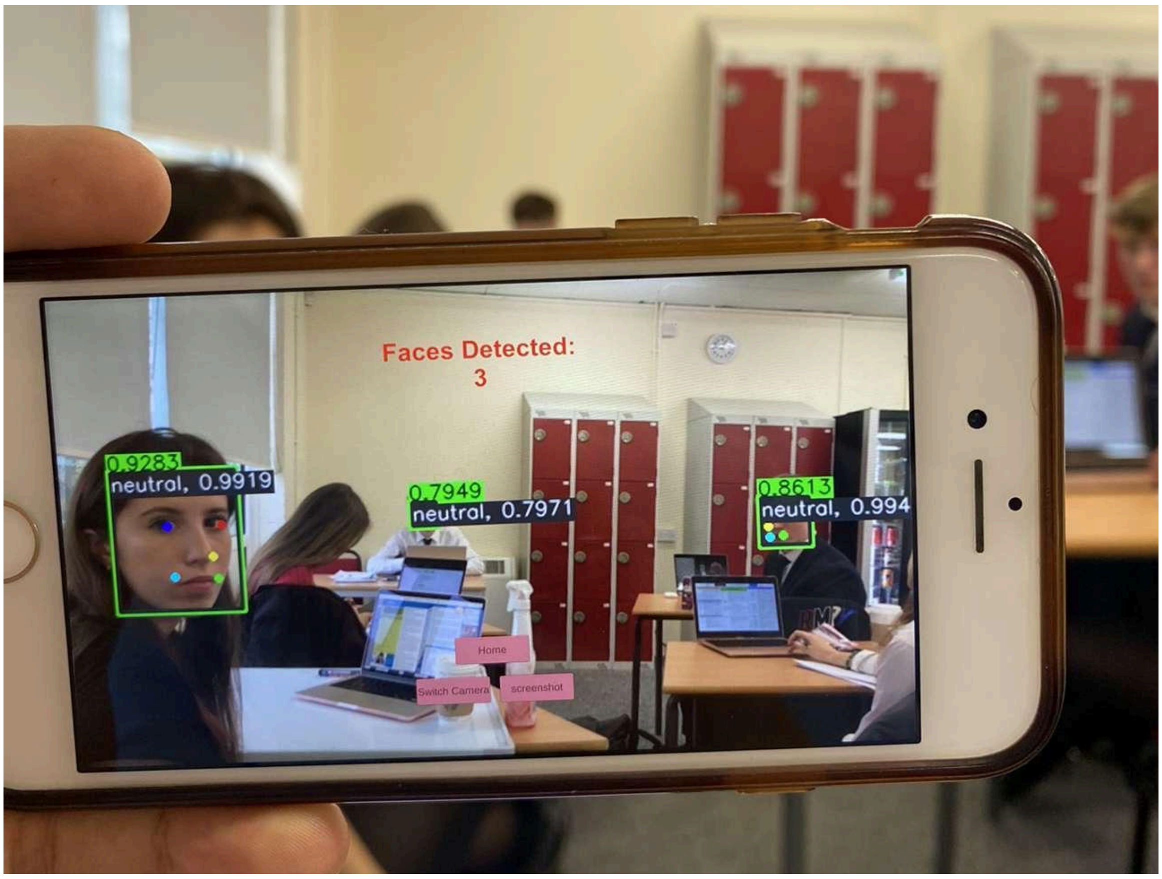
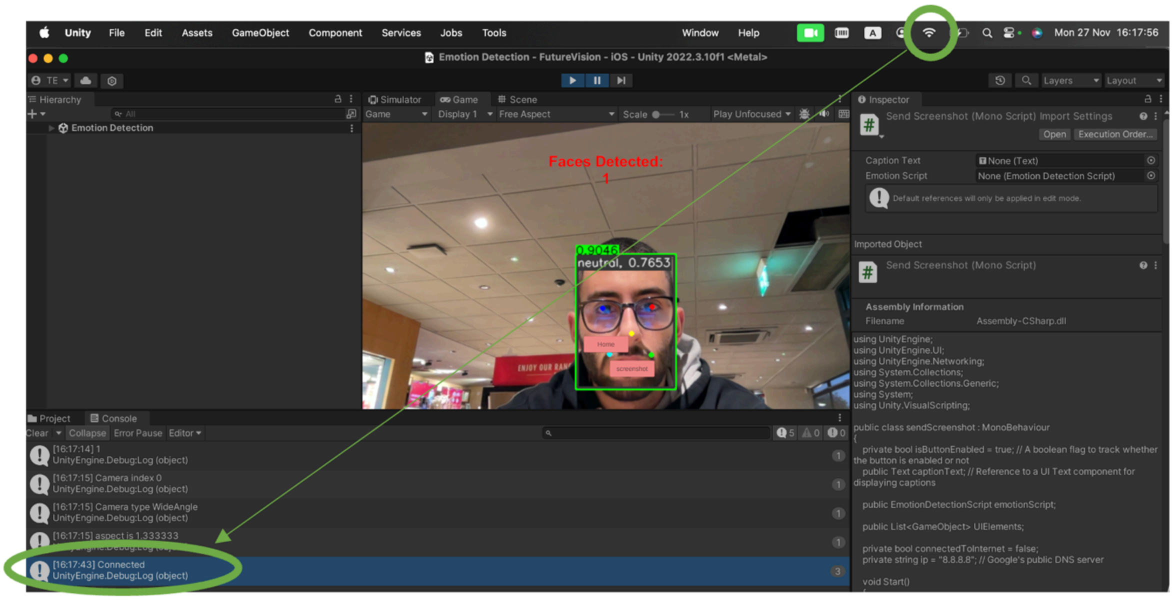


Figure 2.3.1
When the WiFi is on, 'Connected' is printed in the Log


Unity File Edit Assets GameObject Component Services Jobs Tools Window Help
Mon 27 Nov 16:17:56
Emotion Detection - FutureVision - iOS - Unity 2022.3.10f1 <Metal>
Hierarchy
Emotion Detection
Simulator Game Scene
Faces Detected:
1
0.9046
neutral, 0.7653
Home
screenshot
Project Console
Clear Collapse Error Pause Editor
Inspector
Layers Layout

```
[16:17:14] 1
UnityEngine.Debug:Log (object)
[16:17:15] Camera index 0
UnityEngine.Debug:Log (object)
[16:17:15] Camera type WideAngle
UnityEngine.Debug:Log (object)
[16:17:15] aspect is 1.333333
UnityEngine.Debug:Log (object)
[16:17:43] Connected
UnityEngine.Debug:Log (object)
```

Send Screenshot (Mono Script) Import Settings

Caption Text: None (Text)
Emotion Script: None (Emotion Detection Script)

Default references will only be applied in edit mode.

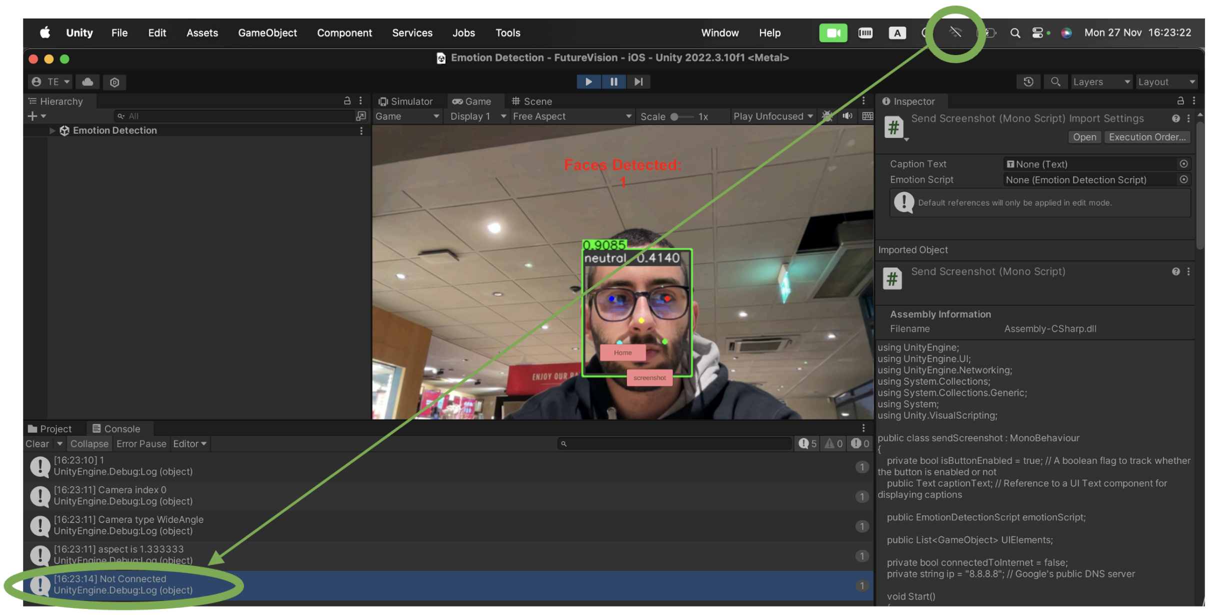
Imported Object

Send Screenshot (Mono Script)

Assembly Information
Filename: Assembly-CSharp.dll

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Networking;
using System.Collections;
using System.Collections.Generic;
using System;
using Unity.VisualScripting;

public class sendScreenshot : MonoBehaviour
{
   private bool isButtonEnabled = true; // A boolean flag to track whether
the button is enabled or not
   public Text captionText; // Reference to a UI Text component for
displaying captions

   public EmotionDetectionScript emotionScript;

   public List<GameObject> UIElements;

   private bool connectedToInternet = false;
   private string ip = "8.8.8.8"; // Google's public DNS server

   void Start()
```

Figure 2.3.2
When the WiFi is off, 'Not connected' is printed in the Log


Unity File Edit Assets GameObject Component Services Jobs Tools Window Help
Mon 27 Nov 16:23:22
Emotion Detection - FutureVision - iOS - Unity 2022.3.10f1 <Metal>
Hierarchy
Emotion Detection
Simulator Game Scene
Game Display 1 Free Aspect Scale 1x Play Unfocused
Faces Detected: 1
0.9085
neutral 0.4140
Home
screenshot
Layers Layout
Inspector
Project Console
Clear Collapse Error Pause Editor
5 0 0

Send Screenshot (Mono Script) Import Settings

Open | Execution Order...

Caption Text	None (Text)
Emotion Script	None (Emotion Detection Script)

Default references will only be applied in edit mode.

Imported Object

Send Screenshot (Mono Script)

Assembly Information

Filename	Assembly-CSharp.dll

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Networking;
using System.Collections;
using System.Collections.Generic;
using System;
using Unity.VisualScripting;

public class sendScreenshot : MonoBehaviour
{
   private bool isButtonEnabled = true; // A boolean flag to track whether
the button is enabled or not
   public Text captionText; // Reference to a UI Text component for
displaying captions

   public EmotionDetectionScript emotionScript;

   public List<GameObject> UIElements;

   private bool connectedToInternet = false;
   private string ip = "8.8.8.8"; // Google's public DNS server

   void Start()
```

```
[16:23:10] 1
UnityEngine.Debug:Log (object)                                  1
[16:23:11] Camera index 0
UnityEngine.Debug:Log (object)                                  1
[16:23:11] Camera type WideAngle
UnityEngine.Debug:Log (object)                                  1
[16:23:11] aspect is 1.333333
UnityEngine.Debug:Log (object)                                  1
[16:23:14] Not Connected
UnityEngine.Debug:Log (object)                                  1
```

Figure 2.4.1
Images are captioned E.g.1
The caption states "A woman sitting at a desk in front of a laptop computer"

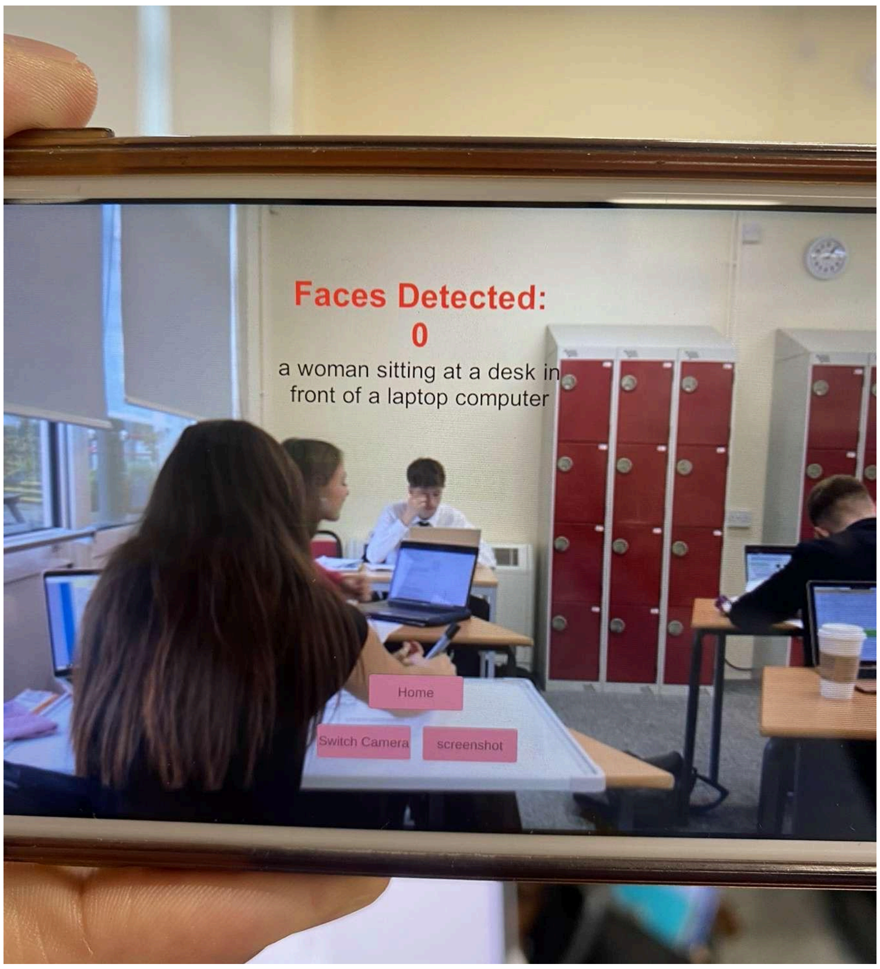


Figure 2.4.2

Images are captioned E.g. 2
The caption states "A man is looking at a television screen in a store"

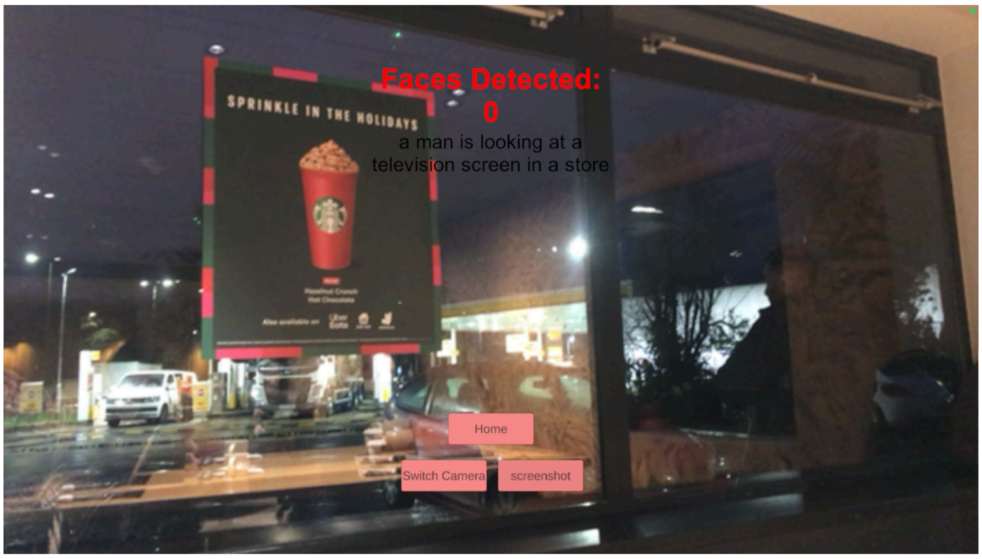


Figure 2.4.3
Images are captioned E.g. 3
The caption states "A man in a black shirt is holding a skateboard"

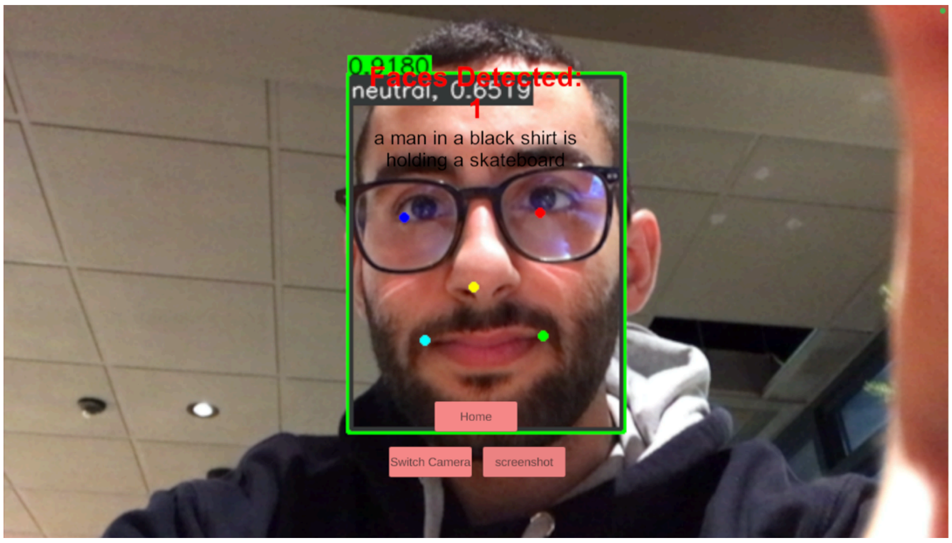


Figure 2.5.1
Object detection

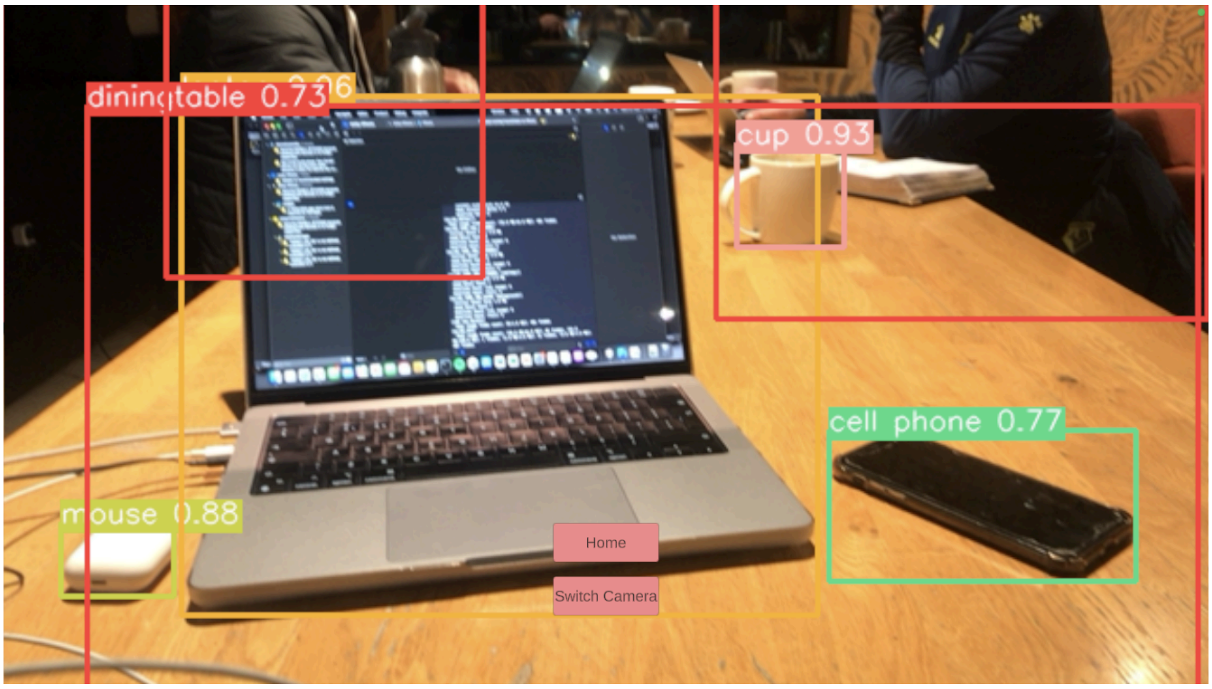


Figure 2.5.2
Object Detection

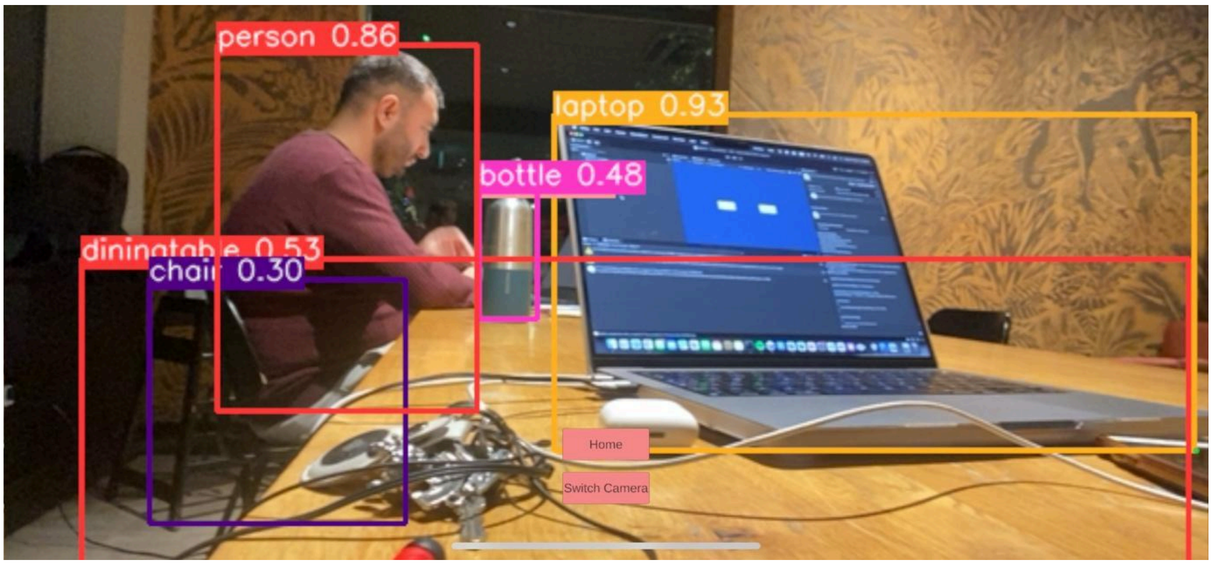
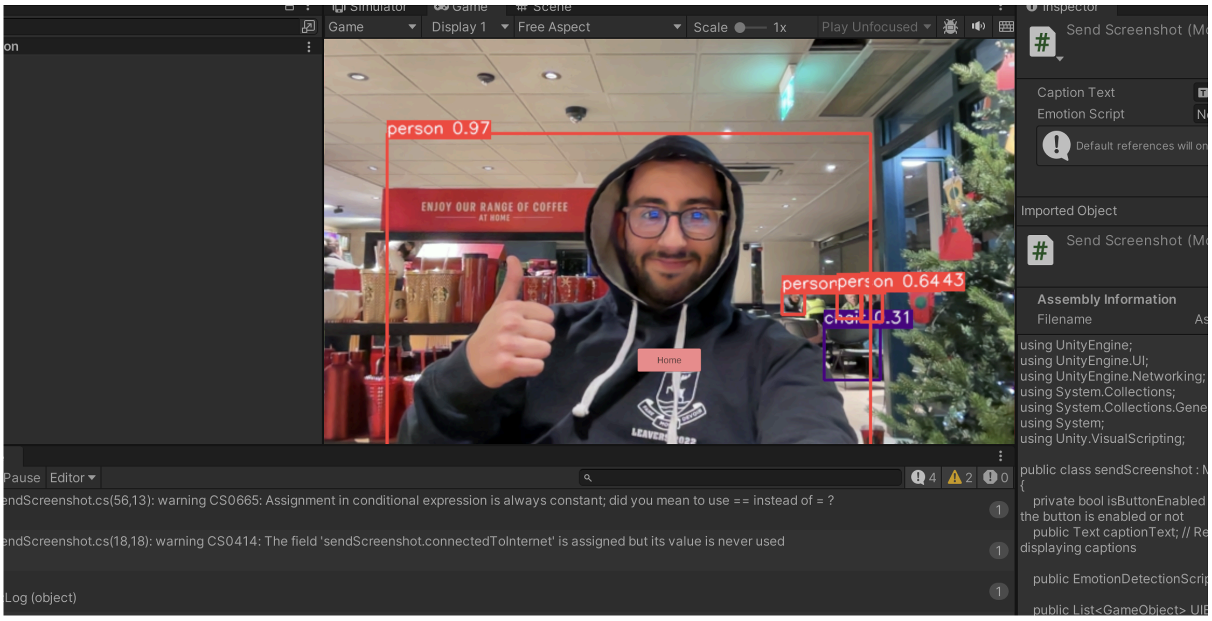


Figure 2.5.3
Object Detection



```
endScreenshot.cs(56,13): warning CS0665: Assignment in conditional expression is always constant; did you mean to use == instead of = ?
endScreenshot.cs(18,18): warning CS0414: The field 'sendScreenshot.connectedToInternet' is assigned but its value is never used
Log (object)
```

Send Screenshot (M
Caption Text
Emotion Script
Default references will on
Imported Object
Send Screenshot (M
Assembly Information
Filename

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Networking;
using System.Collections;
using System.Collections.Gene
using System;
using Unity.VisualScripting;

public class sendScreenshot : M
{
   private bool isButtonEnabled
the button is enabled or not
   public Text captionText; // Re
displaying captions

   public EmotionDetectionScrip

   public List<GameObject> UIE
```

Figure 2.6.1
Before taking screenshot (upright)

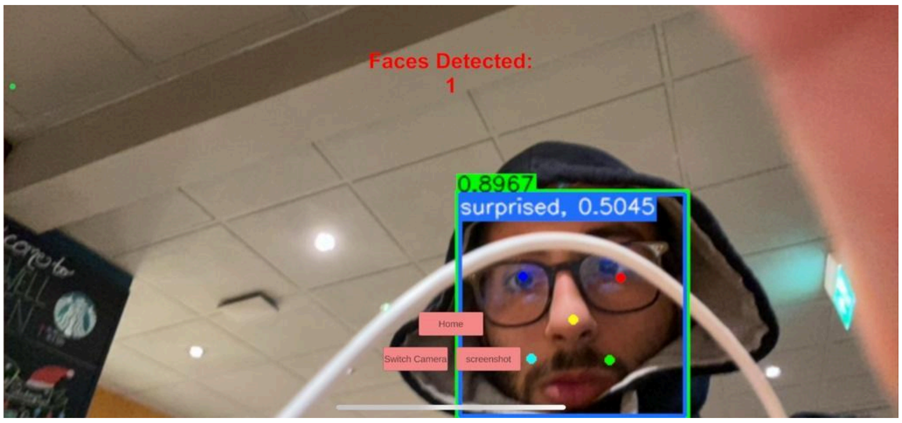


Figure 2.6.2
While taking screenshot (EmotionDetection script is disabled so ApplyRotation() function is not working therefore image is upside down while processing)

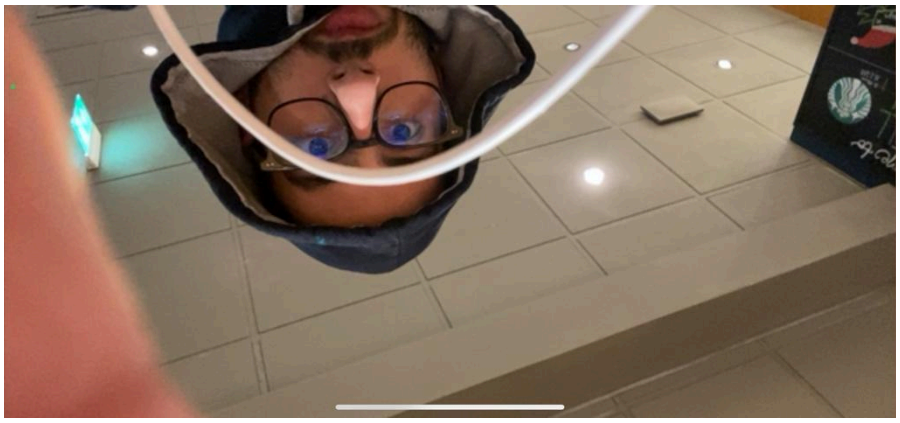


Figure 2.6.3
After caption is returned (EmotionDetection is re-enabled and thus ApplyRotation() is being called every frame so the image is corrected)

(The caption states "A person in a hospital bed with a tube in their mouth")

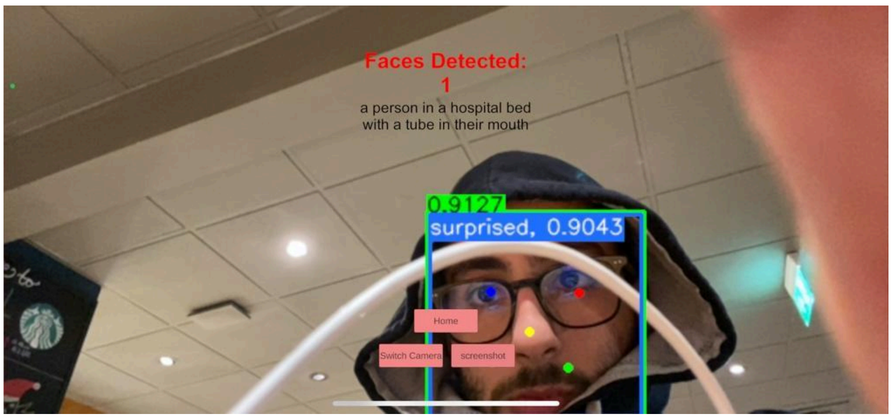
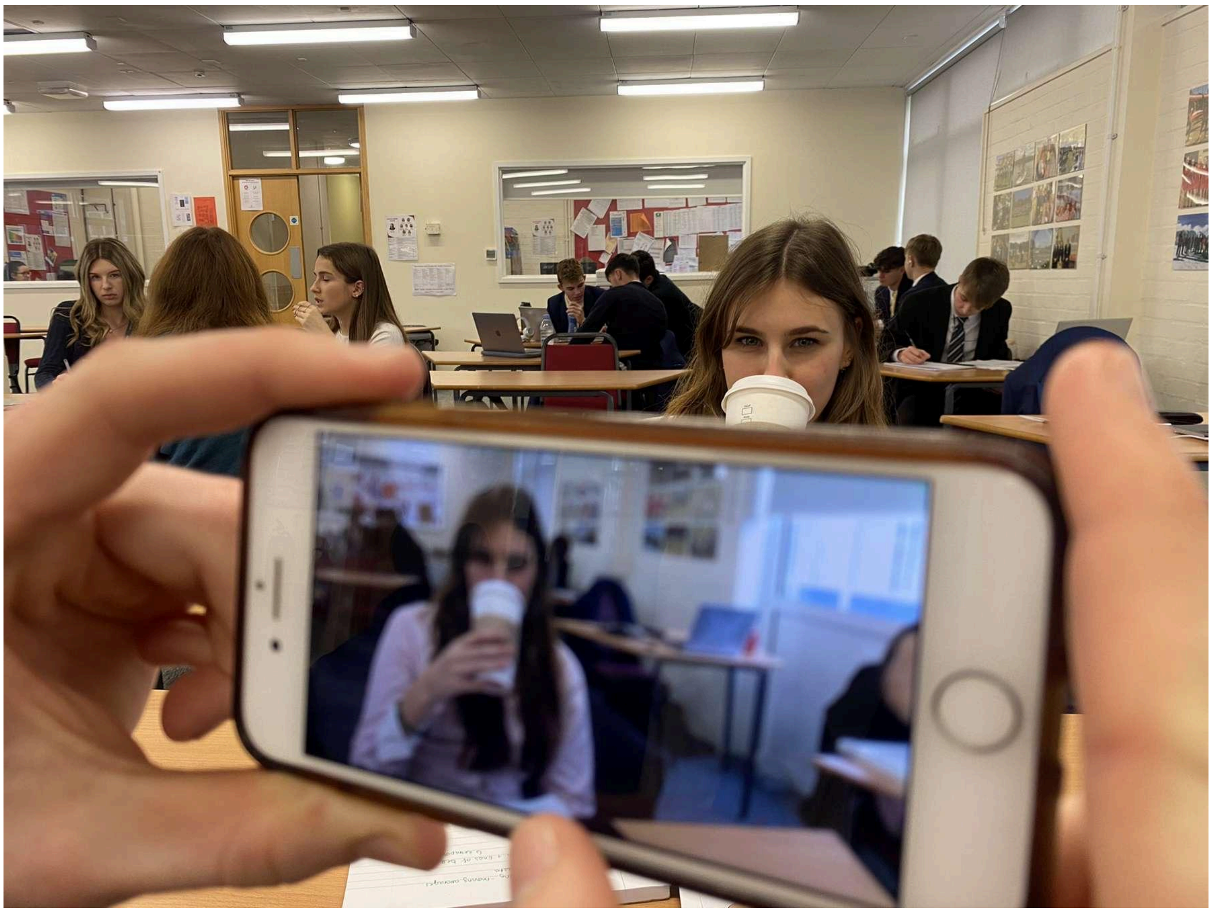


Figure 2.6.4
Sometimes, the camera will not flip when taking a screenshot and sending it. This depends on the device rotation and is a bug within Unity's camera texture logic.

(UI objects disappear while screenshot being taken and sent to not distract the AI with unnecessary objects (abstraction))



Feedback from Stakeholder

As I mentioned earlier, my stakeholders are a mix of potential users and tech enthusiasts who hope to gain more knowledge about the common applications of artificial intelligence or who would like to use this project to improve their personal lives. Since I have implemented many key features now into the project, I will interview all 3 stakeholders: Aiad Tarik, Mario Prifti, and Mike Parish this time, as I believe this project is now fairly usable by him. The 3 stakeholders will be referred to as AT, MPr and MPa to indicate the answers that they will provide.

Does the app feel seamless and easy to use?

AT: The app overall feels quite seamless and easy to use. The facial recognition system works smoothly, and the object detection, despite occasional crashes, shows promise. However, the UI issues during the screenshot process could be addressed for a more seamless experience.

MPr: The app has a cool vibe, but I think there's room for improvement in terms of user engagement. Adding some visually appealing elements or interactive features on the home/splash screen could make it more interesting for users of all ages.

MPa: The app feels a bit confusing for me at times, especially with the landscape-only mode. I'm more used to portrait orientation. A simpler navigation system might make it easier for users like me.

How do you feel about the accuracy of the image captioning?

AT: The accuracy of image captioning is quite weak. It occasionally provides some information about the images, aiding users with visual impairments effectively, however, most of the time the information is incorrect or out-of-context, perhaps a more heavily-trained model would help.

MPr: The image captioning is a great feature. It adds a creative touch to the app and makes it more enjoyable for users, however it doesn’t always caption the image properly

MPa: The image captioning is quite confusing. The idea is good but it frankly doesn’t work well enough.

How do you feel about the accuracy of object detection?

AT: Object detection, despite occasional crashes, performs well when it works. Improving memory management to prevent crashes would significantly enhance the user experience.

MPr: Object detection is good, but those occasional crashes aren't. Adding some creative loading animations or messages during processing could make the user experience more engaging, even when there are hiccups.

MPa: Although a nice feature, I don't think it is very necessary, particularly the crashes make it a bit frustrating.

How do you feel about maintaining a landscape rotation for the use of this application?

AT: While I understand the challenges with auto-rotation, maintaining only landscape mode may limit user preferences. Exploring a solution to improve auto-rotation functionality would be beneficial for user convenience.

MPr: Landscape mode is okay, but having the option for both portrait and landscape would be great. People have different preferences, and giving them the choice would make the app more versatile.

MPa: I prefer apps that work in portrait mode too. It would be nice if the app supported both orientations for a better user experience.

Are there any improvements you would recommend to the currently implemented features?

AT: Improvements could be made in handling the UI during the server processing time after taking a screenshot. A smoother transition and faster response from the server would enhance the user experience significantly. Additionally, resolving the occasional crashes during object detection would be crucial.

MPr: Maybe a progress bar or a visually appealing animation could distract users from the wait time during the image processing. Also, making the object detection run at a higher frame rate and smoother. This would really help out.

MPa: For someone like me, the app might benefit from a more straightforward layout and maybe a tutorial to explain the features. Also, improving the image captioning and fixing the crashes during object detection is vital.

Changes/Fixes that I now plan to make to the design or code as a result of testing and feedback

Fixes that I now plan to make to the design or code as a result of testing and feedback in this iteration include addressing several user experience and functionality aspects. The primary concern is the occasional crashes during object detection, likely attributed to memory management issues. To mitigate this, I will conduct a thorough review of the code, focusing on optimising memory usage and enhancing error handling to prevent abrupt application closures. Additionally, the UI disruptions during the screenshot process, where the screen momentarily flips and delays in UI elements reappearing, will be addressed to ensure a smoother transition. Furthermore, I acknowledge the feedback regarding the limitation of landscape-only mode and will work on incorporating both portrait and landscape orientations for improved user flexibility. These adjustments aim to enhance the overall usability and reliability of the app, providing a more seamless and inclusive experience for users across various backgrounds and technological proficiencies.

For my next iteration, I plan to add:

- text-to-speech
- variable confidence metre
- Improve UI (this will be the main focus of iteration 3, after addressing the bug fixes)

Evaluation

In this second iteration, my focus was on expanding the app's capabilities by integrating a button for image captioning and incorporating an object-detection section. A significant portion of my time was devoted to researching CNN and RNN AI models, with the original intention of creating a custom model. However, after encountering challenges and weeks of unsuccessful attempts, I opted for a pre-trained model from HuggingFace. Notably, I added an on-screen face counter to enhance user feedback. Despite these positive advancements, several bugs surfaced during testing. The object-detection feature occasionally led to app crashes, likely stemming from memory management issues. Furthermore, pressing the screenshot button resulted in the disappearance of UI elements, initiating a server process for caption processing. The screen flipping during this process and delayed UI element reappearance pose usability concerns. Auto-rotation issues led to a decision to maintain landscape mode exclusively. A home/splash screen was introduced in this iteration, allowing users to select application functionality. The app's overarching goal, aiding visually impaired individuals in identifying their surroundings, remains a priority. Future iterations may explore the incorporation of text-to-speech functionality, extending beyond image captioning to include features like facial recognition. The image captioning server, developed using Python and Flask, presented a learning curve, and the integration of HuggingFace into the project via a server required substantial research. Overcoming challenges, I acquired the skills to make and receive Flask requests in C#, demonstrating adaptability and persistence in addressing new technologies and methodologies.

Iteration 3 - *Date 03/10/2023*

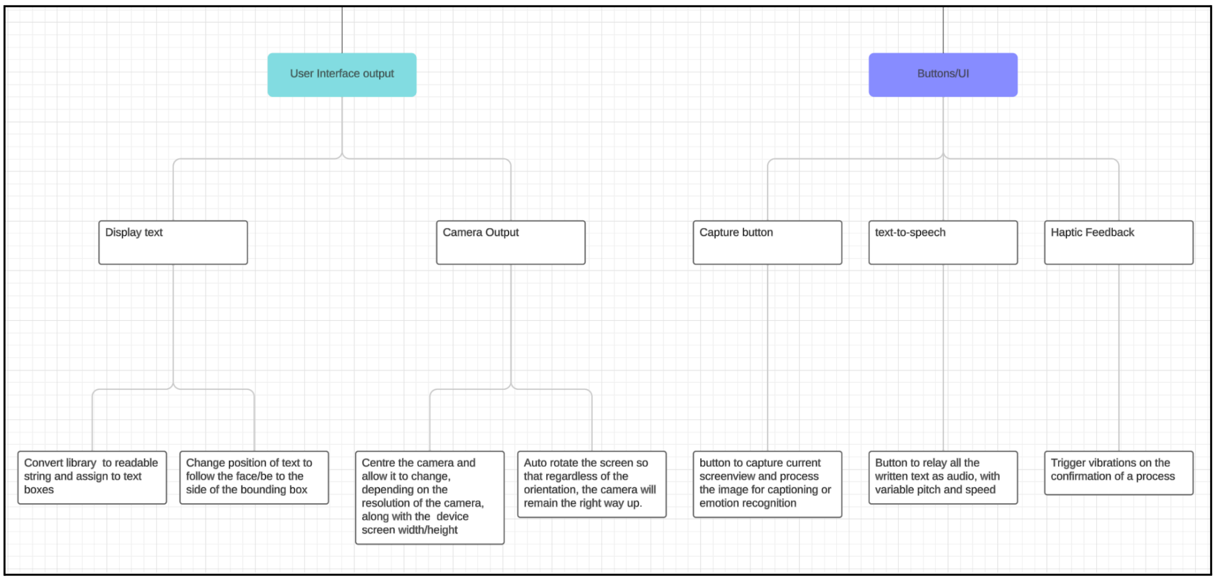
Aims for this iteration

In iteration 3, the primary goal is to enhance the front-end and user interface (UI) to elevate the overall user experience and accessibility. My focus remains on preserving the core functionality of the main program while introducing additional accessibility features, including potential audible triggers.

By the conclusion of this iteration, my aim is to deliver a fully operational, visually pleasing UI and UX that aligns seamlessly with the pre-established design specifications for this project. The intention is to create a user interface that is not only aesthetically pleasing but also intuitive and accessible to a wide range of users.

To achieve this, I plan to incorporate open-source designed buttons and images, ensuring a cohesive and professional look for the project. The overarching objective is to make the application simple and user-friendly, catering to the needs of all individuals.

Below are some of the functions from the top-down design that I will be working on and improving from iteration 1 and 2 during this portion of development:



User Interface output
Buttons/UI
Display text
Camera Output
Capture button
text-to-speech
Haptic Feedback
Convert library to readable string and assign to text boxes
Change position of text to follow the face/be to the side of the bounding box
Centre the camera and allow it to change, depending on the resolution of the camera, along with the device screen width/height
Auto rotate the screen so that regardless of the orientation, the camera will remain the right way up.
button to capture current screenview and process the image for captioning or emotion recognition
Button to relay all the written text as audio, with variable pitch and speed
Trigger vibrations on the confirmation of a process

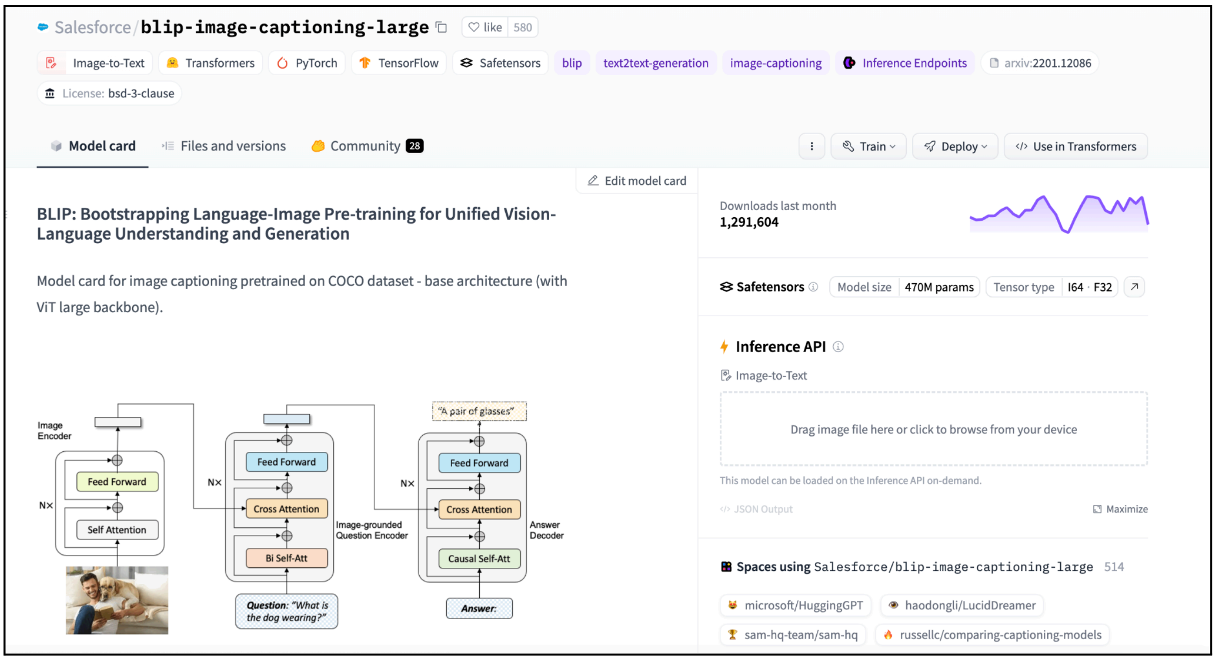
The functions above that have been included in this iteration are primarily focused on extra, further development, with a focus on UI and buttons, haptic feedback and text-to-speech, allowing for advanced accessibility, which is a key element of this project.

I had planned to include a variable settings slider/button during iteration 2, however due to time constraints, I will attempt to develop this into the project within this iteration; a settings\pause button where the user can alter some of the program's functionality, such as the confidence threshold, text-to-speech volume etc.

Bug Fixing:

One of the main pieces of feedback that I received from my stakeholders during this iteration was the accuracy of the captioned models throughout testing. This can be seen in figure 2.4.2, 2.4.3, and 2.6.3 where the captions are quite inaccurate and do not tell the contents of the image correctly. To counter this, I returned to Huggingface.co and selected a different pre-trained model. This time, one that was trained on the COCO dataset, a large-scale object detection, segmentation, and captioning dataset, with over 330,000 images, each with 5 captions per image, leading to over 1.5 million total captions to train off. This will in turn, hopefully produce a more accurate image captioning model.

The updated model that I will be using from HuggingFace.co



Salesforce/**blip-image-captioning-large** like 580

Image-to-Text · Transformers · PyTorch · TensorFlow · Safetensors · blip · text2text-generation · image-captioning · Inference Endpoints · arxiv:2201.12086

License: bsd-3-clause

Model card · Files and versions · Community 28 · Train · Deploy · Use in Transformers

Edit model card

BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation

Model card for image captioning pretrained on COCO dataset - base architecture (with ViT large backbone).

Downloads last month
1,291,604

Safetensors · Model size 470M params · Tensor type I64 · F32

Inference API

Image-to-Text

Drag image file here or click to browse from your device

This model can be loaded on the Inference API on-demand.

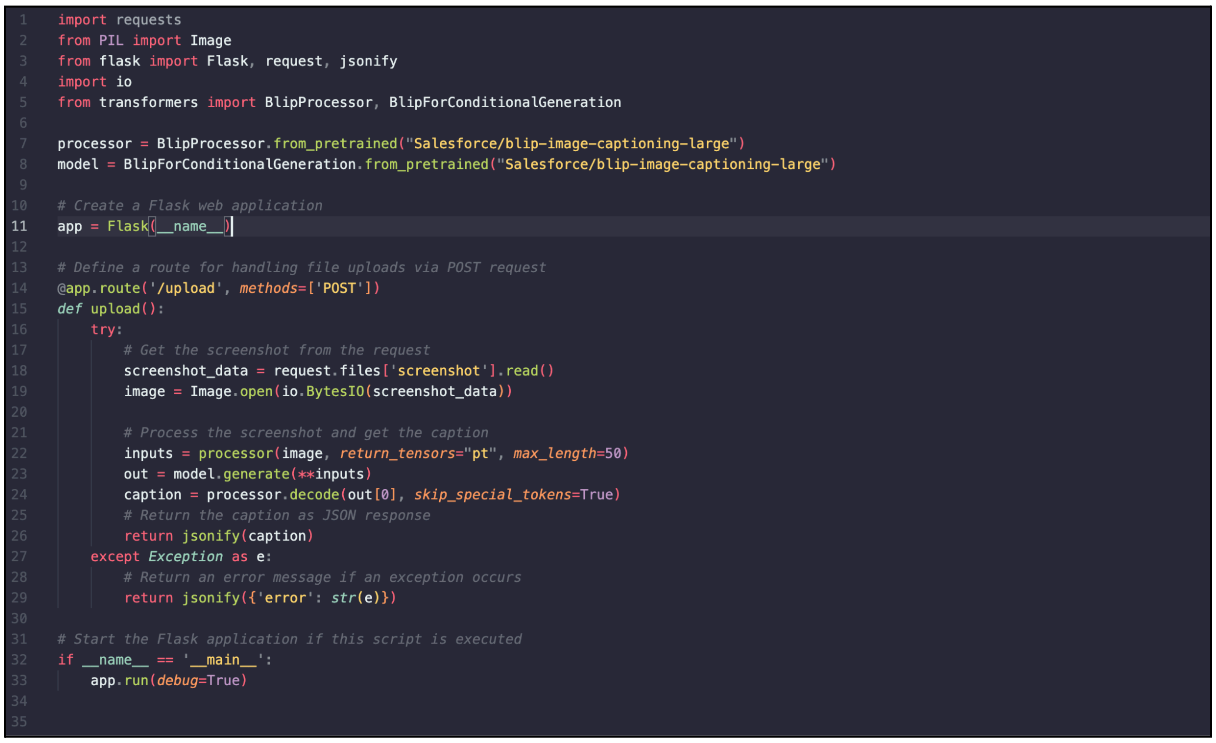
JSON Output · Maximize

Spaces using Salesforce/blip-image-captioning-large 514

microsoft/HuggingGPT · haodongli/LucidDreamer · sam-hq-team/sam-hq · russellc/comparing-captioning-models

To use this model, I will have to update my python server code, and perhaps my C# code for receiving the captions and decoding them.

model.py

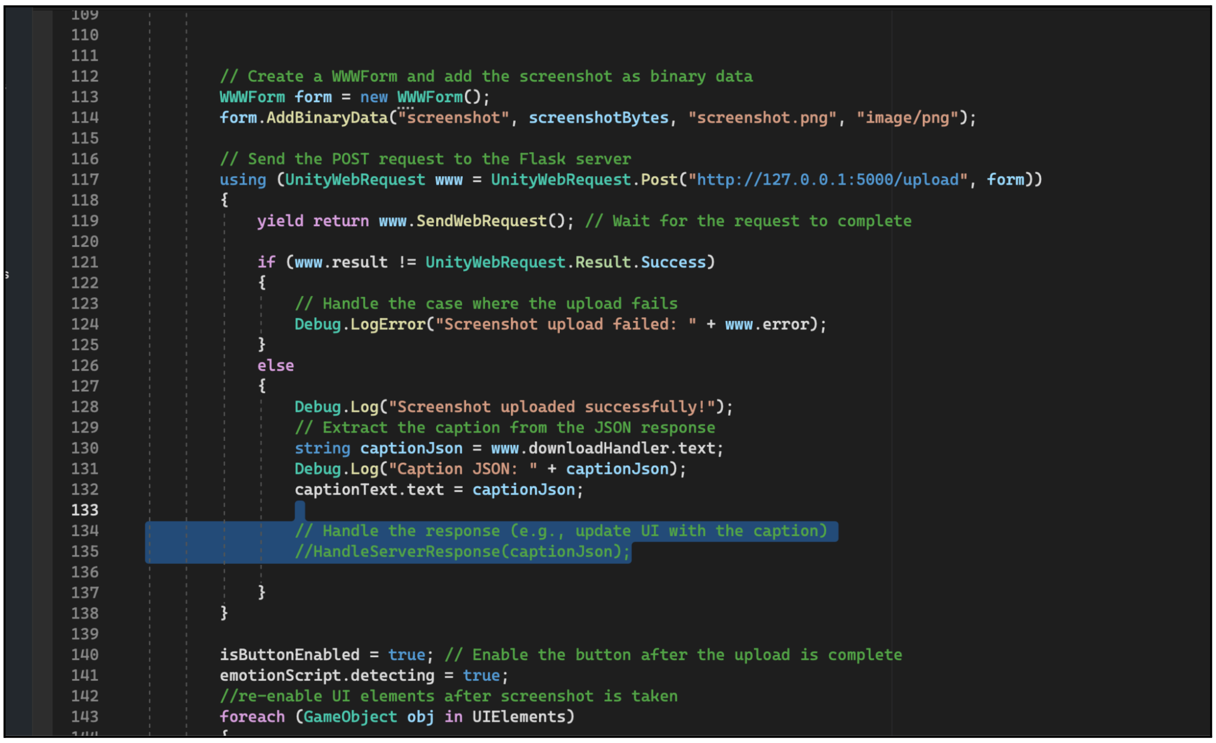


```python
import requests
from PIL import Image
from flask import Flask, request, jsonify
import io
from transformers import BlipProcessor, BlipForConditionalGeneration

processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-large")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-large")

# Create a Flask web application
app = Flask(__name__)

# Define a route for handling file uploads via POST request
@app.route('/upload', methods=['POST'])
def upload():
    try:
        # Get the screenshot from the request
        screenshot_data = request.files['screenshot'].read()
        image = Image.open(io.BytesIO(screenshot_data))

        # Process the screenshot and get the caption
        inputs = processor(image, return_tensors="pt", max_length=50)
        out = model.generate(**inputs)
        caption = processor.decode(out[0], skip_special_tokens=True)
        # Return the caption as JSON response
        return jsonify(caption)
    except Exception as e:
        # Return an error message if an exception occurs
        return jsonify({'error': str(e)})

# Start the Flask application if this script is executed
if __name__ == '__main__':
    app.run(debug=True)
```

I have updated the model script. This time, It is using the *Salesforce/blip-image-captioning-large* model that was documented to be found more accurate than other models.

For this model, I do not have to set parameters for text generation nor do I have to extract pixel values. Rather I simply set an input, deduce the output destination and the caption to be extracted for sending to the main application. This is a much simpler script than the original model that I was using in iteration 2 as all of the processing and extraction is done when generating the model outputs, thus reducing any chance of error and improving efficiency.

sendScreenshot.cs



```
// Create a WWWForm and add the screenshot as binary data
WWWForm form = new WWWForm();
form.AddBinaryData("screenshot", screenshotBytes, "screenshot.png", "image/png");

// Send the POST request to the Flask server
using (UnityWebRequest www = UnityWebRequest.Post("http://127.0.0.1:5000/upload", form))
{
    yield return www.SendWebRequest(); // Wait for the request to complete

    if (www.result != UnityWebRequest.Result.Success)
    {
        // Handle the case where the upload fails
        Debug.LogError("Screenshot upload failed: " + www.error);
    }
    else
    {
        Debug.Log("Screenshot uploaded successfully!");
        // Extract the caption from the JSON response
        string captionJson = www.downloadHandler.text;
        Debug.Log("Caption JSON: " + captionJson);
        captionText.text = captionJson;

        // Handle the response (e.g., update UI with the caption)
        //HandleServerResponse(captionJson);

    }
}

isButtonEnabled = true; // Enable the button after the upload is complete
emotionScript.detecting = true;
//re-enable UI elements after screenshot is taken
foreach (GameObject obj in UIElements)
```

Since the output is in a simple Json string format, there is no need to extract words from an array, so I can remove the *HandleServerResponse()* function from the script, and simply attach the returned Json string from the server to the gameobject text field. This also simplifies the code and reduces the risk of potential errors.

In terms of the lagging when using the object detection section of the application, I was unable to find an immediate solution to this problem and so I will continue with development and hope to tackle this further down the line.

Summary of aims:

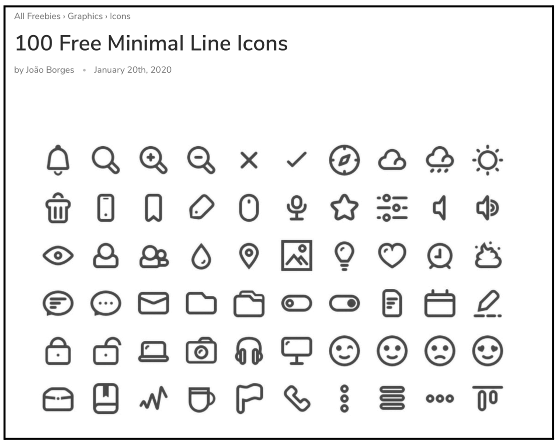
- Improve home screen UI
    - Minimalist, but visually appealing and easy-to-use buttons
- Settings button, to adjust various thresholds
- Improve general UI and responsiveness of application
- If time allows, incorporate text-to-speech for captioned outputs
- Add loading animation while caption is being generated
    - If time exceeds, cancel request and return to normal functionality

Annotated code screenshots with description

UI overhaul

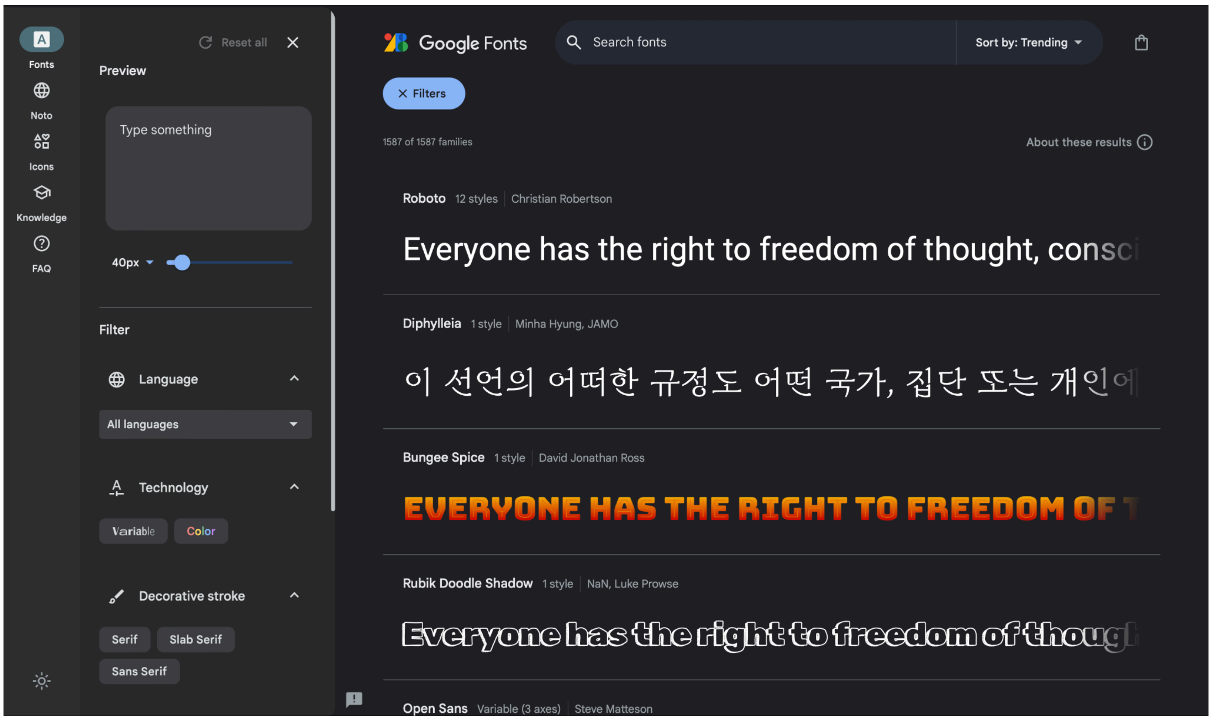
In this iteration, my primary goal was to enhance user experience and UI by redesigning the entire interface, starting with the home page/landing screen. This involved a comprehensive overhaul to address existing usability issues and create a more intuitive and visually appealing interface. The decision to prioritise the home page recognizes its crucial role as the digital gateway to our platform. The revamped design focuses on refining layout, optimising navigation, and streamlining the user's journey for a more gratifying digital experience. This iterative process reflects a commitment to continuous improvement and a user-centric approach in the evolution of our platform.

In this redo, my main aim is to give the interface a simple and clean look. I want it to be easy for everyone to use, without unnecessary complications. Going for a minimalist style means fewer distractions and a more straightforward design. This not only makes things accessible for all users but also looks neat and tidy. The idea is to strike a balance between being easy to use and looking good – making sure it's a smooth experience for everyone who uses the platform.


All Freebies › Graphics › Icons
100 Free Minimal Line Icons
by João Borges • January 20th, 2020

I've opted for a straightforward grey and white icon pack for the buttons, aiming to maintain a clean and uncluttered visual style. The simplicity of these icons aligns with the overall minimalist design approach I'm implementing, ensuring that they seamlessly integrate with the interface without overwhelming the user. By using a consistent colour scheme, specifically grey and white, the buttons maintain a cohesive and unified appearance. This not only enhances the aesthetic appeal but also promotes clarity and ease of recognition for users. The selected icon pack serves not only as a functional element for navigation but also as a visual cue, contributing to an intuitive and user-friendly experience throughout the platform.

For the button text fonts, I am using google fonts to browse and search for a font that would match the current style of the program. It should be easily readable and easy on the eye, while also being simple and basic.



40px

All languages

Sort by: Trending

1587 of 1587 families

Roboto 12 styles | Christian Robertson

Everyone has the right to freedom of thought, consc

Diphylleia 1 style | Minha Hyung, JAMO

이 선언의 어떠한 규정도 어떤 국가, 집단 또는 개인에

Bungee Spice 1 style | David Jonathan Ross

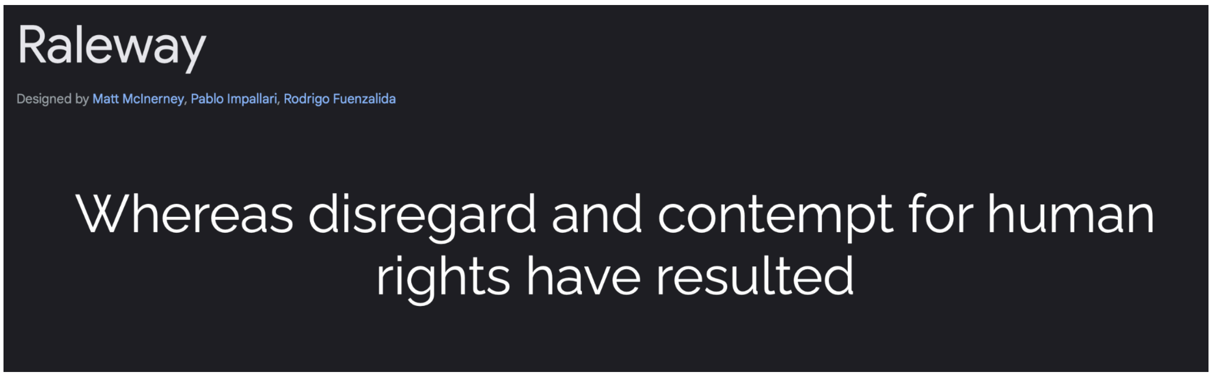
EVERYONE HAS THE RIGHT TO FREEDOM OF T

Rubik Doodle Shadow 1 style | NaN, Luke Prowse

Everyone has the right to freedom of thoug

Open Sans Variable (3 axes) | Steve Matteson

The font that resonated with me the most was Raleway. This perfectly fit the aesthetic of the program, being modern while also simple and readable

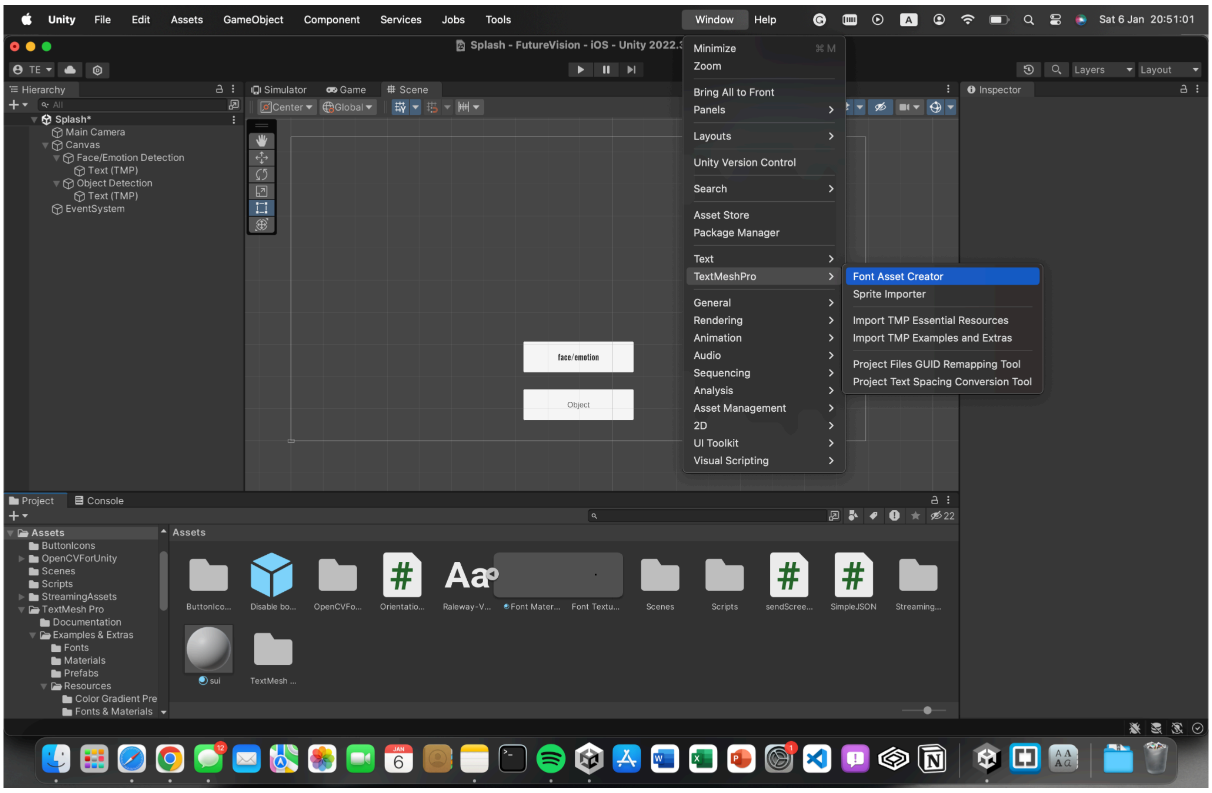


Raleway

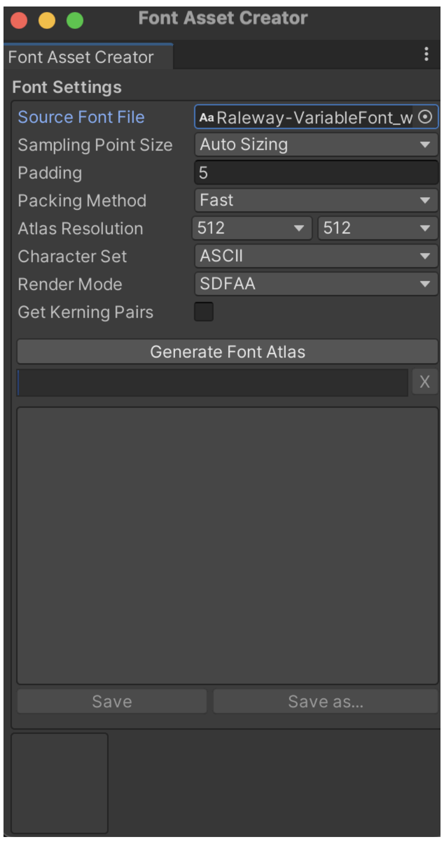
Designed by Matt McInerney, Pablo Impallari, Rodrigo Fuenzalida

Whereas disregard and contempt for human rights have resulted

For the buttons on the home screen, I ensured that the text field was set to a TextMeshPro object. This is because TMP is a text rendering asset for Unity that provides enhanced text rendering and formatting capabilities compared to Unity's built-in Text component. TextMeshPro supports custom fonts, including dynamic font sizing and improved text rendering quality and is designed to be highly optimised for performance, even with a large amount of text on the screen. It uses efficient text rendering techniques, making it suitable for UI elements and in-game text.

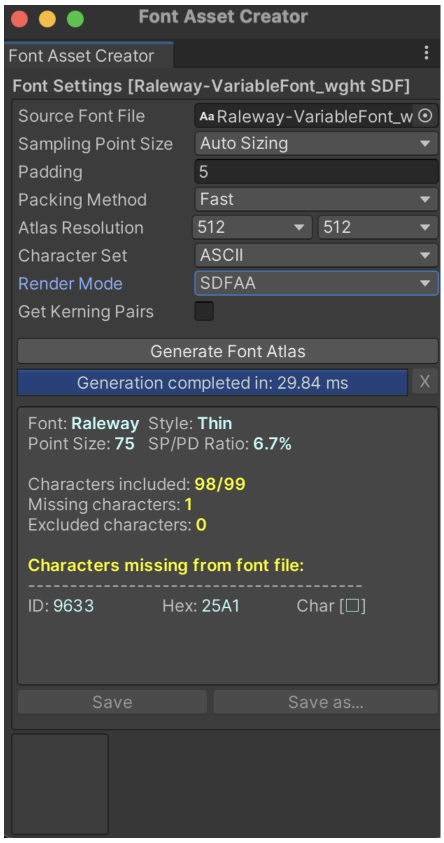

Window
TextMeshPro
Font Asset Creator
Sprite Importer
Import TMP Essential Resources
Import TMP Examples and Extras
Project Files GUID Remapping Tool
Project Text Spacing Conversion Tool

In order to use the selected font with TMP, I must first use the font asset creator. This is used to generate a font asset from a TrueType Font (TTF) or OpenType Font (OTF) file. This process involves importing the font, configuring settings such as font size and style, and creating an asset that can be used for rendering text in Unity.


Font Asset Creator

Font Settings

Source Font File: Raleway-VariableFont_w
Sampling Point Size: Auto Sizing
Padding: 5
Packing Method: Fast
Atlas Resolution: 512 512
Character Set: ASCII
Render Mode: SDFAA
Get Kerning Pairs


Font Asset Creator

Font Settings [Raleway-VariableFont_wght SDF]

Source Font File: Raleway-VariableFont_w
Sampling Point Size: Auto Sizing
Padding: 5
Packing Method: Fast
Atlas Resolution: 512 512
Character Set: ASCII
Render Mode: SDFAA
Get Kerning Pairs

Generation completed in: 29.84 ms

Font: Raleway Style: Thin
Point Size: 75 SP/PD Ratio: 6.7%

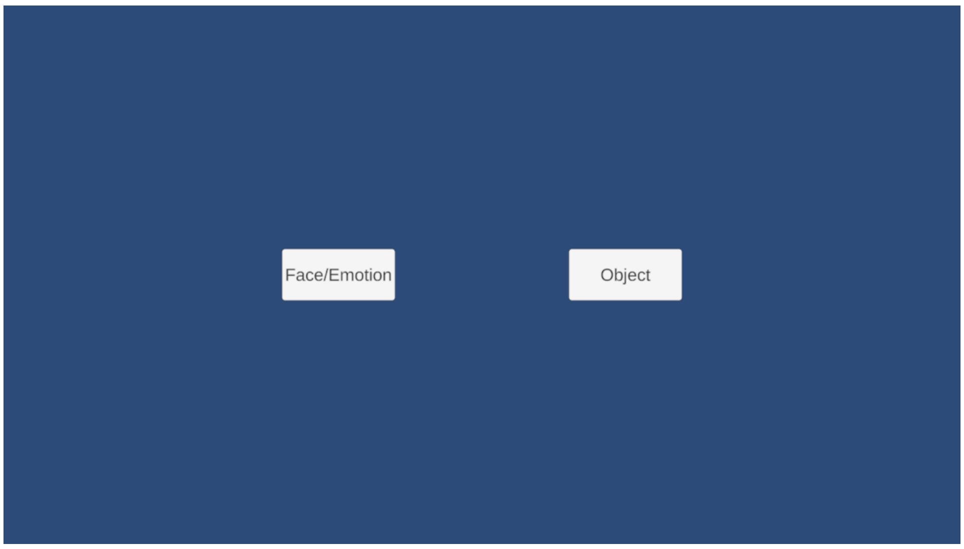
Characters included: 98/99
Missing characters: 1
Excluded characters: 0

Characters missing from font file:
```
-----------------------------------------
ID: 9633        Hex: 25A1        Char [□]
```

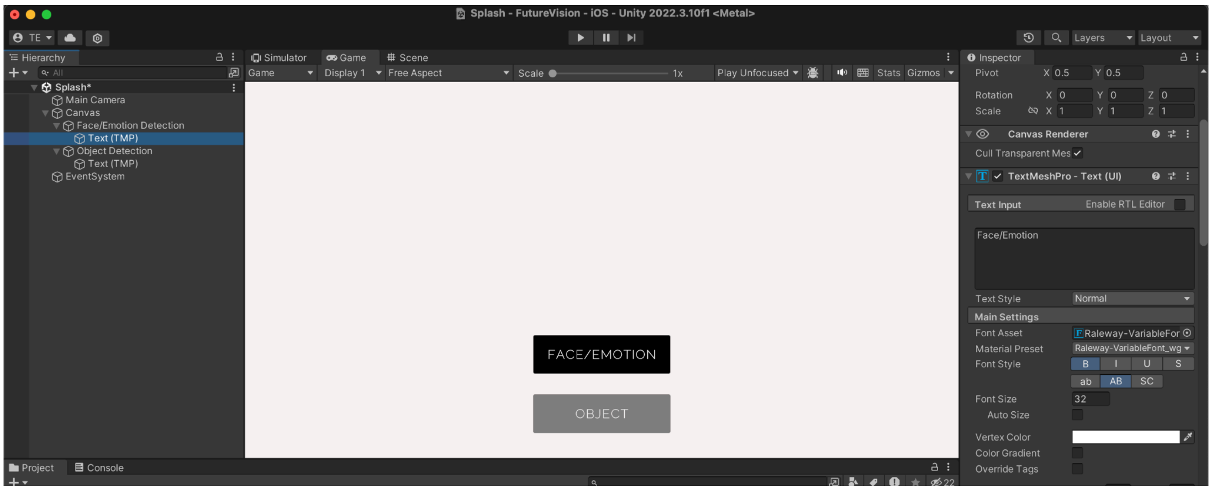
I imported the downloaded .ttf font Raleway and left all the values as default, which then created the unity font asset to be used in TMP.

I applied this font to the button texts and recoloured the background to create a more aesthetic look to the landing page.

Before:



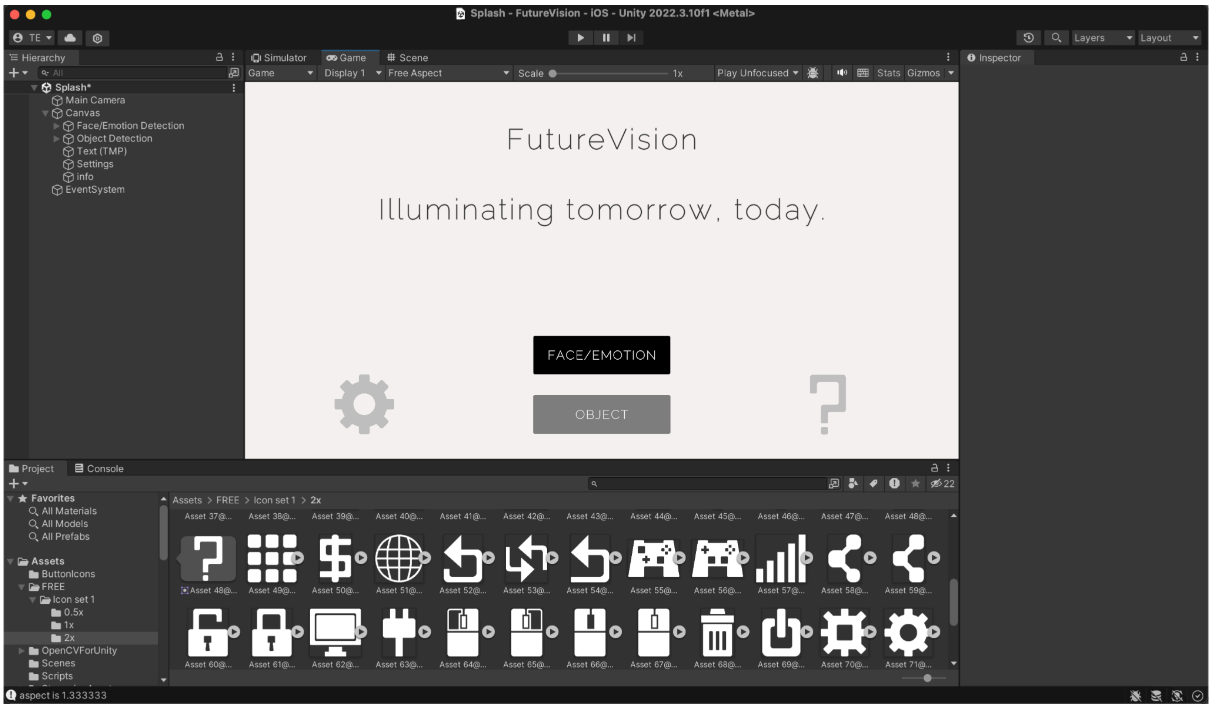
After:



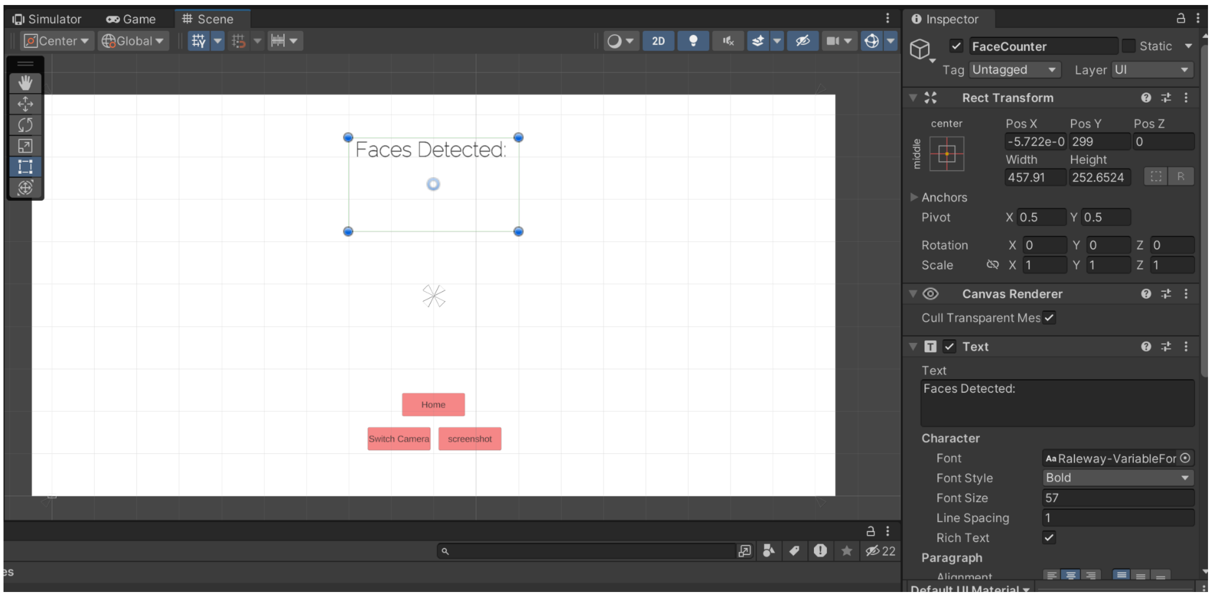
Text Input: Face/Emotion
Text Style: Normal
Font Asset: Raleway-VariableFor
Material Preset: Raleway-VariableFont_wg
Font Size: 32

I aimed to use a neutral colour pallet of white, black and different shades of grey.

There seems to be a lot of empty space mainly towards the middle of the screen, which I was hoping to fill with some welcome text/logo, using the same font and style.



I also needed to create a button for the settings page and the information/about page so I created 2 standard buttons, deleted the text and set the image sprite to one of the suitable images from the icon pack. I also created some filler text for the top of the page, with the name of the application, along with a short slogan.



FaceCounter Static
Tag Untagged Layer UI

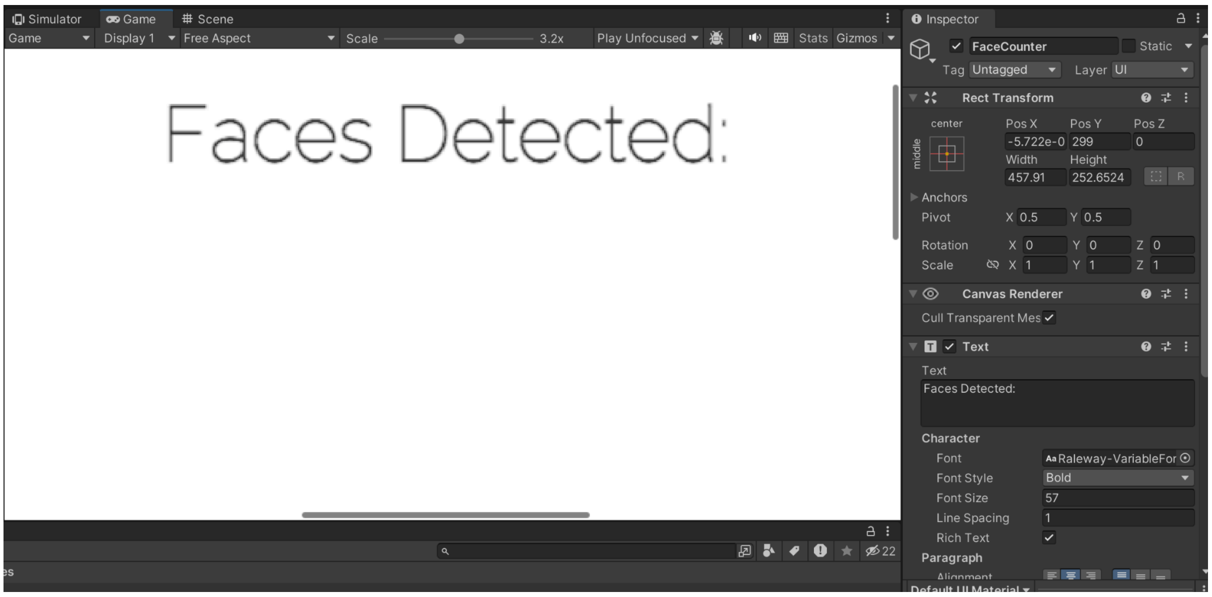
Rect Transform
center
Pos X -5.722e-0 Pos Y 299 Pos Z 0
Width 457.91 Height 252.6524
Anchors
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Canvas Renderer
Cull Transparent Mes ✓

Text
Text
Faces Detected:

Character
Font Raleway-VariableFor
Font Style Bold
Font Size 57
Line Spacing 1
Rich Text ✓
Paragraph
Alignment

Upon updating the UI for the emotionDetection scene, I started out by changing the colour and font of the textObject, however, I noticed that it is not as sharp as the home screen, nor does it scale well, as shown below:



FaceCounter Static
Tag Untagged Layer UI

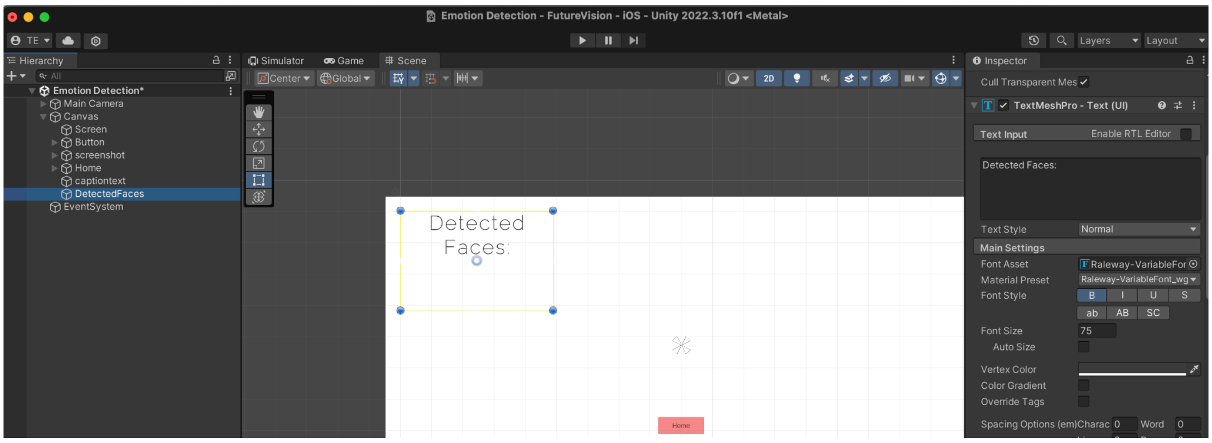
Rect Transform
center
Pos X -5.722e-0 Pos Y 299 Pos Z 0
Width 457.91 Height 252.6524
Anchors
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Canvas Renderer
Cull Transparent Mes ✓

Text
Text
Faces Detected:

Character
Font Raleway-VariableFor
Font Style Bold
Font Size 57
Line Spacing 1
Rich Text ✓
Paragraph
Alignment

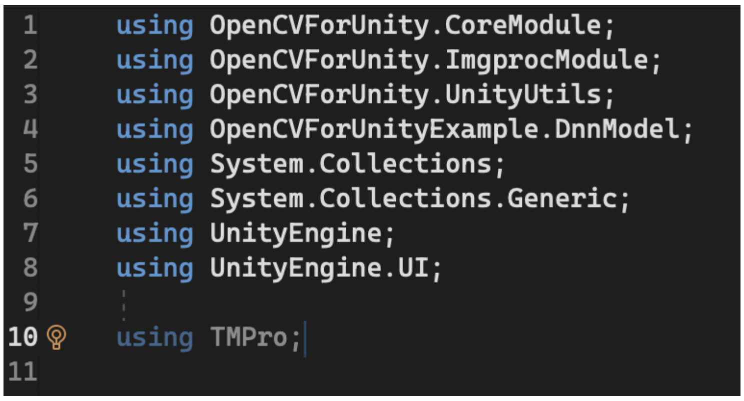
This is due to the fact that the text being edited is a standard Unity text object, rather than a TextMeshPro. TextMeshPro uses a more advanced text rendering engine compared to the default Unity Text component, however a script change will need to be made as updating the standard text object is different to a TextMeshPro object.



```
Cull Transparent Mes ✓
TextMeshPro - Text (UI)
Text Input                 Enable RTL Editor
Detected Faces:
Text Style                 Normal
Main Settings
Font Asset                 Raleway-VariableFor
Material Preset            Raleway-VariableFont_wg
Font Style                 B  I  U  S
                           ab AB SC
Font Size                  75
  Auto Size
Vertex Color
Color Gradient
Override Tags
Spacing Options (em)Charac 0   Word 0
```

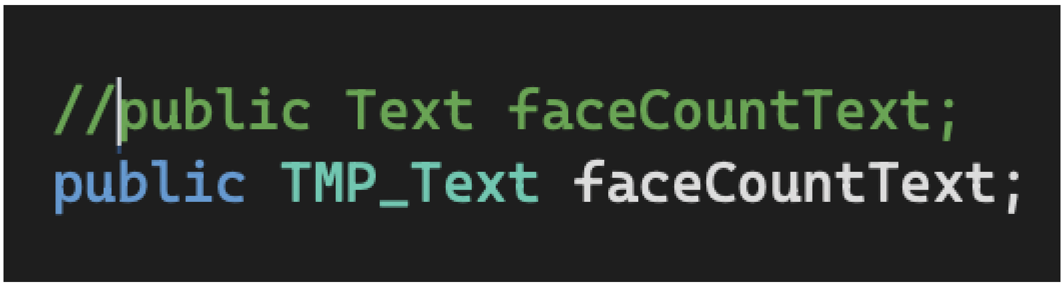
After updating to TextMeshPro, the text scaling and resolution is corrected.

EmptionDetection.cs



```
using OpenCVForUnity.CoreModule;
using OpenCVForUnity.ImgprocModule;
using OpenCVForUnity.UnityUtils;
using OpenCVForUnityExample.DnnModel;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

using TMPro;
```

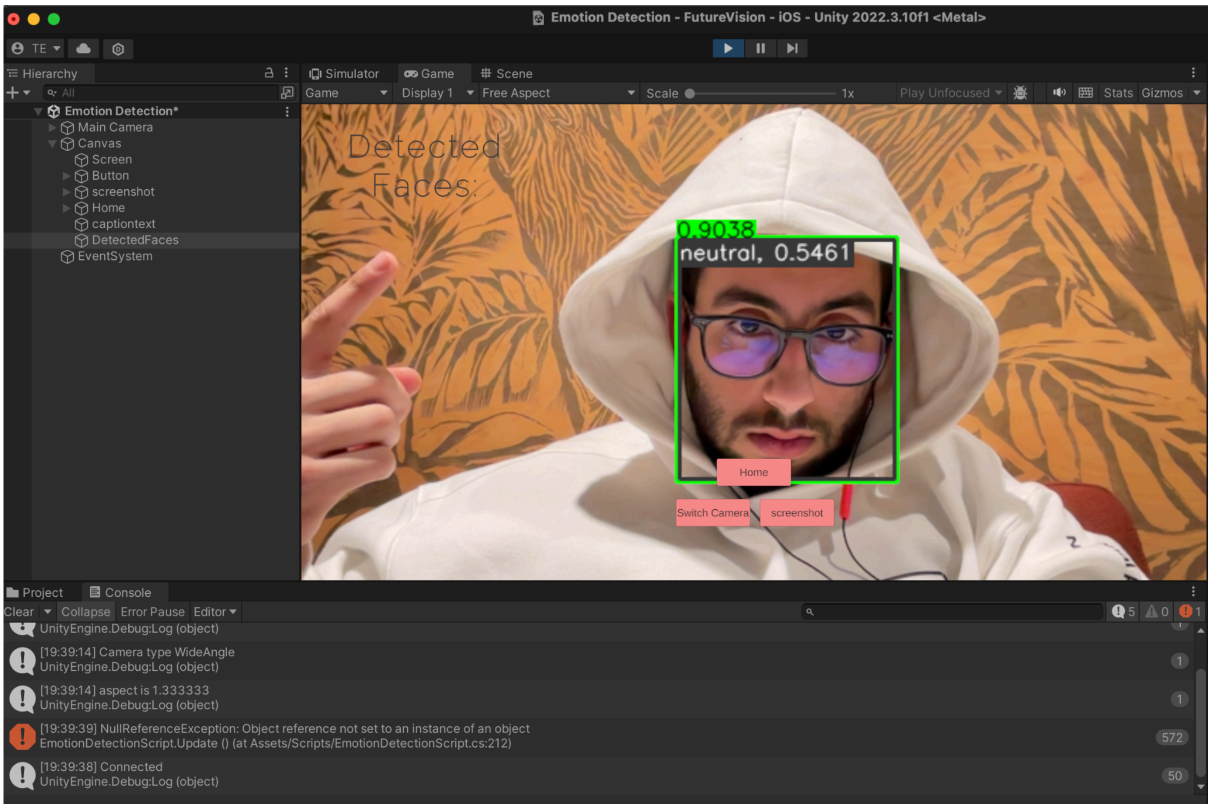
In order to access TextMeshPro modules, I will need to import the TMPro library in C#



```
//public Text faceCountText;
public TMP_Text faceCountText;
```

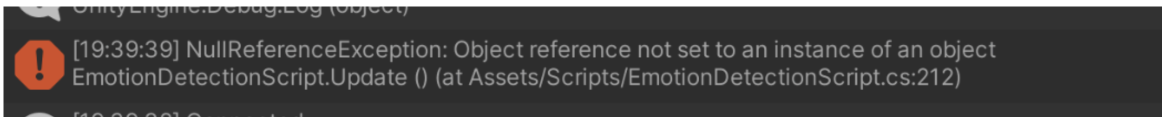
I have also changed the faceCountText variable from a standard unity UI text object to a TMP_Text variable type.

After testing the changes, I came across an error where the TextMeshPro object was not updating:



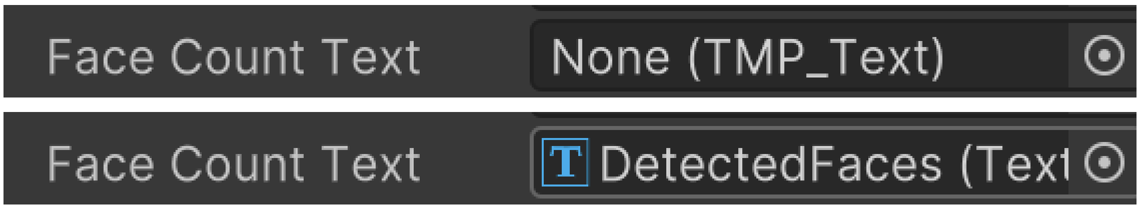
```
UnityEngine.Debug:Log (object)
[19:39:14] Camera type WideAngle
UnityEngine.Debug:Log (object)                                                         1
[19:39:14] aspect is 1.333333
UnityEngine.Debug:Log (object)                                                         1
[19:39:39] NullReferenceException: Object reference not set to an instance of an object
EmotionDetectionScript.Update () (at Assets/Scripts/EmotionDetectionScript.cs:212)   572
[19:39:38] Connected
UnityEngine.Debug:Log (object)                                                        50
```

This was due to the error popping up in the console:



```
[19:39:39] NullReferenceException: Object reference not set to an instance of an object
EmotionDetectionScript.Update () (at Assets/Scripts/EmotionDetectionScript.cs:212)
```

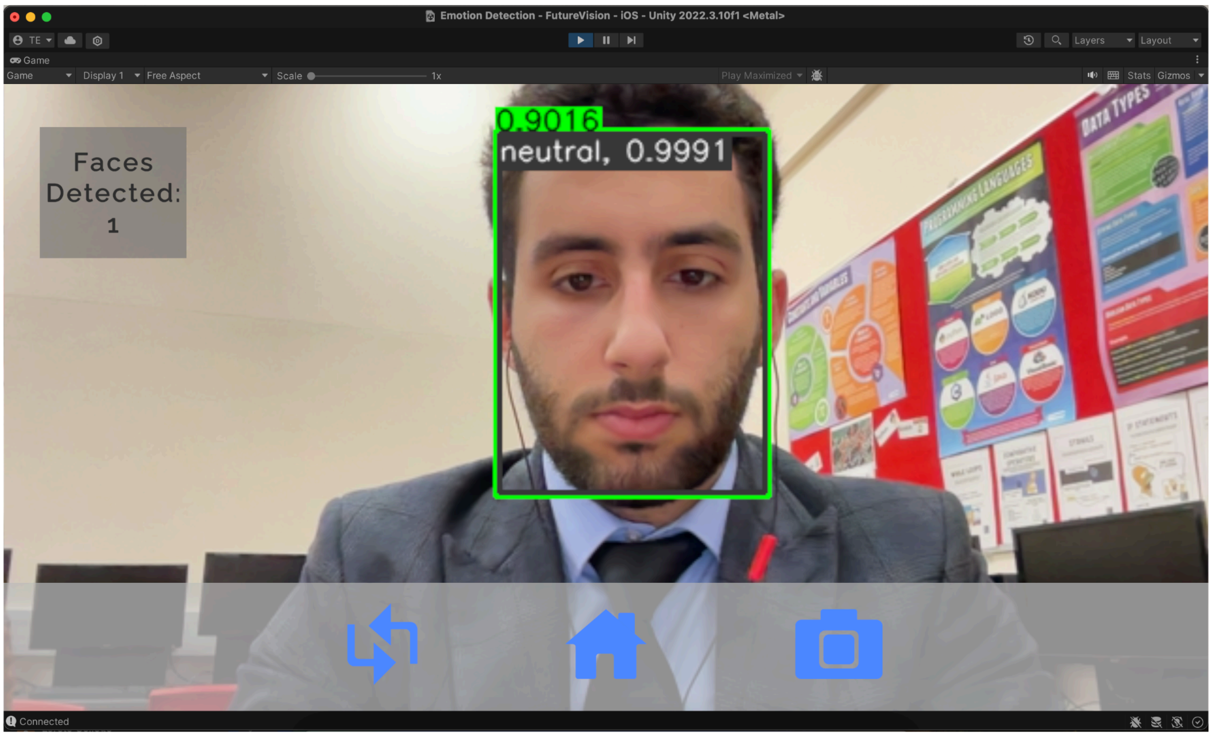
After working on this project for a few months, it became clear to me immediately that this error was due to a public variable not being referenced in the inspector. This was immediately changed.

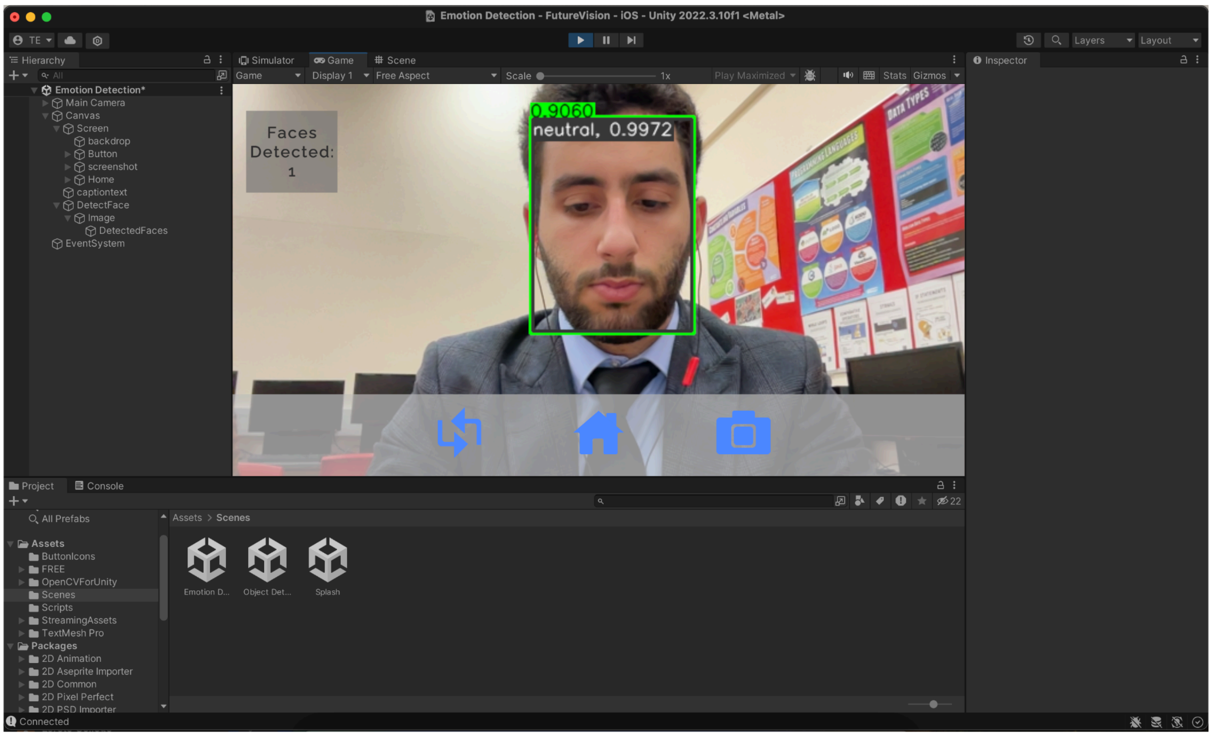


Face Count Text None (TMP_Text)

Face Count Text DetectedFaces (Text

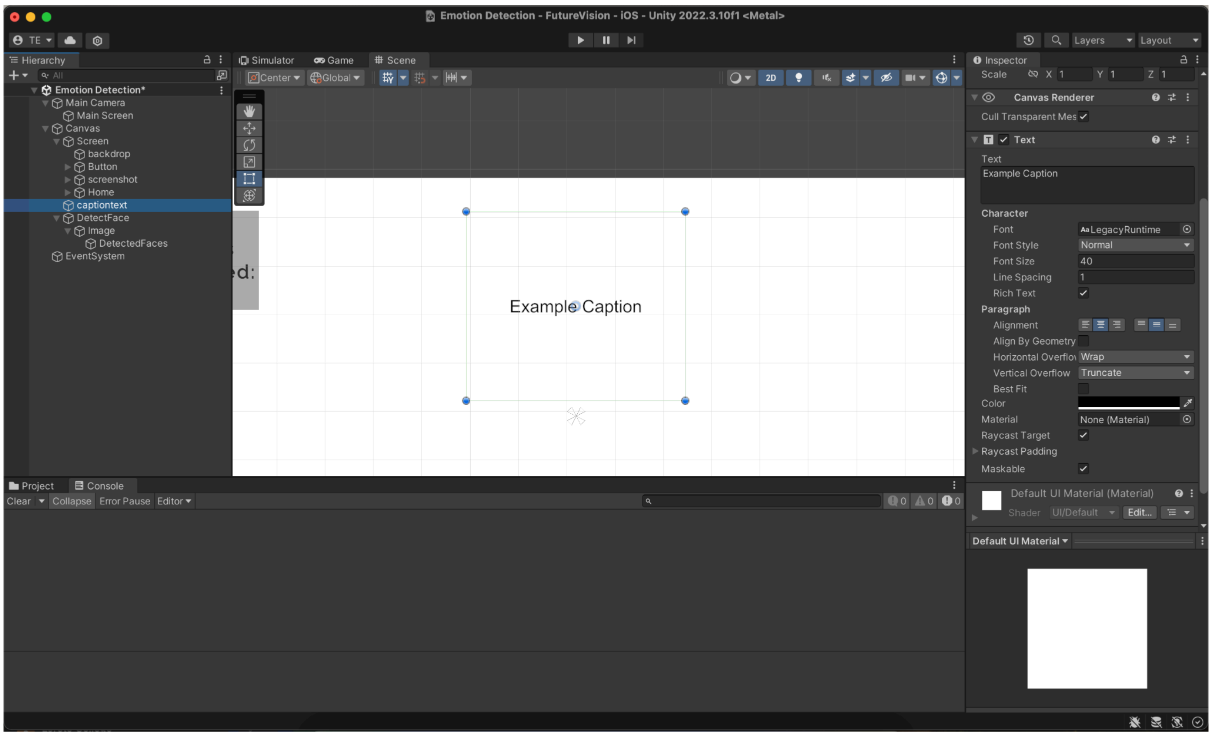
I continued to revamp the entire UI for the emotion detection scene by adding icons instead of text buttons, while also increasing the readability of the text on the screen by adding a grey background box to increase the contrast of the text and buttons as shown below





I may change the colours and sprites of the buttons later on, along with the UI background, but for now, this has helped increase the useability and readability of text and buttons.

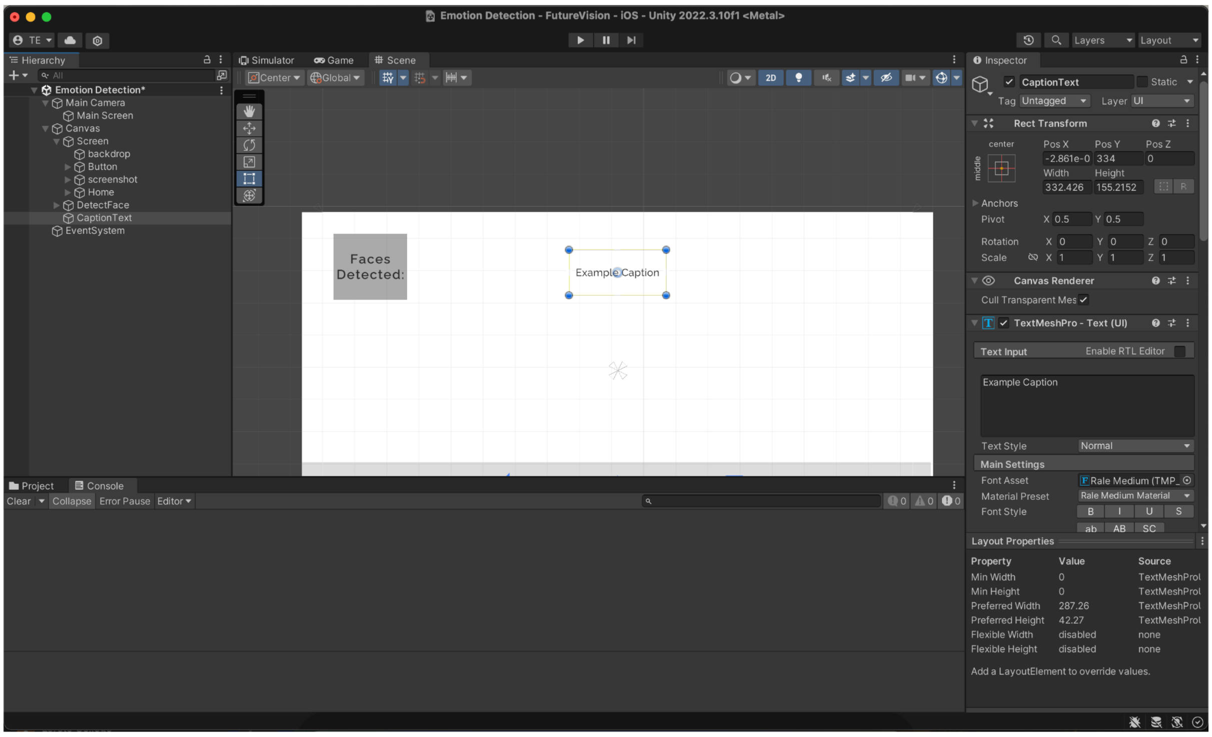
I am also going to do this for the captioned text. My aim is to create a text box with a transparent background, similar to the “Faces detected” text that appears when a caption appears on screen. I also want to implement a feature to hide the caption if the user wants to look at the screen. This will be done through another button, which toggles the captionText gameObject.


Emotion Detection - FutureVision - iOS - Unity 2022.3.10f1 <Metal>

```
Hierarchy
Emotion Detection*
  Main Camera
    Main Screen
  Canvas
    Screen
      backdrop
      Button
      screenshot
      Home
    captiontext
    DetectFace
      Image
        DetectedFaces
  EventSystem
```

```
Inspector
Scale  X 1  Y 1  Z 1
Canvas Renderer
  Cull Transparent Mes ✓
Text
  Text
    Example Caption
  Character
    Font                 LegacyRuntime
    Font Style           Normal
    Font Size            40
    Line Spacing         1
    Rich Text            ✓
  Paragraph
    Alignment
    Align By Geometry
    Horizontal Overflo   Wrap
    Vertical Overflow    Truncate
    Best Fit
  Color
  Material               None (Material)
  Raycast Target         ✓
  Raycast Padding
  Maskable               ✓
Default UI Material (Material)
  Shader  UI/Default
```

This is the current position and style of the captiontext gameObject. It currently uses a standard unity ‘text’ component - I will need to upgrade this to TMP and update the corresponding code also.

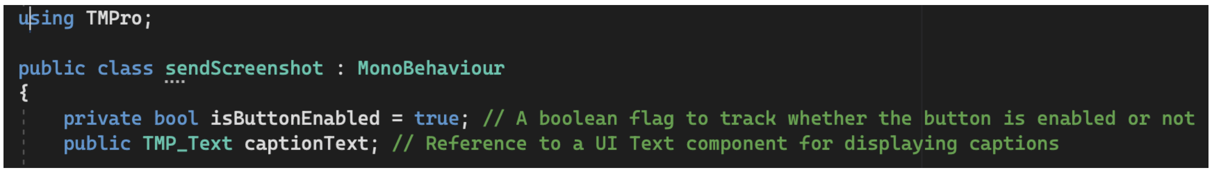

Emotion Detection - FutureVision - iOS - Unity 2022.3.10f1 <Metal>

```
Hierarchy
Emotion Detection*
  Main Camera
    Main Screen
  Canvas
    Screen
      backdrop
      Button
      screenshot
      Home
    DetectFace
    CaptionText
  EventSystem
```

```
Inspector
CaptionText   Static
Tag Untagged   Layer UI
Rect Transform
  center / middle
  Pos X  -2.861e-0   Pos Y  334   Pos Z  0
  Width  332.426     Height 155.2152
  Anchors
  Pivot     X 0.5  Y 0.5
  Rotation  X 0    Y 0    Z 0
  Scale     X 1    Y 1    Z 1
Canvas Renderer
  Cull Transparent Mes ✓
TextMeshPro - Text (UI)
  Text Input          Enable RTL Editor
    Example Caption
  Text Style          Normal
  Main Settings
    Font Asset        Rale Medium (TMP_
    Material Preset   Rale Medium Material
    Font Style        B  I  U  S  ab  AB  SC
Layout Properties
Property           Value      Source
Min Width          0          TextMeshProl
Min Height         0          TextMeshProl
Preferred Width    287.26     TextMeshProl
Preferred Height   42.27      TextMeshProl
Flexible Width     disabled   none
Flexible Height    disabled   none
Add a LayoutElement to override values.
```

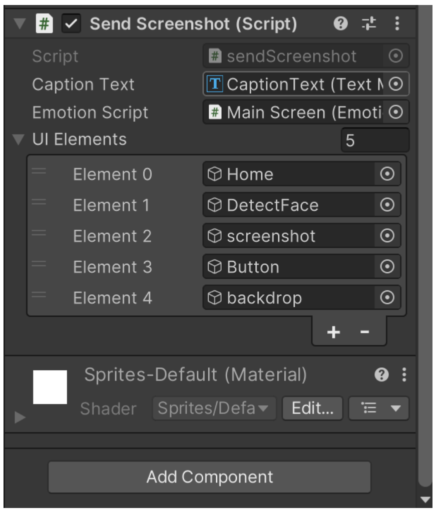
I updated the caption text object to utilise TextMeshPro and applied the chosen font also.

sendScreenshot.cs



```
using TMPro;

public class sendScreenshot : MonoBehaviour
{
    private bool isButtonEnabled = true; // A boolean flag to track whether the button is enabled or not
    public TMP_Text captionText; // Reference to a UI Text component for displaying captions
```

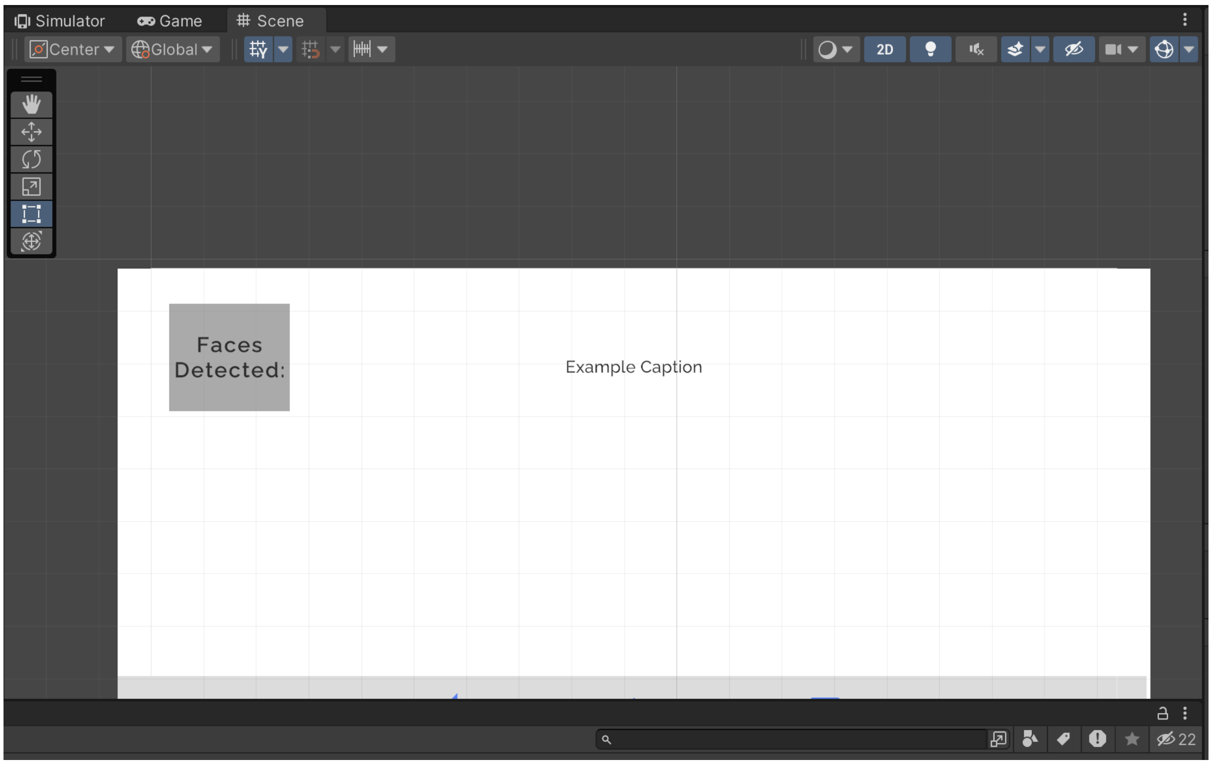
I updated the script to use the TMPro library and changed the public ‘Text’ to a public TMP_Text in order to change and edit the contents of the object component.



Send Screenshot (Script)
Script: sendScreenshot
Caption Text: CaptionText (Text M
Emotion Script: Main Screen (Emoti
UI Elements: 5
Element 0: Home
Element 1: DetectFace
Element 2: screenshot
Element 3: Button
Element 4: backdrop

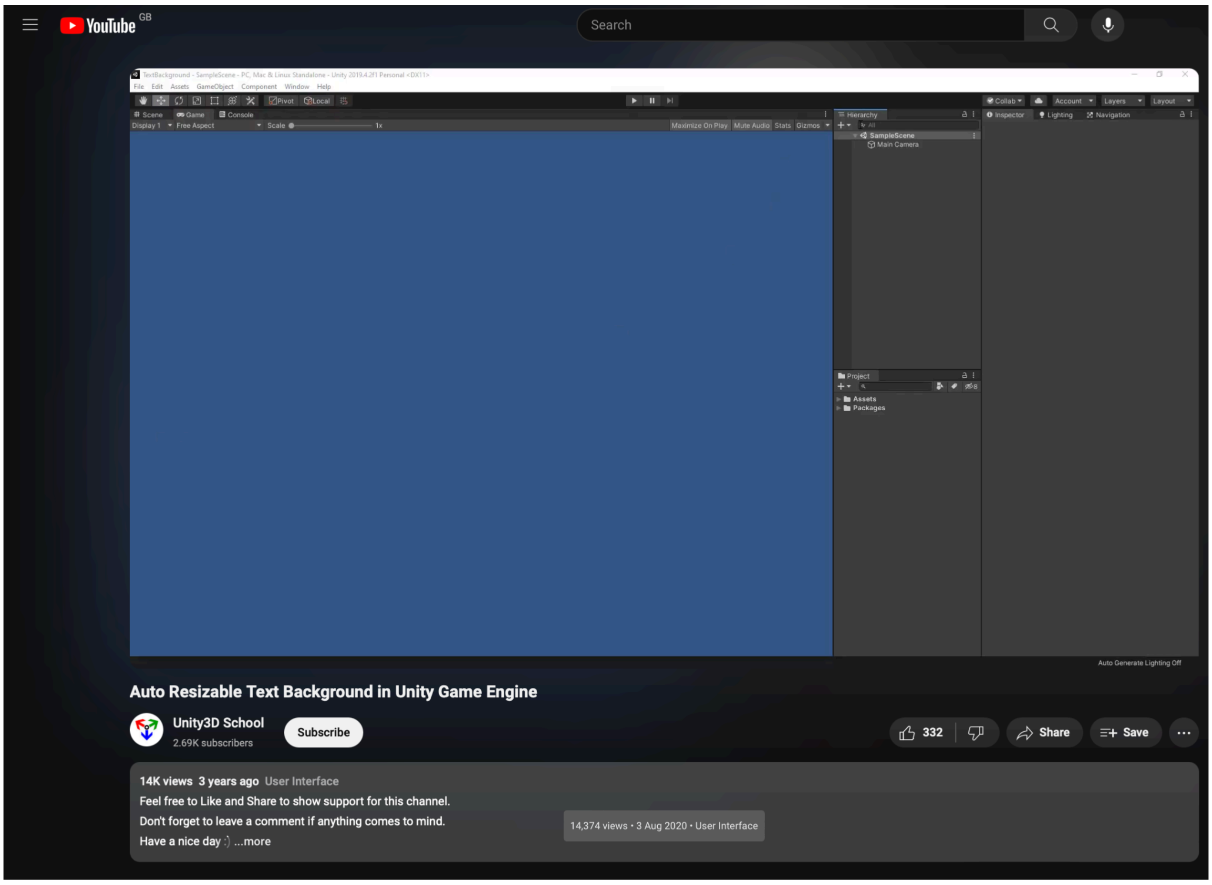
Sprites-Default (Material)
Shader: Sprites/Defa

I did not make the same mistake as last time and ensured that the caption text public variable was assigned before testing.

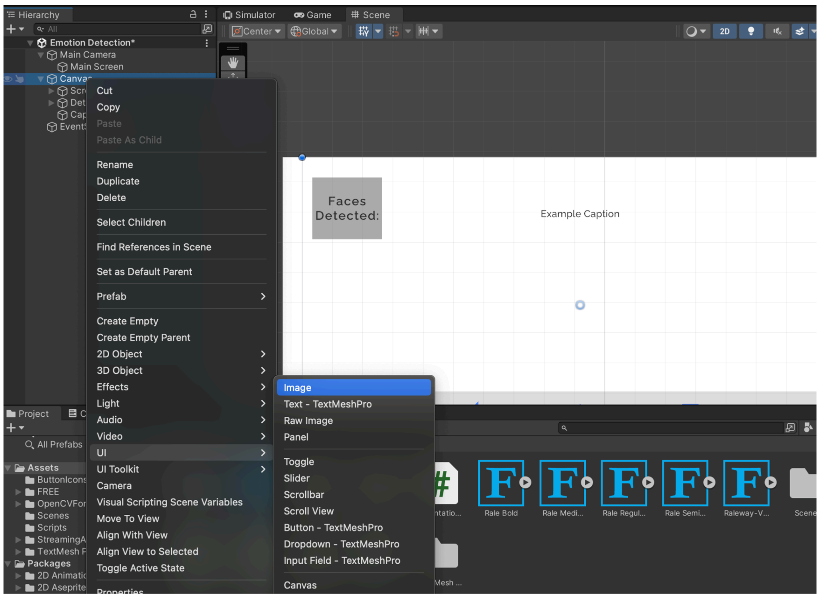


However, I still want to add a background to increase the text contrast as it will be quite difficult to read without. In order to do this, I have been adding a UI image object behind the text, however, for the captions, I also want the text background to be dynamically sized so that the size will change depending on the length of the outputted caption.

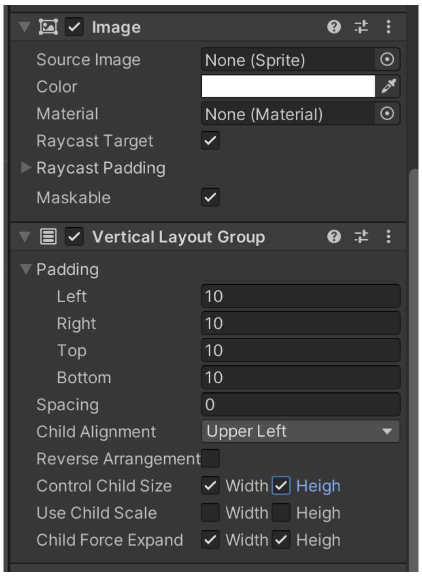
In order to do this, I will be following this guide from youtube.



14K views 3 years ago User Interface
Feel free to Like and Share to show support for this channel.
Don't forget to leave a comment if anything comes to mind.
Have a nice day :) ...more



The first thing to do for this to work, was create a UI image into the scene. This will be the text's background.



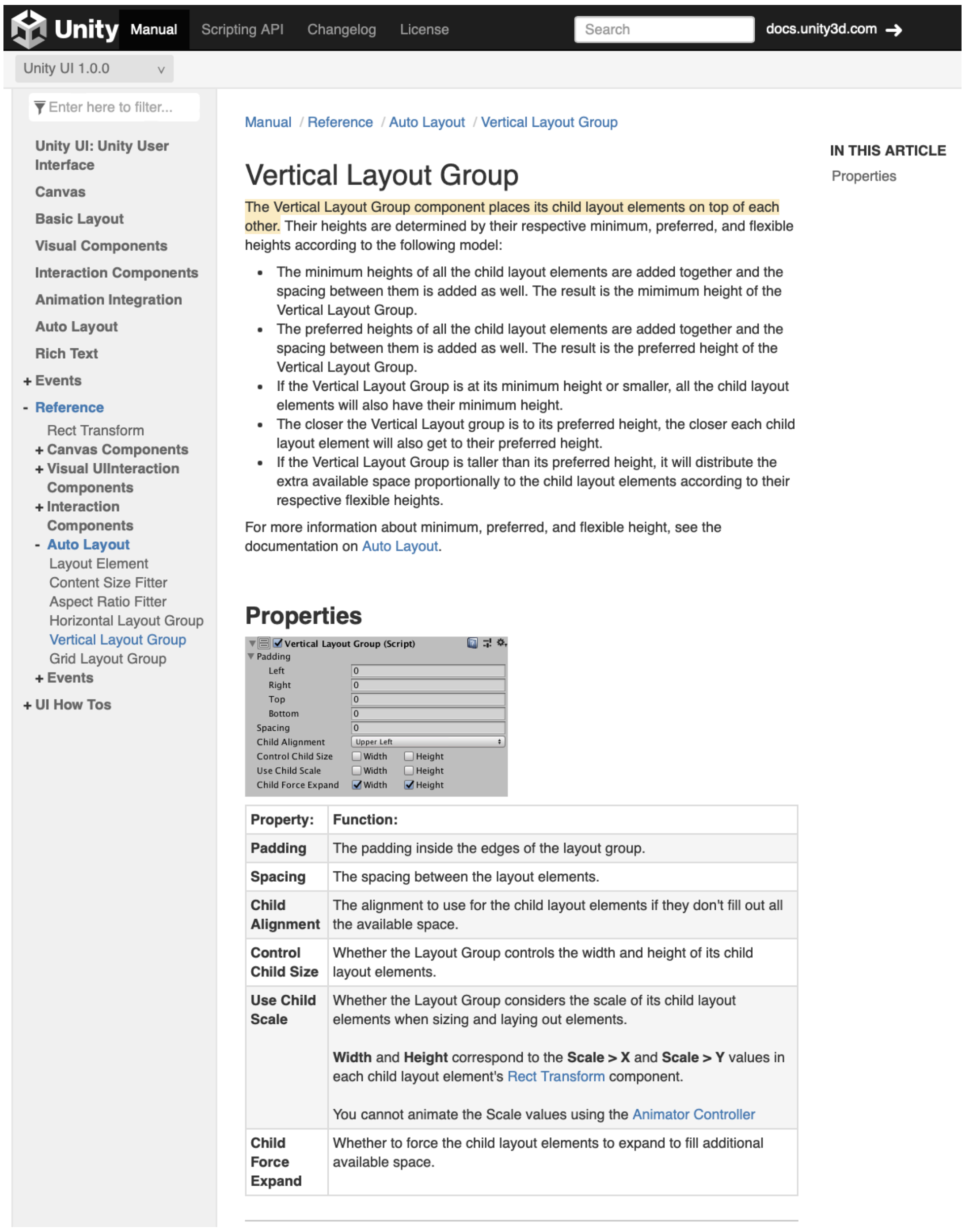
Image

Source Image	None (Sprite)
Color	
Material	None (Material)
Raycast Target	✓
Raycast Padding	
Maskable	✓

Vertical Layout Group

Padding	
Left	10
Right	10
Top	10
Bottom	10
Spacing	0
Child Alignment	Upper Left
Reverse Arrangement	
Control Child Size	✓ Width ✓ Heigh
Use Child Scale	Width Heigh
Child Force Expand	✓ Width ✓ Heigh

I then added a vertical layout group component, this will allow the text and any child objects to control the sizing of the image object. The padding is set to 10 all around - this is so that the edge of the image is not touching the text, and there remains a gap. This can be altered later to preference.


Unity Manual Scripting API Changelog License Search docs.unity3d.com

Unity UI 1.0.0

Enter here to filter...

Unity UI: Unity User Interface
Canvas
Basic Layout
Visual Components
Interaction Components
Animation Integration
Auto Layout
Rich Text
+ Events
- Reference
Rect Transform
+ Canvas Components
+ Visual UIInteraction Components
+ Interaction Components
- Auto Layout
Layout Element
Content Size Fitter
Aspect Ratio Fitter
Horizontal Layout Group
Vertical Layout Group
Grid Layout Group
+ Events
+ UI How Tos

Manual / Reference / Auto Layout / Vertical Layout Group

IN THIS ARTICLE
Properties

Vertical Layout Group

The Vertical Layout Group component places its child layout elements on top of each other. Their heights are determined by their respective minimum, preferred, and flexible heights according to the following model:

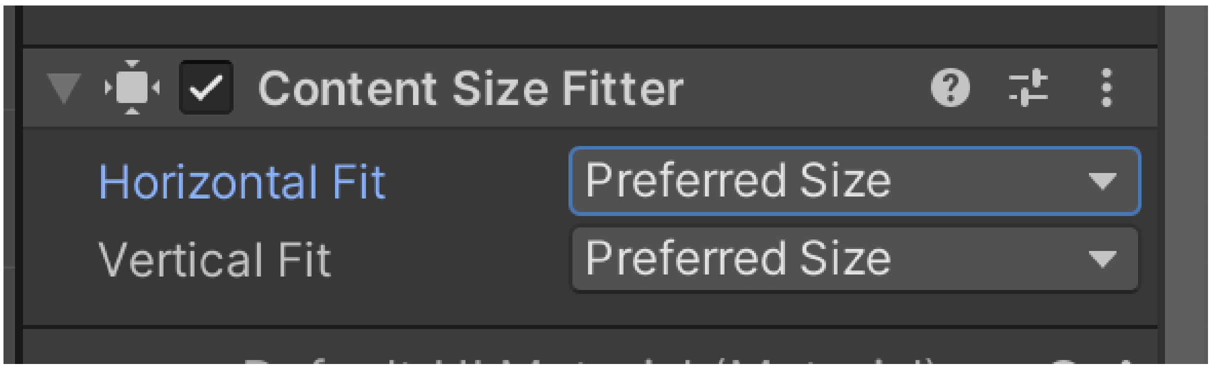
- The minimum heights of all the child layout elements are added together and the spacing between them is added as well. The result is the mimimum height of the Vertical Layout Group.
- The preferred heights of all the child layout elements are added together and the spacing between them is added as well. The result is the preferred height of the Vertical Layout Group.
- If the Vertical Layout Group is at its minimum height or smaller, all the child layout elements will also have their minimum height.
- The closer the Vertical Layout group is to its preferred height, the closer each child layout element will also get to their preferred height.
- If the Vertical Layout Group is taller than its preferred height, it will distribute the extra available space proportionally to the child layout elements according to their respective flexible heights.

For more information about minimum, preferred, and flexible height, see the documentation on Auto Layout.

Properties

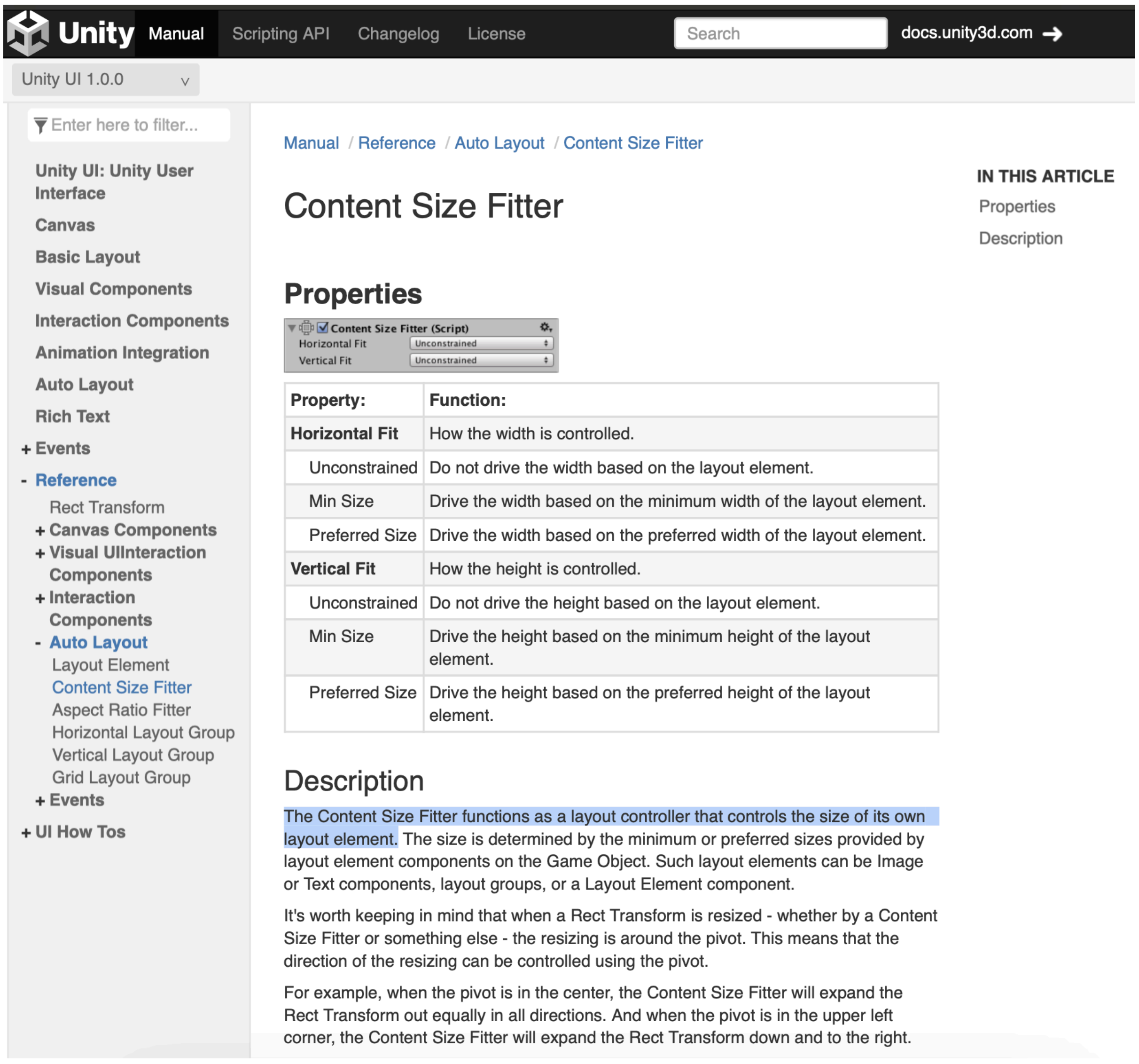
Vertical Layout Group (Script)
Padding
Left 0
Right 0
Top 0
Bottom 0
Spacing 0
Child Alignment Upper Left
Control Child Size ☐ Width ☐ Height
Use Child Scale ☐ Width ☐ Height
Child Force Expand ✓ Width ✓ Height

Property:	Function:
Padding	The padding inside the edges of the layout group.
Spacing	The spacing between the layout elements.
Child Alignment	The alignment to use for the child layout elements if they don't fill out all the available space.
Control Child Size	Whether the Layout Group controls the width and height of its child layout elements.
Use Child Scale	Whether the Layout Group considers the scale of its child layout elements when sizing and laying out elements. **Width** and **Height** correspond to the **Scale > X** and **Scale > Y** values in each child layout element's Rect Transform component. You cannot animate the Scale values using the Animator Controller
Child Force Expand	Whether to force the child layout elements to expand to fill additional available space.


Content Size Fitter

Horizontal Fit Preferred Size
Vertical Fit Preferred Size

I then added a Content Size Fitter to the image gameObject, the description of its functionality is given below, provided by Unity documentation:


Unity Manual Scripting API Changelog License Search docs.unity3d.com

Unity UI 1.0.0

Enter here to filter...

Unity UI: Unity User Interface
Canvas
Basic Layout
Visual Components
Interaction Components
Animation Integration
Auto Layout
Rich Text
+ Events
- Reference
Rect Transform
+ Canvas Components
+ Visual UIInteraction Components
+ Interaction Components
- Auto Layout
Layout Element
Content Size Fitter
Aspect Ratio Fitter
Horizontal Layout Group
Vertical Layout Group
Grid Layout Group
+ Events
+ UI How Tos

Manual / Reference / Auto Layout / Content Size Fitter

IN THIS ARTICLE
Properties
Description

Content Size Fitter

Properties

Content Size Fitter (Script)
Horizontal Fit Unconstrained
Vertical Fit Unconstrained

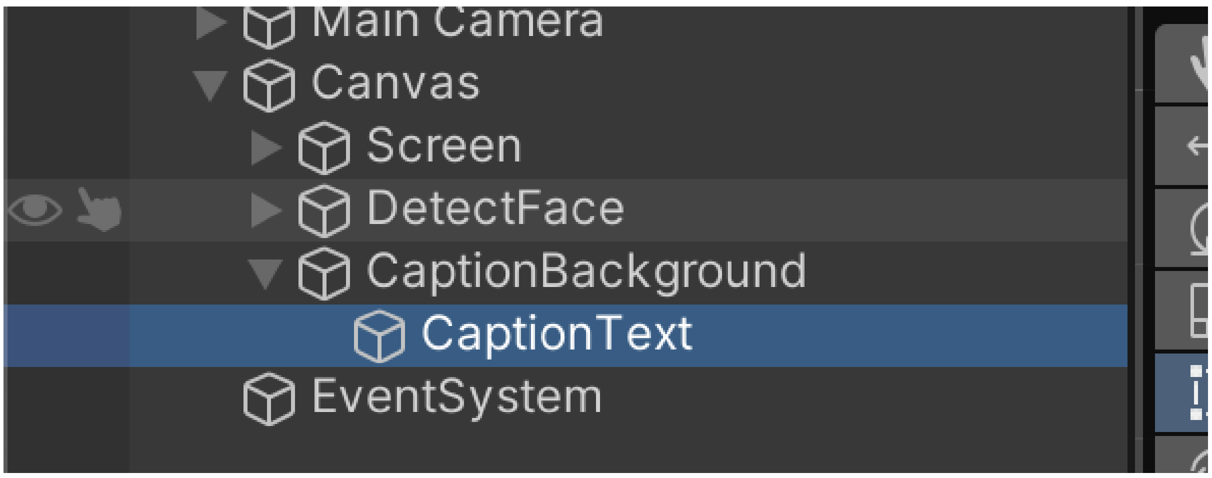
Property:	Function:
Horizontal Fit	How the width is controlled.
Unconstrained	Do not drive the width based on the layout element.
Min Size	Drive the width based on the minimum width of the layout element.
Preferred Size	Drive the width based on the preferred width of the layout element.
Vertical Fit	How the height is controlled.
Unconstrained	Do not drive the height based on the layout element.
Min Size	Drive the height based on the minimum height of the layout element.
Preferred Size	Drive the height based on the preferred height of the layout element.

Description

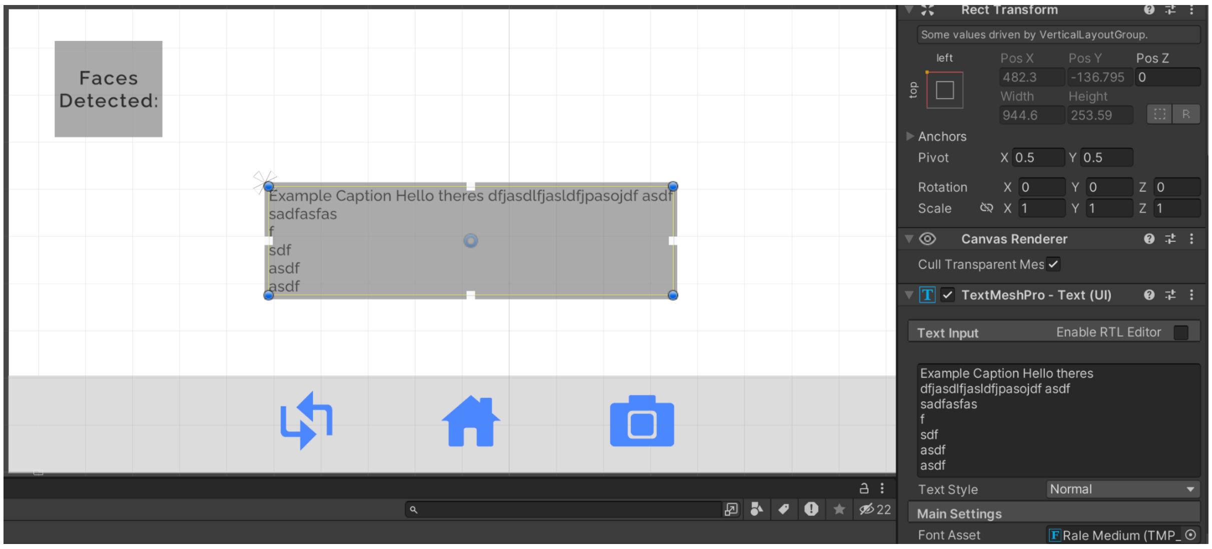
The Content Size Fitter functions as a layout controller that controls the size of its own layout element. The size is determined by the minimum or preferred sizes provided by layout element components on the Game Object. Such layout elements can be Image or Text components, layout groups, or a Layout Element component.

It's worth keeping in mind that when a Rect Transform is resized - whether by a Content Size Fitter or something else - the resizing is around the pivot. This means that the direction of the resizing can be controlled using the pivot.

For example, when the pivot is in the center, the Content Size Fitter will expand the Rect Transform out equally in all directions. And when the pivot is in the upper left corner, the Content Size Fitter will expand the Rect Transform down and to the right.



Afterwards, I add the TMP text object underneath and it is complete.



Example Caption Hello theres dfjasdlfjasldfjpasojdf asdf
sadfasfas
f
sdf
asdf
asdf

Rect Transform
Some values driven by VerticalLayoutGroup.

	Pos X	Pos Y	Pos Z
left / top	482.3	-136.795	0
	Width	Height	
	944.6	253.59	

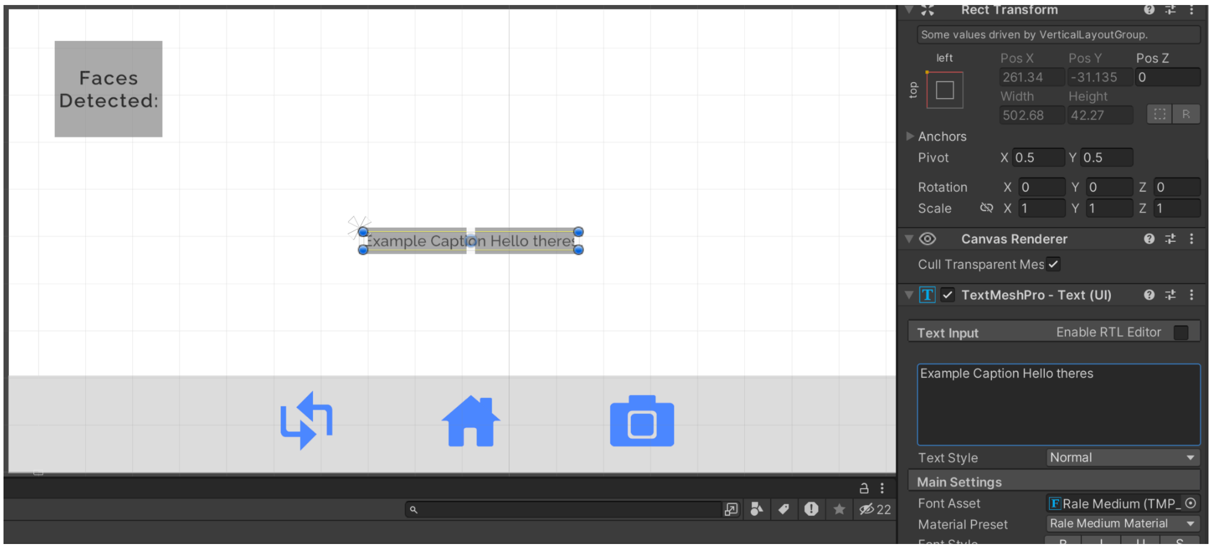
Anchors
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Canvas Renderer
Cull Transparent Mes ✓

TextMeshPro - Text (UI)
Text Input — Enable RTL Editor

```
Example Caption Hello theres
dfjasdlfjasldfjpasojdf asdf
sadfasfas
f
sdf
asdf
asdf
```

Text Style: Normal
Main Settings
Font Asset: Rale Medium (TMP_



Example Caption Hello theres

Rect Transform
Some values driven by VerticalLayoutGroup.

	Pos X	Pos Y	Pos Z
left / top	261.34	-31.135	0
	Width	Height	
	502.68	42.27	

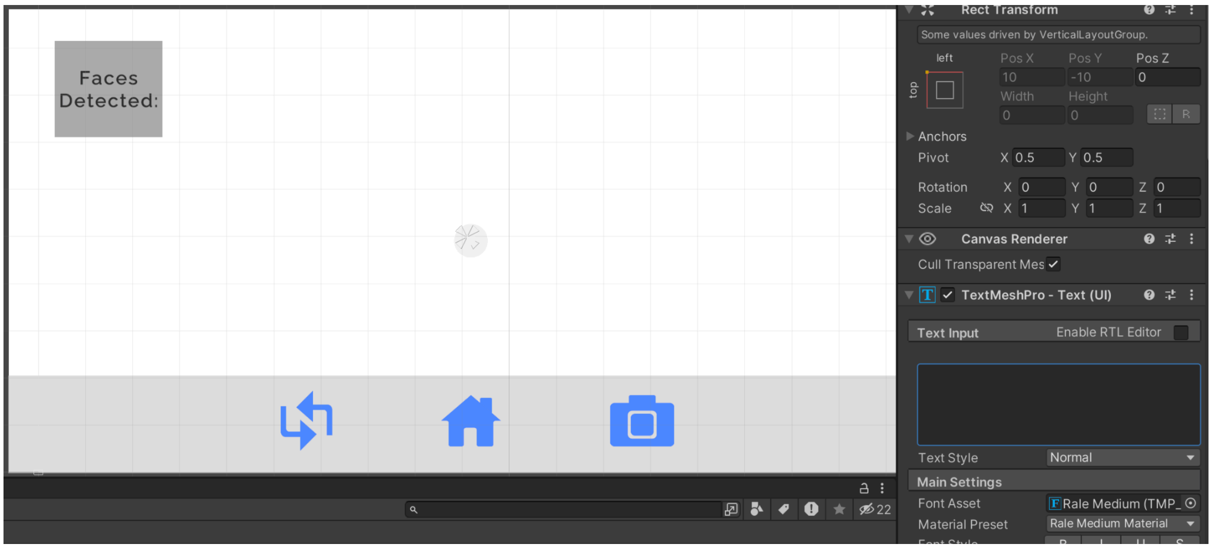
Anchors
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Canvas Renderer
Cull Transparent Mes ✓

TextMeshPro - Text (UI)
Text Input — Enable RTL Editor

```
Example Caption Hello theres
```

Text Style: Normal
Main Settings
Font Asset: Rale Medium (TMP_
Material Preset: Rale Medium Material



Rect Transform
Some values driven by VerticalLayoutGroup.

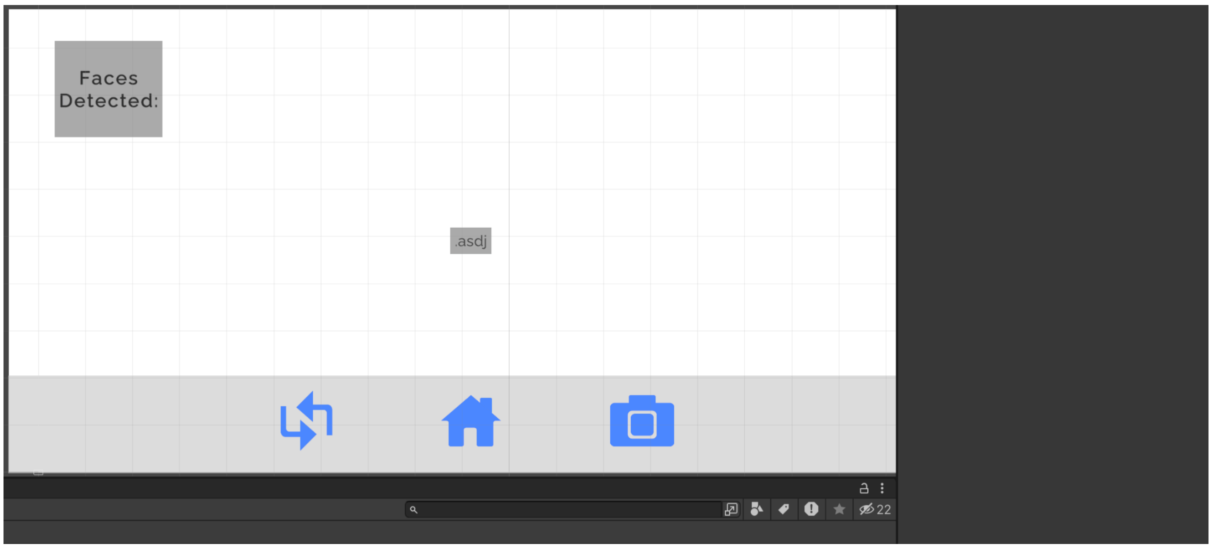
	Pos X	Pos Y	Pos Z
left / top	10	-10	0
	Width	Height	
	0	0	

Anchors
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Canvas Renderer
Cull Transparent Mes ✓

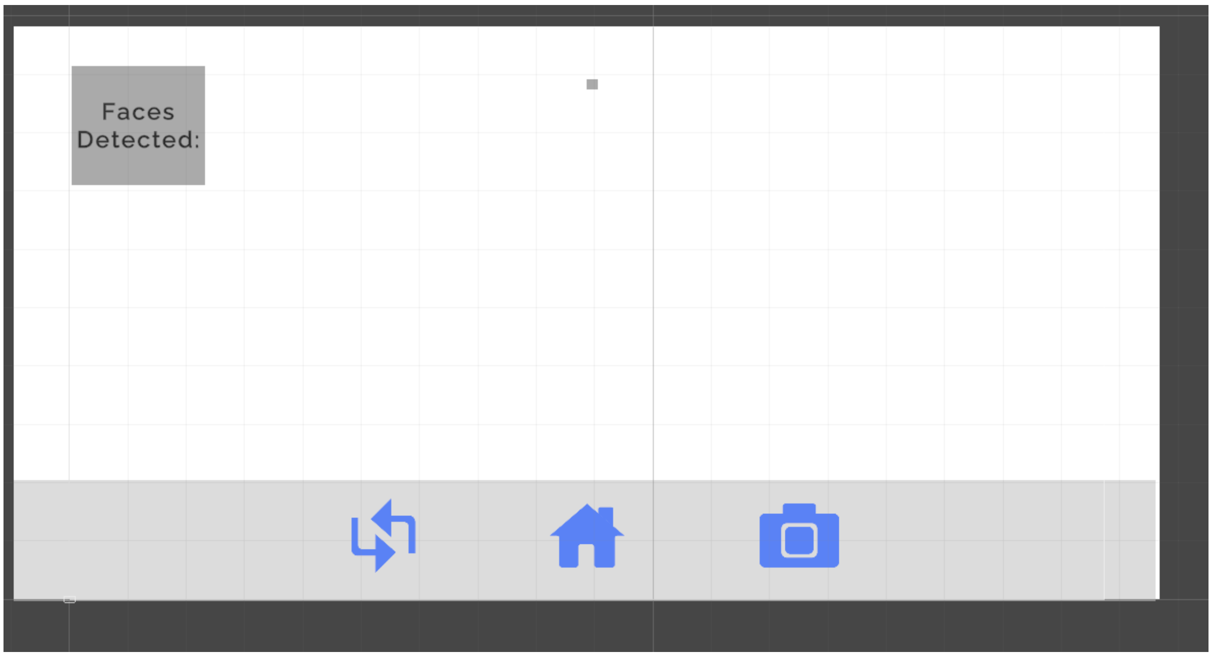
TextMeshPro - Text (UI)
Text Input — Enable RTL Editor

Text Style: Normal
Main Settings
Font Asset: Rale Medium (TMP_
Material Preset: Rale Medium Material



As seen in the figures above, the text background dynamically changes with the length of the text in the TMP textbox below. This means that there is no need for any complex code based on the length of the returned caption.

I have come across the issue where an empty caption shows a small grey box with no content. This is because of the padding that I put on the image object which stops it from being empty.

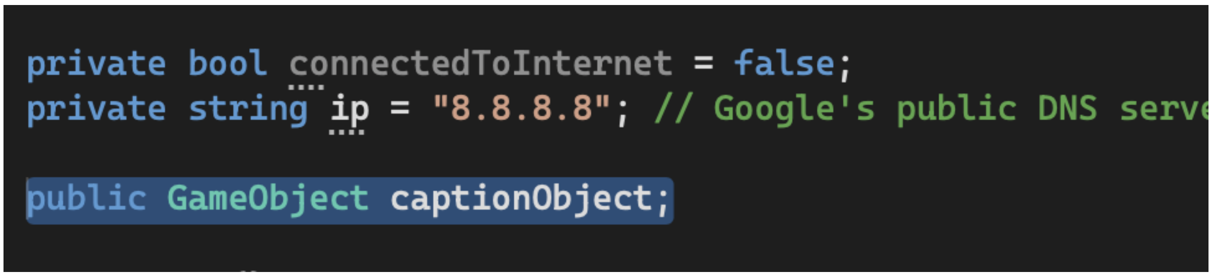


This can be fixed in 2 ways:

1) Set the padding to 0 and then switch it to 10 via a script when a caption is passed
2) Disable the gameObject and then re-enable when a caption is returned using a boolean
3) Check if the text component of the gameObject is empty. If it is, disable/hide the object

I am going to fix this bug using the third method as it seems the simplest and the best foolproof method.

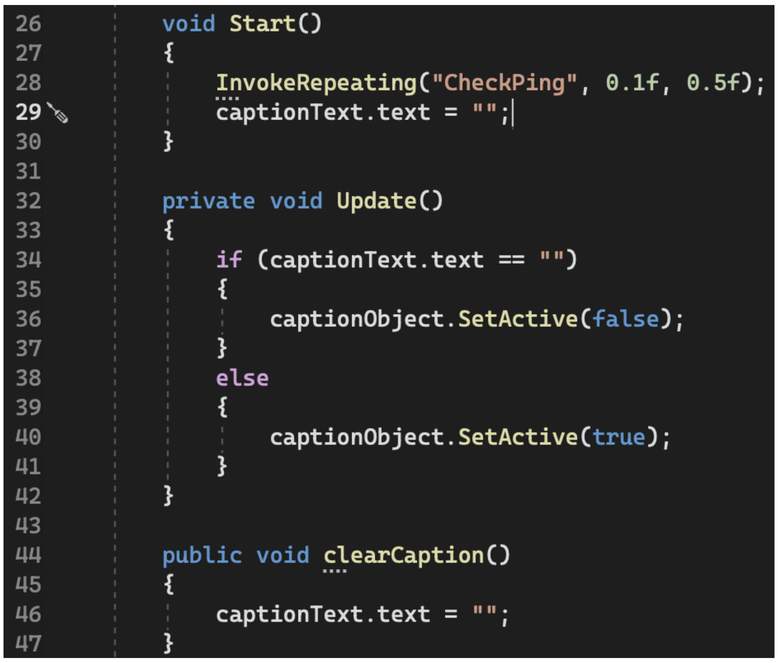
sendScreenshot.cs



```
private bool connectedToInternet = false;
private string ip = "8.8.8.8"; // Google's public DNS serve

public GameObject captionObject;
```

Firstly, I added the public gameObject that will be enabled and disabled based on the caption length.

sendScreenshot.cs

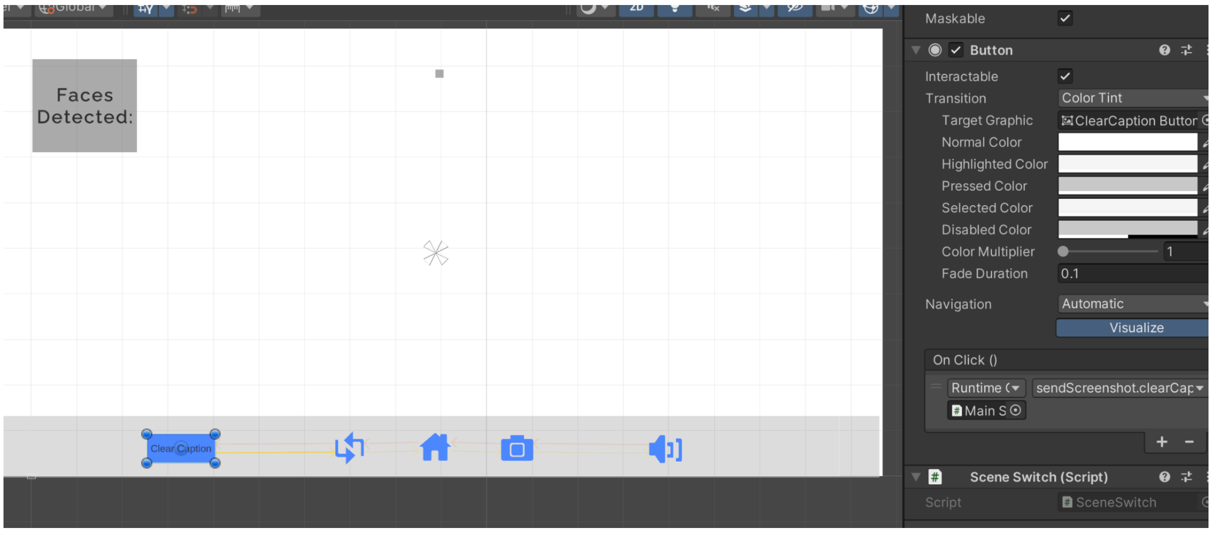


```
void Start()
{
    InvokeRepeating("CheckPing", 0.1f, 0.5f);
    captionText.text = "";
}

private void Update()
{
    if (captionText.text == "")
    {
        captionObject.SetActive(false);
    }
    else
    {
        captionObject.SetActive(true);
    }
}

public void clearCaption()
{
    captionText.text = "";
}
```

This logic ensures that when the text value in the captionText is empty, the parent object will be disabled. This is done by continuously checking the state of the captiontext text field through the update() method.

When the program starts, the captionText is set to be empty which ensures that the text box is disabled.

I have also included a public function called clearCaption() that sets the text field to empty when the user requests. This is so that if they want the caption to disappear when they want a better look at the screen, they can press a button, the text field is emptied, and thus the gameObject is hidden.



Maskable ✓

Button

Interactable	✓
Transition	Color Tint
Target Graphic	ClearCaption Button
Normal Color	
Highlighted Color	
Pressed Color	
Selected Color	
Disabled Color	
Color Multiplier	1
Fade Duration	0.1
Navigation	Automatic

Visualize

On Click ()

Runtime | sendScreenshot.clearCap | Main S

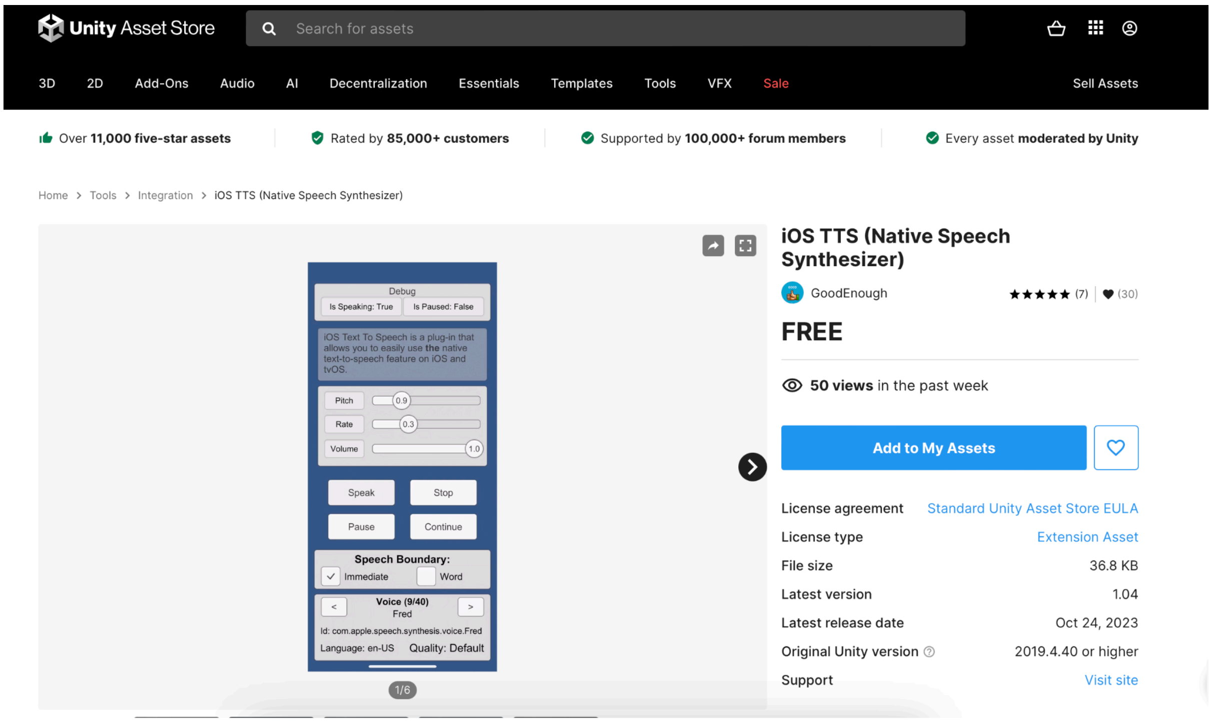
Scene Switch (Script)

Script | SceneSwitch

I have created a button labeled 'clear captions' and set the linking function to the clearCaption() function from the *sendScreenshot.cs* script.

Text-To-Speech

In order for TTS to work, I will need to import an external library, asset, or SDK, many of which are not freely available. This is because Unity does not have TTS functionality. Upon researching, I found the following asset which should work with iOS, which is what my project aims to target:



Over 11,000 five-star assets | Rated by 85,000+ customers | Supported by 100,000+ forum members | Every asset moderated by Unity

Home > Tools > Integration > iOS TTS (Native Speech Synthesizer)

iOS Text To Speech is a plug-in that allows you to easily use the native text-to-speech feature on iOS and tvOS.

Id: com.apple.speech.synthesis.voice.Fred
Language: en-US Quality: Default

iOS TTS (Native Speech Synthesizer)

GoodEnough ★★★★★ (7) | ♥ (30)

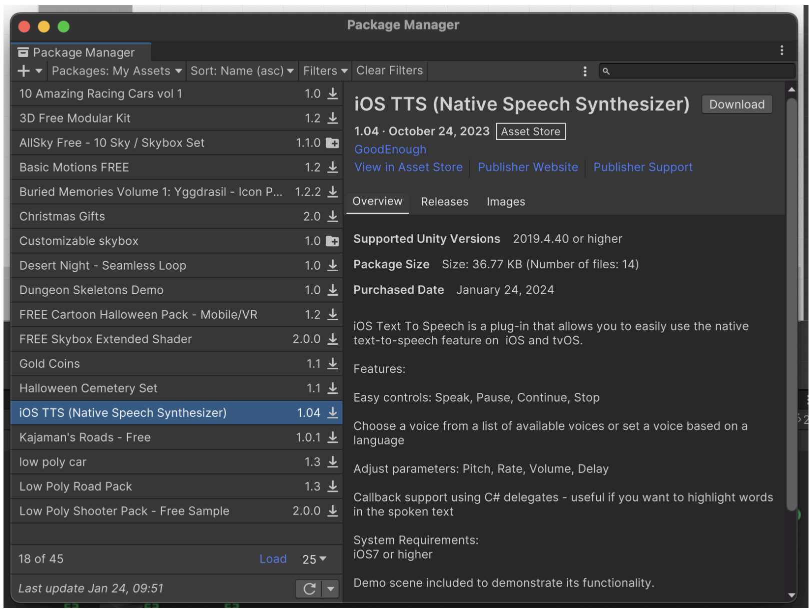
FREE

50 views in the past week

Add to My Assets

License agreement	Standard Unity Asset Store EULA
License type	Extension Asset
File size	36.8 KB
Latest version	1.04
Latest release date	Oct 24, 2023
Original Unity version	2019.4.40 or higher
Support	Visit site

Importing the package into Unity



10 Amazing Racing Cars vol 1	1.0
3D Free Modular Kit	1.2
AllSky Free - 10 Sky / Skybox Set	1.1.0
Basic Motions FREE	1.2
Buried Memories Volume 1: Yggdrasil - Icon P...	1.2.2
Christmas Gifts	2.0
Customizable skybox	1.0
Desert Night - Seamless Loop	1.0
Dungeon Skeletons Demo	1.0
FREE Cartoon Halloween Pack - Mobile/VR	1.2
FREE Skybox Extended Shader	2.0.0
Gold Coins	1.1
Halloween Cemetery Set	1.1
iOS TTS (Native Speech Synthesizer)	1.04
Kajaman's Roads - Free	1.0.1
low poly car	1.3
Low Poly Road Pack	1.3
Low Poly Shooter Pack - Free Sample	2.0.0

18 of 45 Load 25

Last update Jan 24, 09:51

iOS TTS (Native Speech Synthesizer) Download

1.04 · October 24, 2023 Asset Store

GoodEnough

View in Asset Store | Publisher Website | Publisher Support

Overview Releases Images

Supported Unity Versions 2019.4.40 or higher

Package Size Size: 36.77 KB (Number of files: 14)

Purchased Date January 24, 2024

iOS Text To Speech is a plug-in that allows you to easily use the native text-to-speech feature on iOS and tvOS.

Features:

Easy controls: Speak, Pause, Continue, Stop

Choose a voice from a list of available voices or set a voice based on a language

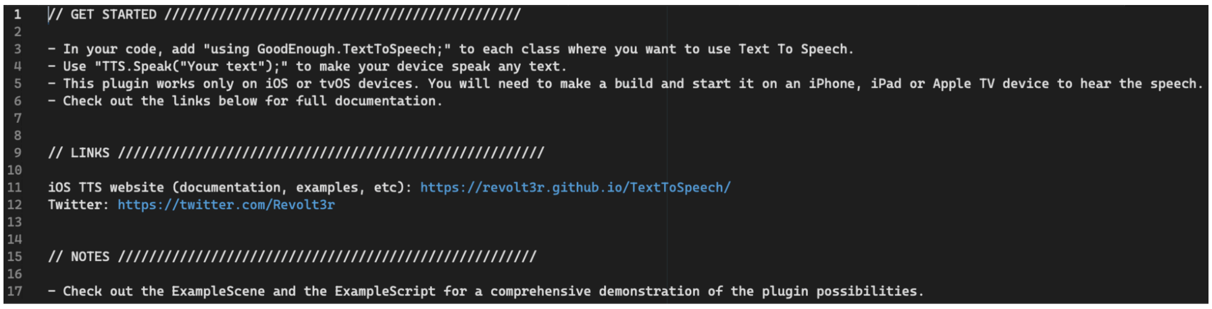
Adjust parameters: Pitch, Rate, Volume, Delay

Callback support using C# delegates - useful if you want to highlight words in the spoken text

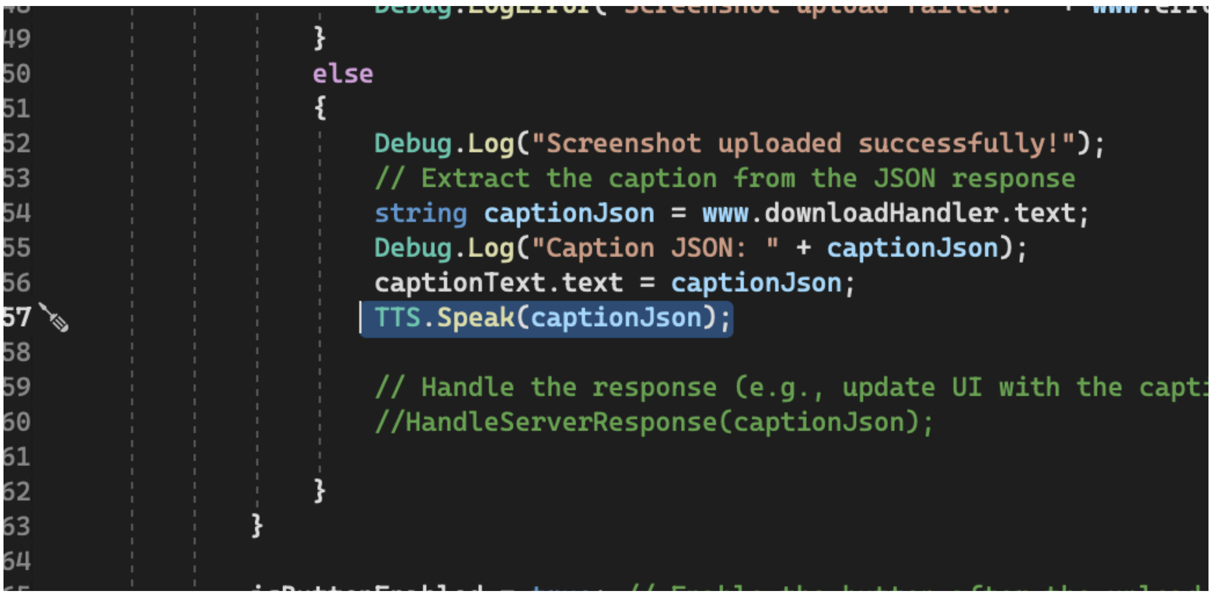
System Requirements:
iOS7 or higher

Demo scene included to demonstrate its functionality.

In order to understand how to implement this into my project, I took a look at the included 'readme.txt' which included some vital information seen below:

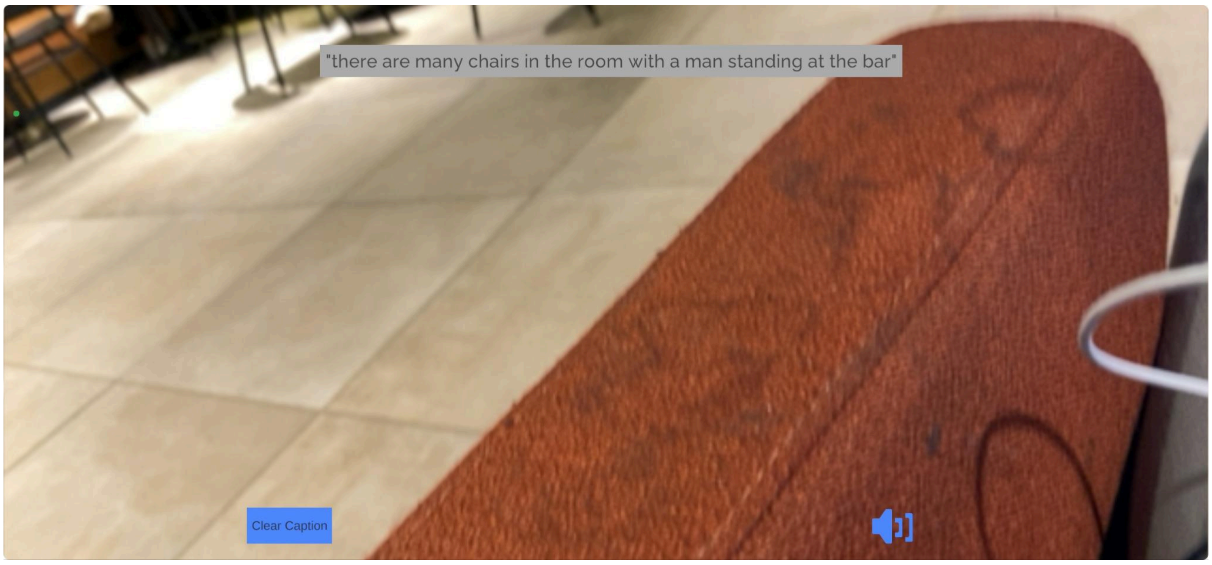

```
// GET STARTED ////////////////////////////////////////////////

- In your code, add "using GoodEnough.TextToSpeech;" to each class where you want to use Text To Speech.
- Use "TTS.Speak("Your text");" to make your device speak any text.
- This plugin works only on iOS or tvOS devices. You will need to make a build and start it on an iPhone, iPad or Apple TV device to hear the speech.
- Check out the links below for full documentation.


// LINKS ///////////////////////////////////////////////////////

iOS TTS website (documentation, examples, etc): https://revolt3r.github.io/TextToSpeech/
Twitter: https://twitter.com/Revolt3r


// NOTES ///////////////////////////////////////////////////////

- Check out the ExampleScene and the ExampleScript for a comprehensive demonstration of the plugin possibilities.
```

sendScreenshot.cs

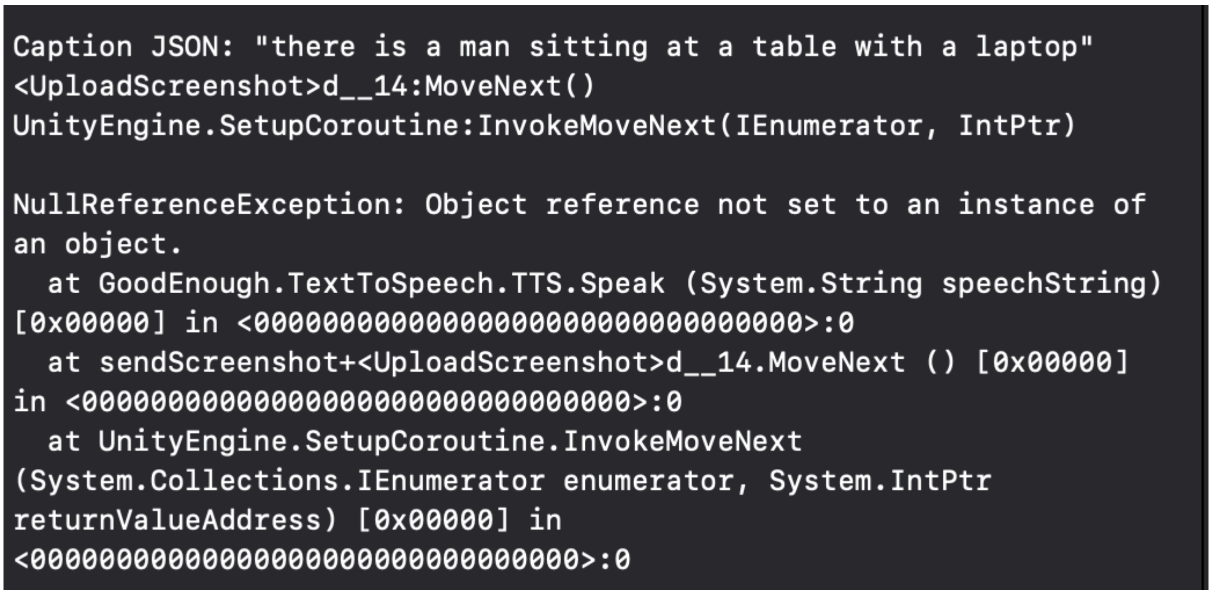

```
                }
                else
                {
                    Debug.Log("Screenshot uploaded successfully!");
                    // Extract the caption from the JSON response
                    string captionJson = www.downloadHandler.text;
                    Debug.Log("Caption JSON: " + captionJson);
                    captionText.text = captionJson;
                    TTS.Speak(captionJson);

                    // Handle the response (e.g., update UI with the capt
                    //HandleServerResponse(captionJson);

                }
            }
```

In order to test the functionality of the TTS plugin, I will set the code to generate a caption as soon as the caption is returned and say it aloud. I will build this on my phone and assess its functionality.

Upon testing, a caption is generated, however nothing is said and the UI items that disappear and then re-appear remain disabled, which means that the code is stuck after generating and displaying the caption

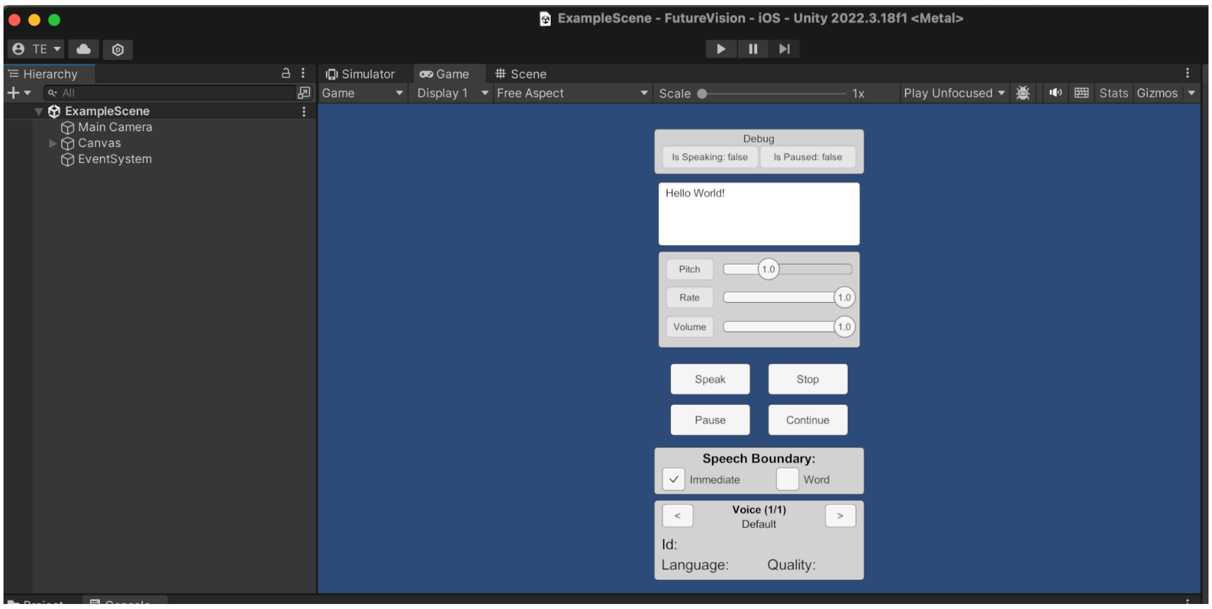

"there are many chairs in the room with a man standing at the bar"
Clear Caption

I checked the xCode debug console and came across this error.



```
Caption JSON: "there is a man sitting at a table with a laptop"
<UploadScreenshot>d__14:MoveNext()
UnityEngine.SetupCoroutine:InvokeMoveNext(IEnumerator, IntPtr)

NullReferenceException: Object reference not set to an instance of
an object.
  at GoodEnough.TextToSpeech.TTS.Speak (System.String speechString)
[0x00000] in <00000000000000000000000000000000>:0
  at sendScreenshot+<UploadScreenshot>d__14.MoveNext () [0x00000]
in <00000000000000000000000000000000>:0
  at UnityEngine.SetupCoroutine.InvokeMoveNext
(System.Collections.IEnumerator enumerator, System.IntPtr
returnValueAddress) [0x00000] in
<00000000000000000000000000000000>:0
```

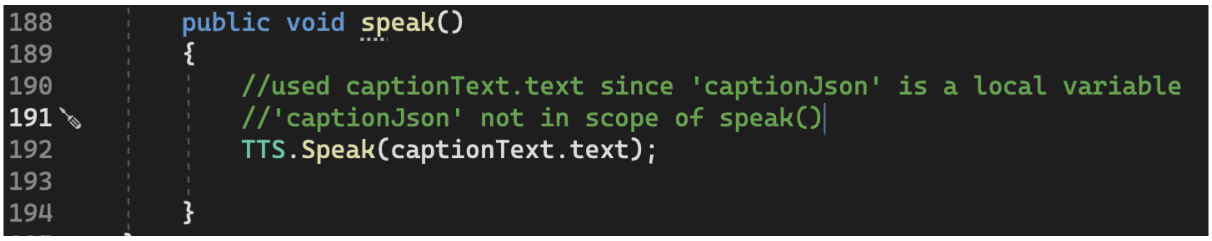
I decided to use the examplescene provided with the plugin and test this, seeing if it works adequately:



Upon compiling for my phone, it searched the onboard memory for the built-in ios voices folder and worked perfectly. I will perhaps use this as settings tab to simplify the process, as it is all readily available. If I have time, I may update the UI, however this isn't necessary at this point.

Now, I will attempt to use the same 'speak' button for my main scene in unity, by using the same built-in script, but instead of reading the textbox, it will read the server response.

sendScreenshot.cs



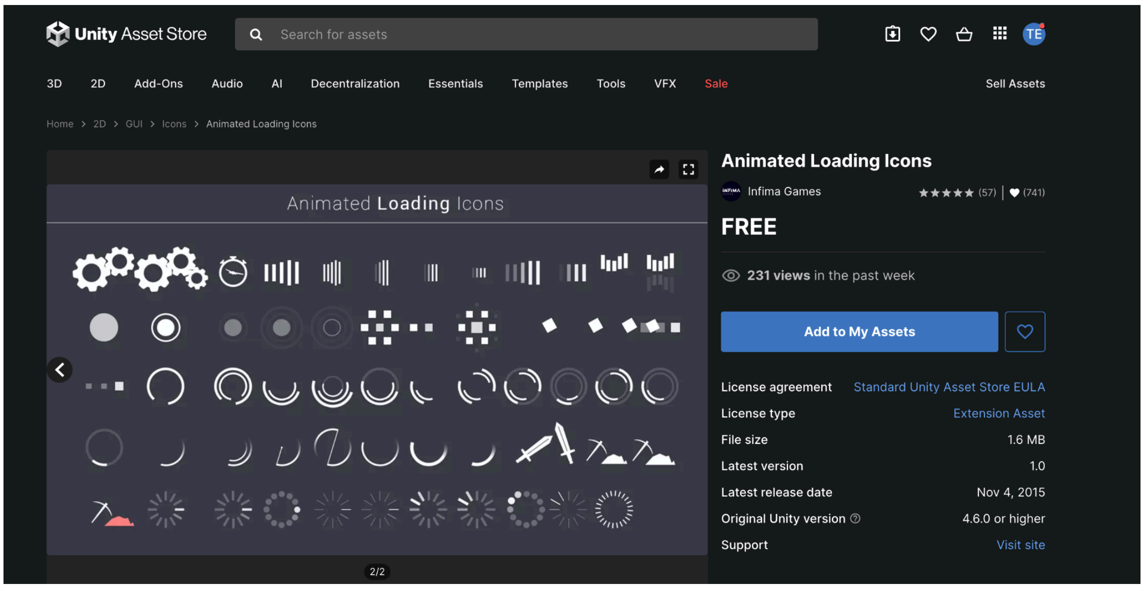
```
    public void speak()
    {
        //used captionText.text since 'captionJson' is a local variable
        //'captionJson' not in scope of speak()
        TTS.Speak(captionText.text);

    }
```

The program will now load the voices by loading the exampleScene with all the relevant code and now, when pressing the speak button, the caption will be read aloud, in any voice that the user pleases to choose.

Loading Caption

When the application is loading a caption, the UI all disappears and depending on internet speeds and server response time, the time taken for a caption to be returned can be relatively long. As a result, I want to implement some form of loading animation while the image is being processed and a button that allows the user to terminate the request, allowing the UI objects to re-appear and continue with the emotion detection as before.

For the loading animation, I will be using the following open-source loading icon pack from the unity asset store:



Animated Loading Icons

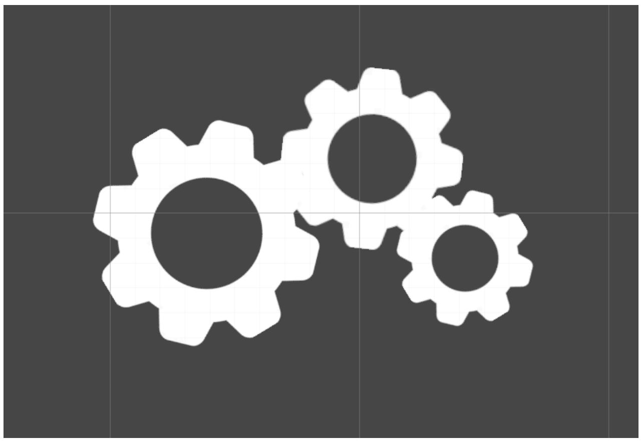
Infima Games ★★★★★ (57) | ♥ (741)

FREE

231 views in the past week

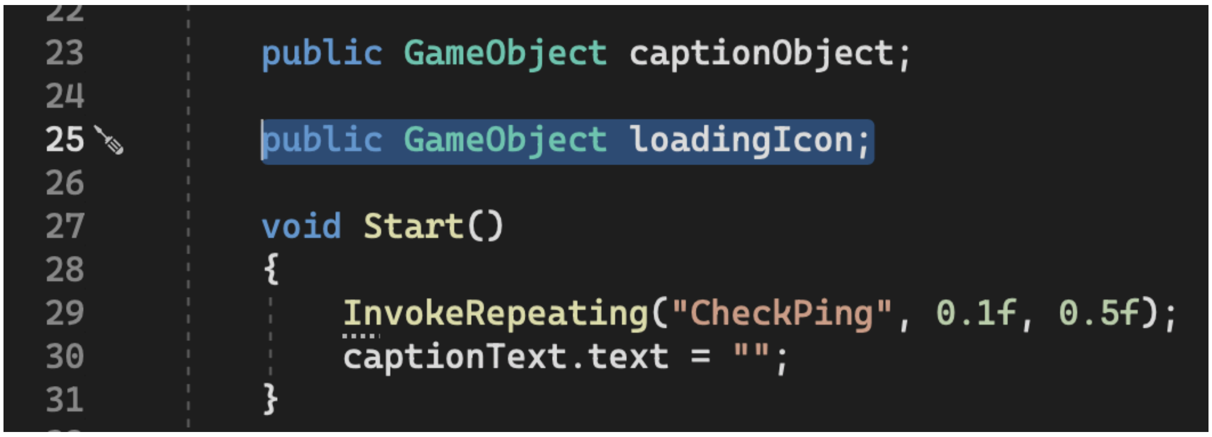
License agreement	Standard Unity Asset Store EULA
License type	Extension Asset
File size	1.6 MB
Latest version	1.0
Latest release date	Nov 4, 2015
Original Unity version	4.6.0 or higher
Support	Visit site

After importing the asset folder into my project, I had a look at the various prefabs that were included, and came across the following, which caught my attention, and I thought would fit the theme of the project quite well:



I will import these into the main scene, when the image is sent to the server, the script should enable this gameObject and disable everything else. To do this, I should create a public variable to hold this gameObject and change its state via the *sendScreenshot.cs* script.

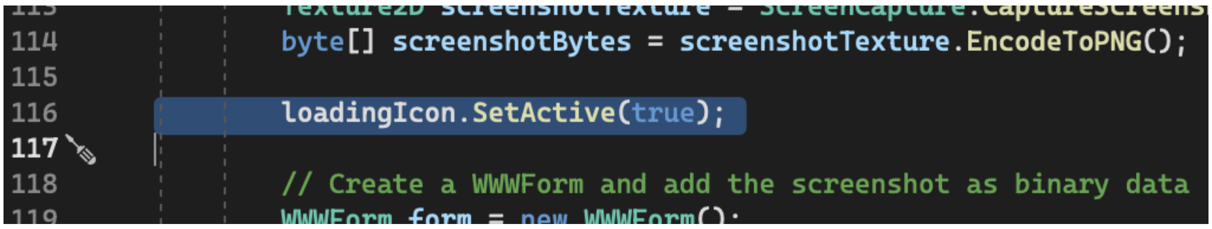
sendScreenshot.cs



```
public GameObject captionObject;

public GameObject loadingIcon;

void Start()
{
    InvokeRepeating("CheckPing", 0.1f, 0.5f);
    captionText.text = "";
}
```

Create the variable for the loading Icon.

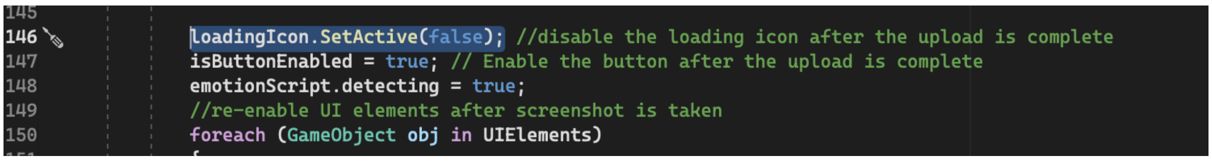
sendScreenshot.cs



```
Texture2D screenshotTexture = ScreenCapture.CaptureScreens
byte[] screenshotBytes = screenshotTexture.EncodeToPNG();

loadingIcon.SetActive(true);

// Create a WWWForm and add the screenshot as binary data
WWWForm form = new WWWForm();
```

Right after the screenshot has been taken and converted to byte format, the loading icon prefab is loaded and the animation will automatically start playing.

sendScreenshot.cs



```
loadingIcon.SetActive(false); //disable the loading icon after the upload is complete
isButtonEnabled = true; // Enable the button after the upload is complete
emotionScript.detecting = true;
//re-enable UI elements after screenshot is taken
foreach (GameObject obj in UIElements)
```

Right after the upload is complete and the caption is returned, the loadingIcon is re-activated.

Due to time constraints for iteration 3, I will not be able to implement a ‘cancel upload’ button. If I were to implement this feature in a future iteration, I would implement a method that can stop the coroutine and handle any necessary cleanup, I would do this through setting up a flag that would be able to skip the request.

Test Plan for this version

Due to the nature of my project being dependent on many different functions and inputs, I will need to test that each of the desired outputs are displayed when the expected input is used. To track the test data, I will be using the table below, and providing each test data with a level of priority, (1 being high priority, 2 being mediocre priority, and 3 being low priority)

Test num.	What is being tested	Priority	Input	Expected output	Justification for this data
1	On-screen face counter UI updating	2	Detected faces	Integer displaying number of faces in real-time on a grey background	This will aid in useability and promote a more user-friendly readable interface
2	Image caption on grey background	2	Image Captions	Grey background that the captions are displayed on so the user can see clearly	The YoLo object detection is another key aspect of this project. It is important that this functionality works well.
3	Splash screen/scene switcher UI Buttons	3	touchscreen	The splash screen will allow the user to choose which functionality they would like to use in the app, whether it be emotion/face detection, or a general-purpose object detection AI.	This is to allow the user to navigate the application. They need a splash screen that will allow it. The splash screen should be the one the loads up when the user opens the application.
4	Settings/Voice Settings page	1	Settings Page button	Opens the settings scene for the TTS functionality	This is needed as the captions will be read aloud through a TTS system. This should be customizable by the user in order to tailor to their needs and preferences
5	Caption Text-To-Speech	1	TTS button	Upon pressing the button, the application will read out the displayed caption out loud, in the voice/setup done by the user	This is one of the key features of iteration 3 and the project itself, which aims to assist those with visual impairments by providing them with a new, audible way of viewing their surroundings using AI.
6	Clear Captions button	2	Clear Captions button	When this button is pressed, any caption loaded and displayed on screen is cleared to provide a clearer view of the screen.	This is important as if the user wants to focus more on the camera output and facial expressions, the caption can get in the way, so there is a button to get rid of this.

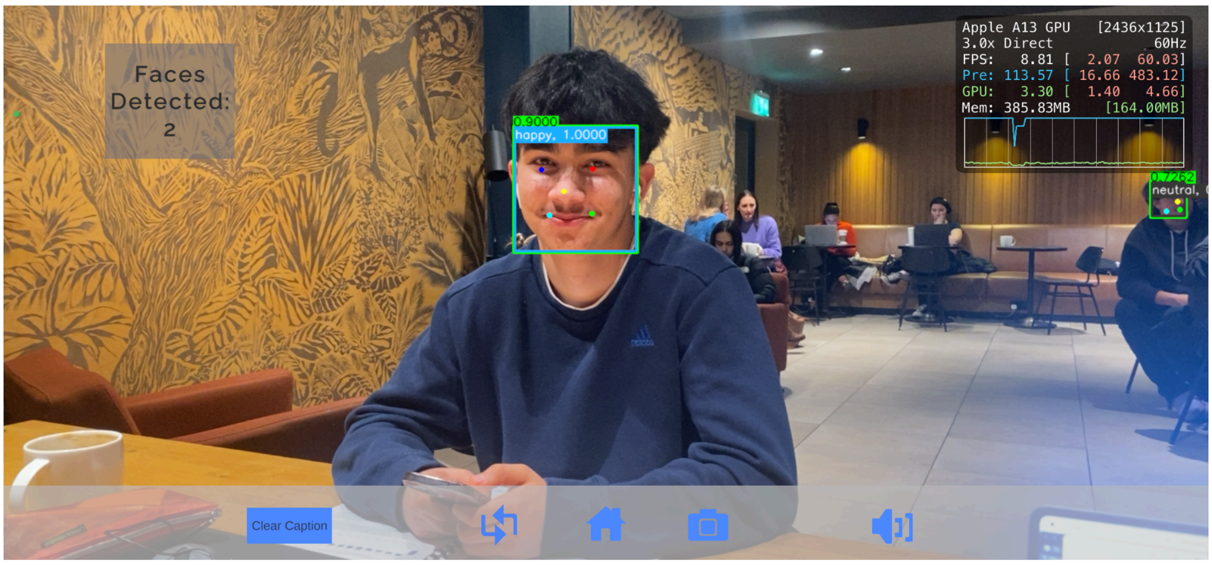
7	‘Cancel Upload’ functionality	2	‘Cancel upload’ button	When this button is pressed, the caption request is cancelled and the UI elements re-appear on screen	This is important as if the client has a very slow internet connection, this process can be tedious and boring, therefore they should have an option to cancel this operation on request.
8	‘Loading’ icon for image captioning	2	Image caption button	When the image is being processed and request is sent to server, animated icon starts playing to indicate it is being processed.	The loading icon will indicate to the user when the image is being captioned, that way they aren’t confused about the lack of UI and functionality when the button is pressed.
9	TTS customisation	2	‘Settings’ Page	The user will have the ability to customise the speed, pitch, and volume of the Text-to-Speech voice	This will help the user have a more personalised experience within the app, which is the primary focus for iteration 3
10	iPhone deployment	1	n/a	The application runs and works when built on an iphone using xCode for deployment	This will be tested at each iteration in order to ensure that during development, no build errors occur which may cause the application to fail when deploying to an iPhone.

Test Results / Evidence

Test num.	What is being tested	Result	Input	Expected output	Comments	Evidence
1	On-screen face counter UI updating	🟩	Device webcam + detected faces	Integer displaying number of faces in real-time on a grey background	n/a	Figure 3.1
2	Image caption on grey background	🟩	Image Captions	Grey background that the captions are displayed on so the user can see clearly	n/a	Figure 3.2
3	Splash screen/scene switcher UI Buttons	🟩	touchscreen	The splash screen will allow the user to choose which functionality they would like to use in the app, whether it be emotion/face detection, or a general-purpose object detection AI.	n/a	Figure 3.3
4	Settings/Voice Settings page	🟧	Settings Page button	Opens the settings scene for the TTS functionality	Since I ran out of time, I was not able to make a custom settings page with a clean UI	Figure 3.4
5	Caption Text-To-Speech	🟩	TTS button	Upon pressing the button, the application will read out the displayed caption out loud.	Works as expected, in the desired voice that the user sets up	n/a
6	Clear Captions button	🟩	Clear Captions button	When this button is pressed, any caption loaded and displayed on screen is cleared to provide a clearer view of the screen.	n/a	Figure 3.5
7	'Cancel upload' functionality	🟥	'Cancel upload' button	When this button is pressed, the caption request is cancelled and the UI elements re-appear on screen	Due to time constraints, I was unable to implement this feature into iteration 3.	n/a
8	'Loading' icon for image captioning	🟩	Image caption button	When the image is being processed and request is sent to server, animated icon starts playing to indicate it is being processed.	n/a	Figure 3.6

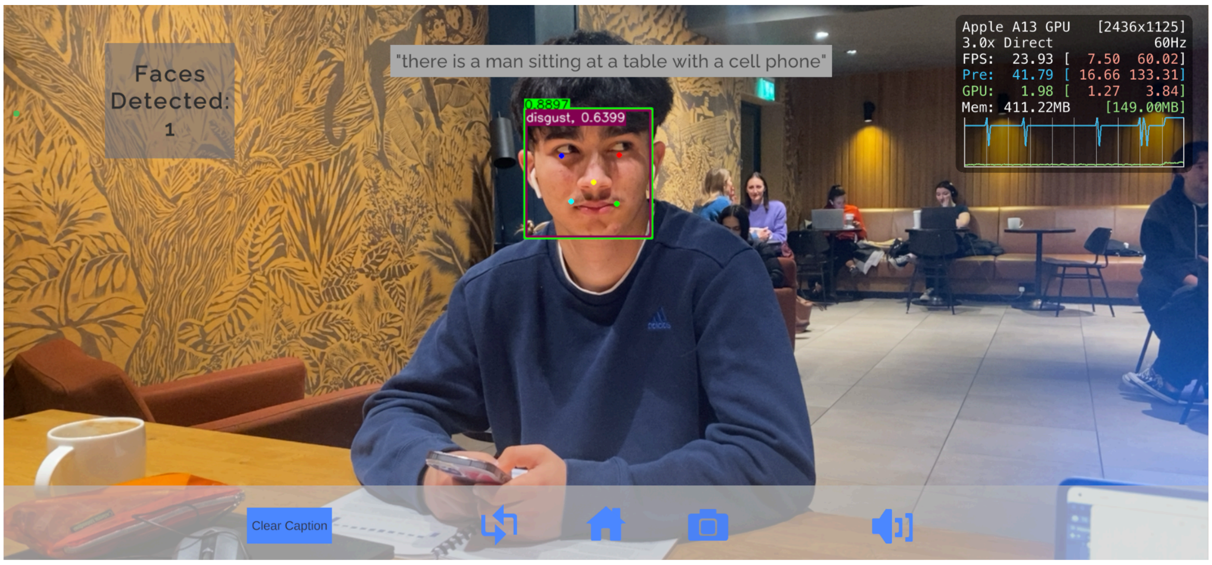
9	TTS customisation	🟩	'Settings' Page	The user will have the ability to customise the speed, pitch, and volume of the Text-to-Speech voice	Due to time constraints, I was unable to improve the UI of the the settings page and had to stick to the pre-made TTS customisation scene provided by the TTS package, however in terms of the functionality, this works brilliantly	Figure 3.4
10	iPhone deployment	🟩	n/a	The application runs and works when built on an iphone using xCode for deployment	n/a	See all figures

Figure 3.1



```
Apple A13 GPU   [2436x1125]
3.0x Direct            60Hz
FPS:    8.81 [  2.07  60.03]
Pre:  113.57 [ 16.66 483.12]
GPU:    3.30 [  1.40   4.66]
Mem: 385.83MB     [164.00MB]
```

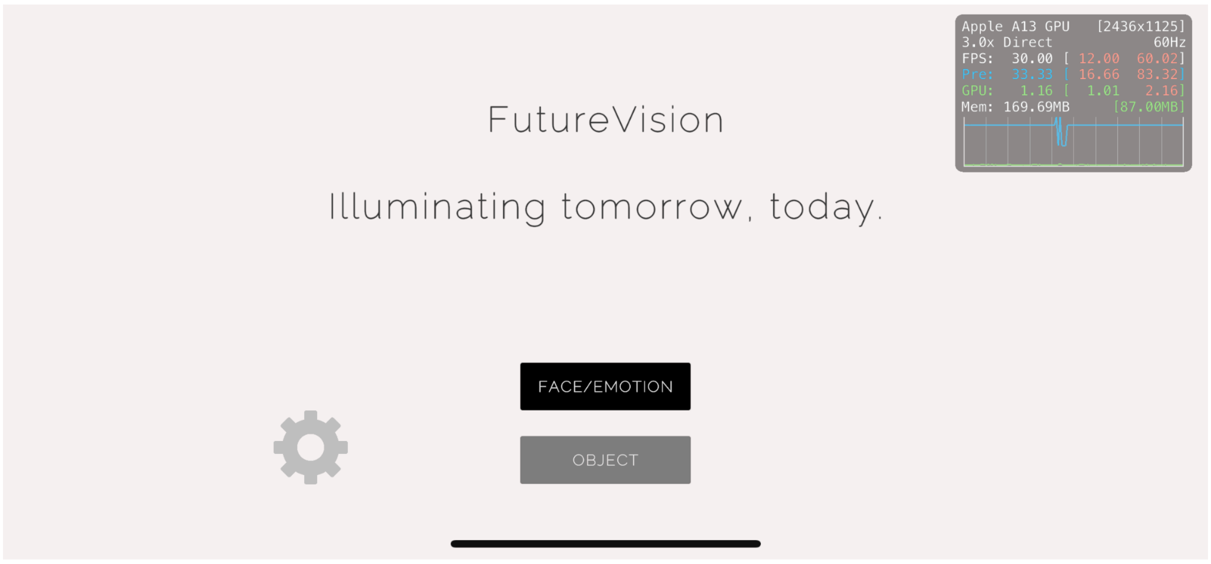
Figure 3.2



"there is a man sitting at a table with a cell phone"

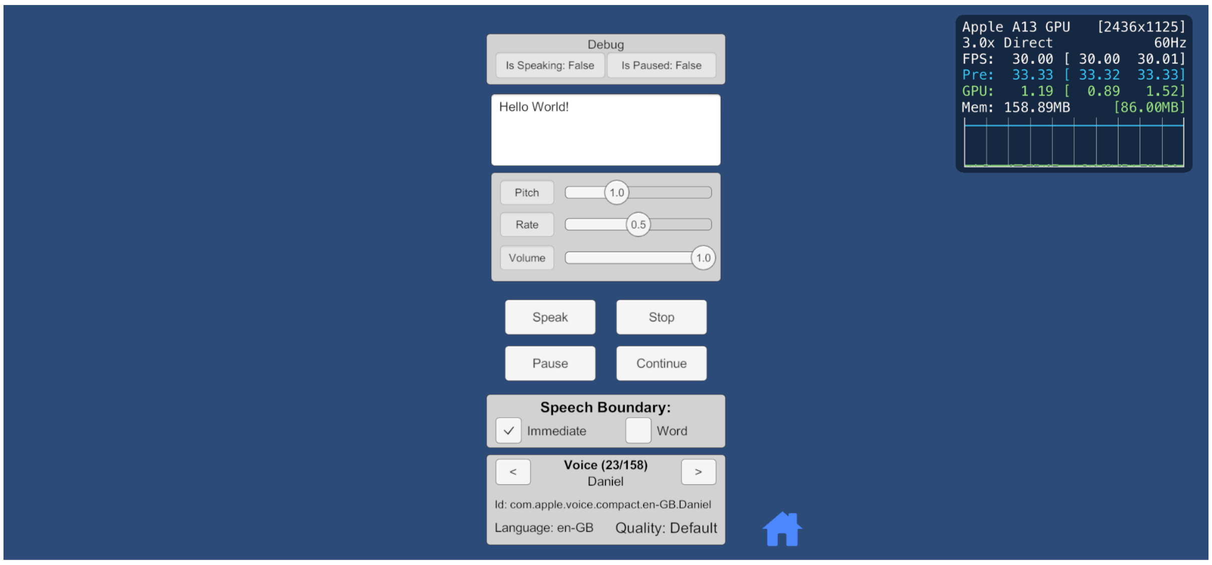
```
Apple A13 GPU    [2436x1125]
3.0x Direct             60Hz
FPS:   23.93 [  7.50  60.02]
Pre:   41.79 [ 16.66 133.31]
GPU:    1.98 [  1.27   3.84]
Mem:  411.22MB      [149.00MB]
```

Figure 3.3



```
Apple A13 GPU    [2436x1125]
3.0x Direct             60Hz
FPS:   30.00 [ 12.00  60.02]
Pre:   33.33 [ 16.66  83.32]
GPU:    1.16 [  1.01   2.16]
Mem:  169.69MB       [87.00MB]
```

Figure 3.4



Debug

Is Speaking: False | Is Paused: False

Hello World!

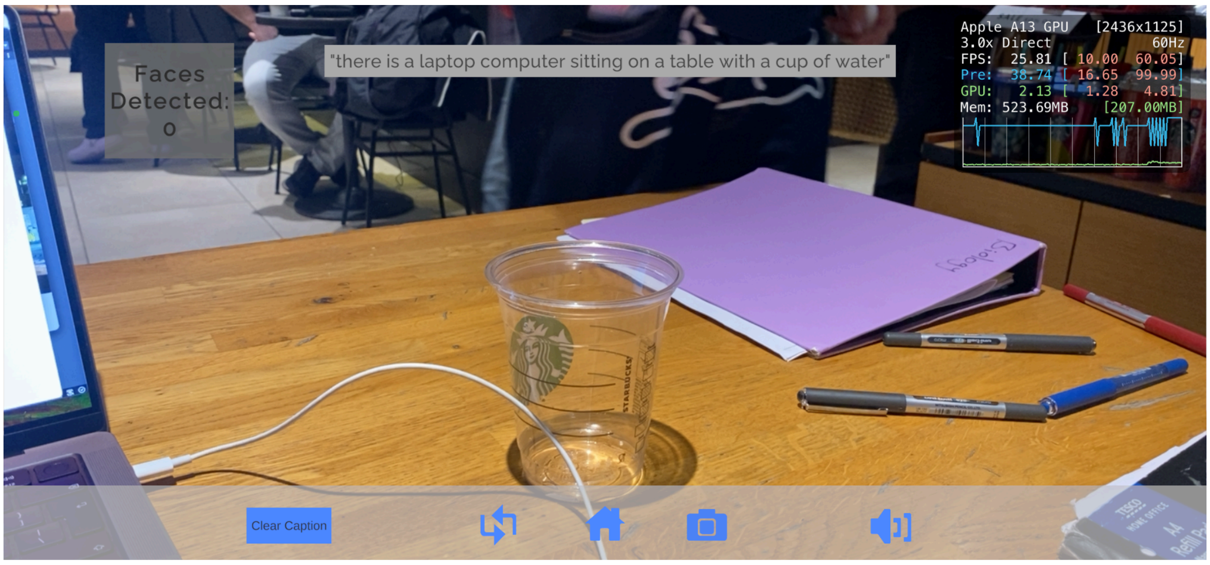
Pitch 1.0
Rate 0.5
Volume 1.0

Speech Boundary:
[✓ Immediate] Word

Voice (23/158)
Daniel
Id: com.apple.voice.compact.en-GB.Daniel
Language: en-GB Quality: Default

```
Apple A13 GPU    [2436x1125]
3.0x Direct             60Hz
FPS:   30.00 [ 30.00  30.01]
Pre:   33.33 [ 33.32  33.33]
GPU:    1.19 [  0.89   1.52]
Mem: 158.89MB      [86.00MB]
```

Figure 3.5.1
(Caption is on the screen, taking up space)

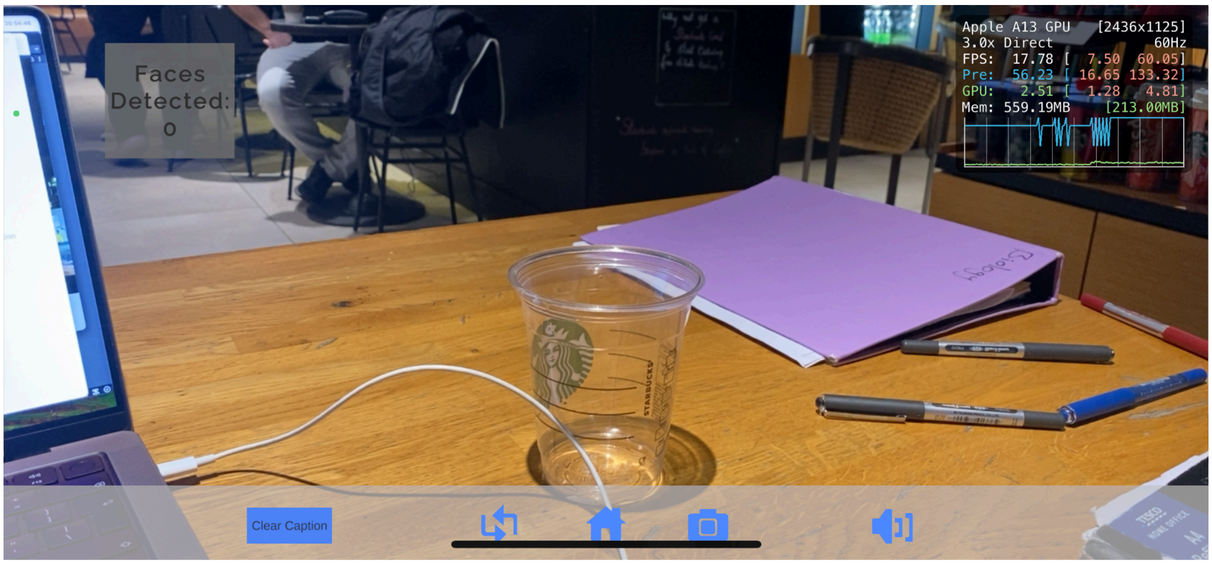


Faces Detected: 0

"there is a laptop computer sitting on a table with a cup of water"

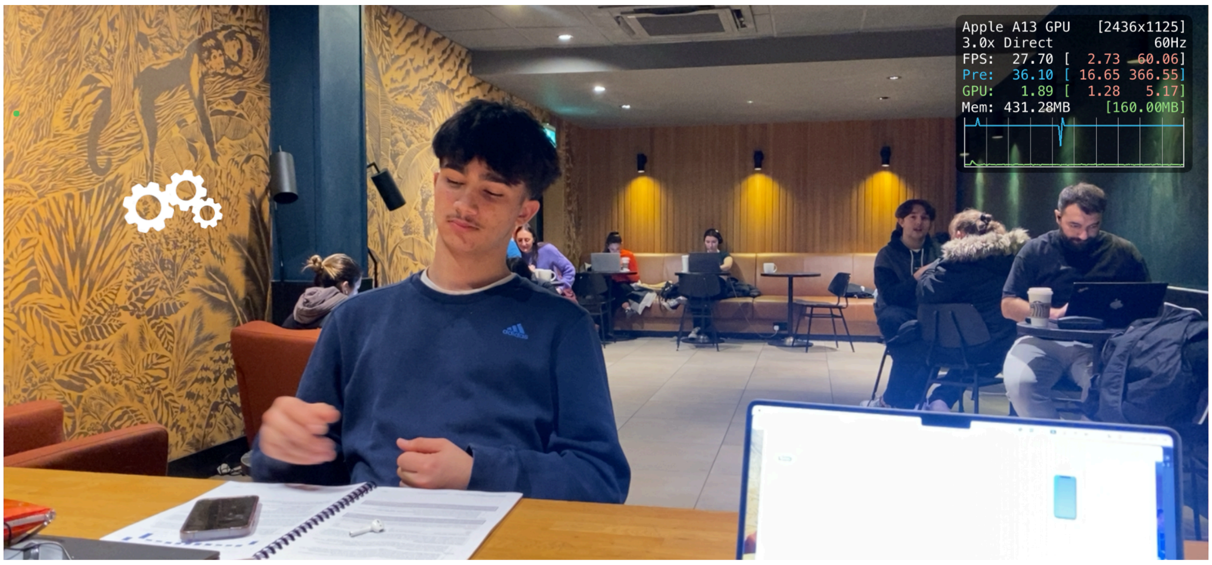
```
Apple A13 GPU    [2436x1125]
3.0x Direct             60Hz
FPS:   25.81 [ 10.00  60.05]
Pre:   38.74 [ 16.65  99.99]
GPU:    2.13 [  1.28   4.81]
Mem: 523.69MB     [207.00MB]
```

Figure 3.5.2
(After pressing clear caption, the caption box disappears, freeing up space)



```
Apple A13 GPU    [2436x1125]
3.0x Direct            60Hz
FPS:  17.78 [  7.50  60.05]
Pre:  56.23 [ 16.65 133.32]
GPU:   2.51 [  1.28   4.81]
Mem: 559.19MB     [213.00MB]
```

Figure 3.6



```
Apple A13 GPU    [2436x1125]
3.0x Direct            60Hz
FPS:  27.70 [  2.73  60.06]
Pre:  36.10 [ 16.65 366.55]
GPU:   1.89 [  1.28   5.17]
Mem: 431.28MB     [160.00MB]
```

Feedback from Stakeholder

In the previous iterations, I highlighted that my stakeholders consist of potential users and keen tech enthusiasts on exploring the practical uses of artificial intelligence or seeking to enhance their daily lives through this project. With the integration of crucial functionalities, including user interface enhancements and auditory feedback, the project has reached a level of usability that I believe warrants further evaluation. Therefore, I plan to conduct interviews with all three stakeholders: Aiad Tarik, Mario Prifti, and Mike Parish. For clarity and brevity during the presentation of their feedback, they will be abbreviated as AT, MPr, and MPa, respectively.

How seamless do you find the navigation with the newly updated user interface, especially with the addition of symbols for buttons and the revamped splash screen?

AT: The updated interface is a significant improvement. The use of symbols over text for buttons simplifies navigation and makes the app feel more intuitive. The splash screen adds a professional touch that aligns well with the overall design language of the application. It's clear that thought has been put into making the UI efficient and visually appealing.

MPr: I love the new look! The symbols are a smart choice because they make the app look modern and are really easy to understand at a glance. The splash screen with the logo and slogan gives a cool first impression that makes me excited to use the app. It's definitely more seamless than before and feels like something designed for my generation.

MPa: I found the changes quite helpful. The symbols on the buttons are much easier for me to navigate through the app, compared to remembering what each text meant. The new splash screen made the app feel more welcoming and less intimidating for someone like me. It's a lot easier to use now, and I appreciate the effort to make it accessible.

What are your thoughts on the integration of the Text-To-Speech function for reading aloud image captions? Does it enhance the app's usability for you?

AT: The Text-To-Speech feature is a fantastic addition. It not only makes the app more accessible but also allows users to experience the content more dynamically. For someone who multitasks, being able to listen to the captions while doing something else is a great usability enhancement.

MPr: I think the Text-To-Speech is super cool! It makes the app feel more interactive, and I can share the captions with friends without having to read them out loud myself. It's like having a narrator for the pictures, which adds a whole new layer to how I engage with the content.

MPa: The Text-To-Speech is a brilliant feature for me. Reading on a small screen can be challenging, so having the captions read aloud is very helpful. It makes me feel more connected to what's on the screen and ensures I don't miss out on any information because of small text.

Considering the updates to the UI and the introduction of auditory feedback, do you find the application more user-friendly and accessible, especially for individuals who might not be as familiar with technology?

AT: Absolutely. The enhancements in UI and auditory feedback significantly contribute to the app's user-friendliness. It feels like the app is now designed to cater to a wider audience, including those who might not have extensive tech experience. These features bridge the gap between advanced functionality and ease of use.

MPr: Yeah, for sure! The updates made the app a lot more fun and easier to use. The sounds and voice feedback make it feel like the app is talking to me, which is pretty awesome. It’s definitely a step up in making technology feel more personal and less daunting for everyone.

MPa: Yes, the improvements have made a big difference. I used to find apps like this quite complicated, but the clearer UI and the voice feedback have made it much simpler for me to understand and use. It’s reassuring to know that developers are considering people like me when they make these updates.

Given the updated image captioning model, how do you now rate the accuracy and relevance of the captions generated by the app? Do you find them consistently reflective of the emotions and scenes depicted?

AT: The updates have markedly improved both the accuracy and relevance of the image captions. It's evident that the algorithm behind the captioning has become more sophisticated, often pinpointing the scenes with impressive precision. I find the captions consistently reflective of what's depicted, adding a layer of engagement that was previously missing. It's not just about recognizing faces or objects anymore; the app now captures the essence of the moment, which greatly enhances the user experience.

MPr: Definitely, the captions are way more on point now. Before, they were kinda hit or miss, especially with more complex scenes or subtle emotions. But now, it's like the app really gets what's happening in the picture and the mood of the people in it.

MPa: I've noticed a big improvement in how the captions describe the pictures I take. Before, they seemed a bit generic, but now they seem to really match what’s in the photo, including the mood or the setting.

Considering the diverse age range of our stakeholders, how do you find the app's applicability and usefulness across different age groups with the new features implemented?

AT: The implementation of new features has significantly enhanced the app's appeal and utility across various age groups. The intuitive design, combined with features like Text-To-Speech, addresses a wide range of user needs and preferences. For tech-savvy users, the sophisticated emotion and face detection offer a depth of interaction that’s both engaging and informative. Meanwhile, the improved usability and accessibility features ensure that even those who aren't as comfortable with technology can navigate the app with ease. It's a testament to thoughtful design that considers a broad user base, making it a valuable tool for anyone, regardless of age or tech proficiency.

MPr: With the latest updates, the app has become something that I can see people of all ages enjoying. The UI is sleek and easy to navigate, which is great for my friends and me, but it's also straightforward enough for older generations to use without getting frustrated. The Text-To-Speech feature is a hit because it makes the app more accessible and fun, especially for sharing captions out loud. It feels like the app is now more inclusive, taking into account the different ways people interact with technology today. Whether it's for education, entertainment, or just staying connected, the app has something to offer everyone.

MPa: The recent updates have made a clear difference in how approachable the app is for people of my age, as well as for the younger folks in my family. The simplification of the interface, combined with the audible feedback from the Text-To-Speech function, has made it much easier for me to use and enjoy the app. I've seen my grandchildren play around with the emotion detection and get a kick out of the captions, which shows me it's engaging for them too. It's rare to find technology that bridges the gap so well, making it not just a tool for one group but a shared experience that brings different generations together.

Based on your experience with the current iteration, are there any further improvements or features you would suggest, particularly in areas of usability or technology integration?

AT: While the app's technological enhancements are commendable, there are areas where further improvements could significantly elevate the user experience. A notable concern is the waiting time for image captions to load. Optimizing the backend processes or employing more efficient algorithms could reduce this latency, providing a smoother and more immediate interaction. Additionally, the app could better accommodate various screen sizes and resolutions. Ensuring that the UI elements dynamically adjust to fit different screen scales would greatly improve usability across all devices. This adaptability is crucial for ensuring that the app delivers a consistent and accessible experience, regardless of the user's device.

MPr: The app is really cool, but I think it could be even better with some social sharing features. It would be awesome to directly share the captions and photos on social media or within the app's own community platform. This would make it more engaging for users like me who love sharing interesting finds with friends.

MPa: I've found the app much easier to use with the recent updates, but I still think there could be more done for those of us who aren't as tech-savvy. Perhaps a tutorial or help section with video guides could be added to walk new users through the features and how to use them. This could be especially helpful for older users who might feel overwhelmed by new technology. Another suggestion would be to create a voice recognition feature to allow for voice commands, making it easier to navigate the app without having to rely as much on the touchscreen.

Evaluation

In this iteration, I concentrated on refining the user interface and enhancing usability, marking a pivotal step towards making the application more inclusive and accessible for users across the age spectrum. This phase was characterised by the successful incorporation of key features aimed at improving the overall user experience, demonstrating our commitment to bridging the technological divide and creating a universally appealing platform. The introduction of intuitive UI enhancements and the implementation of auditory feedback mechanisms exemplify the efforts to cater to diverse user needs, making technology more approachable and engaging for everyone.

Despite these achievements, I encountered limitations due to time constraints, which led to the postponement of several planned features and the inability to address existing issues fully. Notably, the absence of a 'cancel image upload' functionality and unresolved bugs, such as screen scaling inconsistencies and camera rotation challenges for portrait mode, were areas where our aspirations outpaced our capacity. These shortcomings highlight the complexities involved in software development, where prioritization and time management play critical roles in determining what can be accomplished within a given iteration.

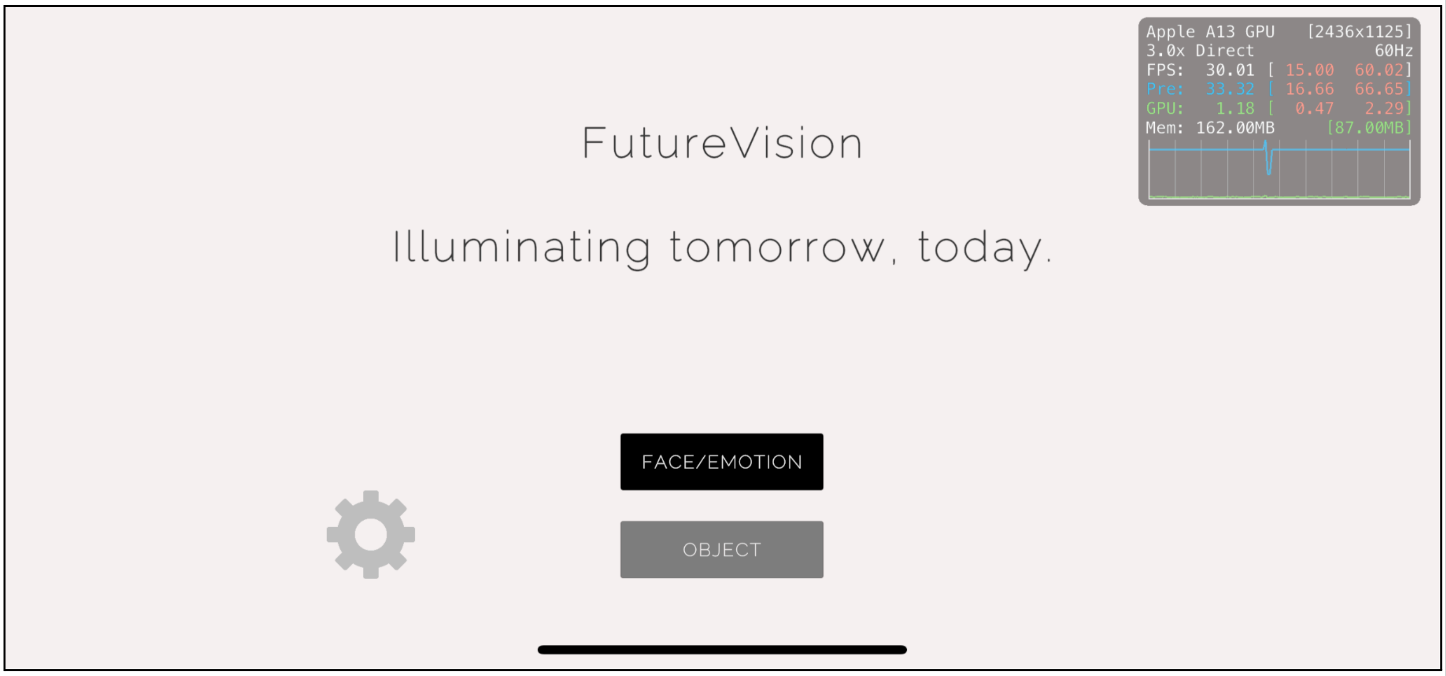
Reflecting on the journey from the previous iteration, where focus was on expanding the app's capabilities through the integration of advanced AI models and the introduction of features like an on-screen face counter, it's clear that each phase of development brings its own set of challenges and learning opportunities. While the shift to a pre-trained model from HuggingFace was a pragmatic decision in the face of technical hurdles, it also underscored the importance of flexibility and adaptability in leveraging existing technologies to advance our goals.

Looking ahead, the insights gained from addressing the technical and usability challenges in this iteration will inform my approach to future developments. The continued focus on refining and expanding the app's features, with an emphasis on improving stability, and accessibility, will ensure that development remains aligned with the overarching goal of creating a versatile and user-friendly platform.

Final Evaluation

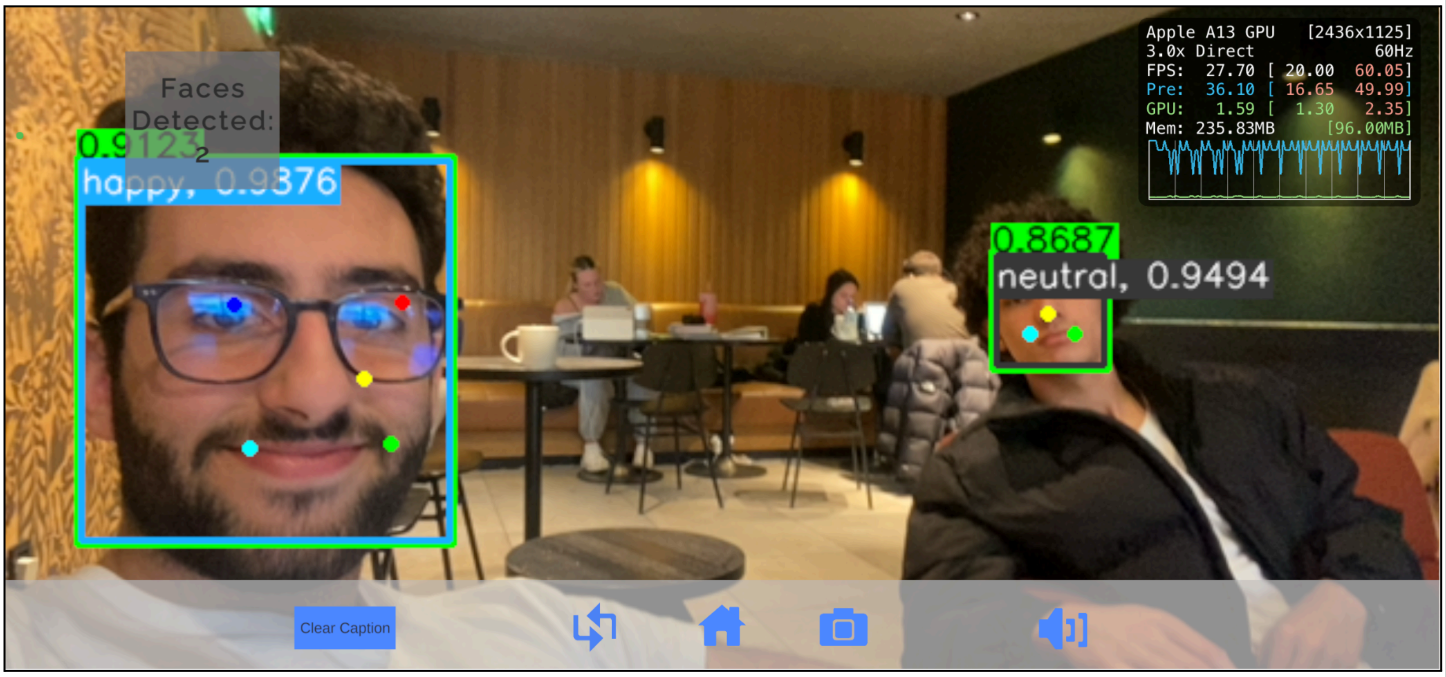
Final Testing Evidence: Functionality and Robustness

Post Development testing plan

Test No.	Aspect being tested/Link to success criteria	Justification	Importance?	How to perform test	Type of test (valid, invalid, boundary)	Expected result	Actual result	Success?
1	A title page/main menu is loaded Succcess criteria points #2 and #5 (User interface design) (Home screen)	Ensuring the title page/main menu loads correctly is crucial for providing users with a clear starting point and navigation options, enhancing their overall application experience.	🟩	Start the application	Valid	App displays the home splash screen	App displays the home splash screen	🟩
	Evidence:  FutureVision Illuminating tomorrow, today. FACE/EMOTION OBJECT Apple A13 GPU [2436x1125] 3.0x Direct 60Hz FPS: 30.01 [15.00 60.02] Pre: 33.32 [16.66 66.65] GPU: 1.18 [0.47 2.29] Mem: 162.00MB [87.00MB] *The app loads the main title page, displaying all the UI elements and text*							
2	Main menu should allow the user to navigate to 'emotion detection' scene Succcess criteria points #2, #5 and #11	Testing the main menu's navigation to the 'emotion detection' scene is essential for ensuring users can access core features seamlessly,	🟩	Press 'emotion detection' button in home screen	Valid	App displays the 'emotion detection' scene	App displays the 'emotion detection' scene	🟩

	(User interface design) (Home screen) (Ease of use)	improving usability and engagement with the application.	🟩					🟩

Evidence:

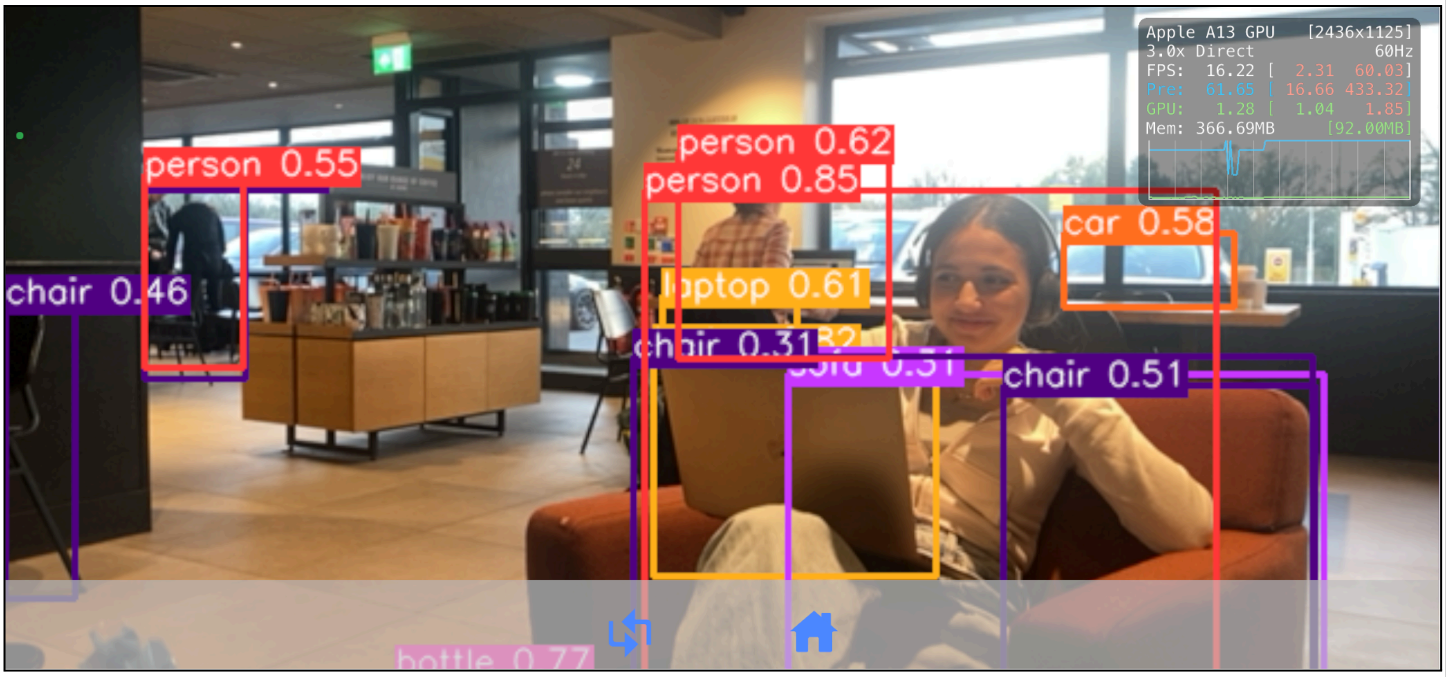


```
Apple A13 GPU    [2436x1125]
3.0x Direct            60Hz
FPS:   27.70 [ 20.00  60.05]
Pre:   36.10 [ 16.65  49.99]
GPU:    1.59 [  1.30   2.35]
Mem: 235.83MB      [96.00MB]
```

Emotion Detection scene is loaded when the user selects the button from the home screen, displaying the following UI and scene layout

3	Main menu should allow the user to navigate to 'object detection' scene Succcess criteria points #2, #5 and #11 (User interface design) (Home screen) (Ease of use)	Testing the main menu's navigation to the 'object detection' scene is essential for ensuring users can access core features seamlessly, improving usability and engagement with the application.	🟨	Press 'object detection' button in home screen	Valid	App displays the 'object detection' scene	App displays the 'object detection' scene	🟩

Evidence:

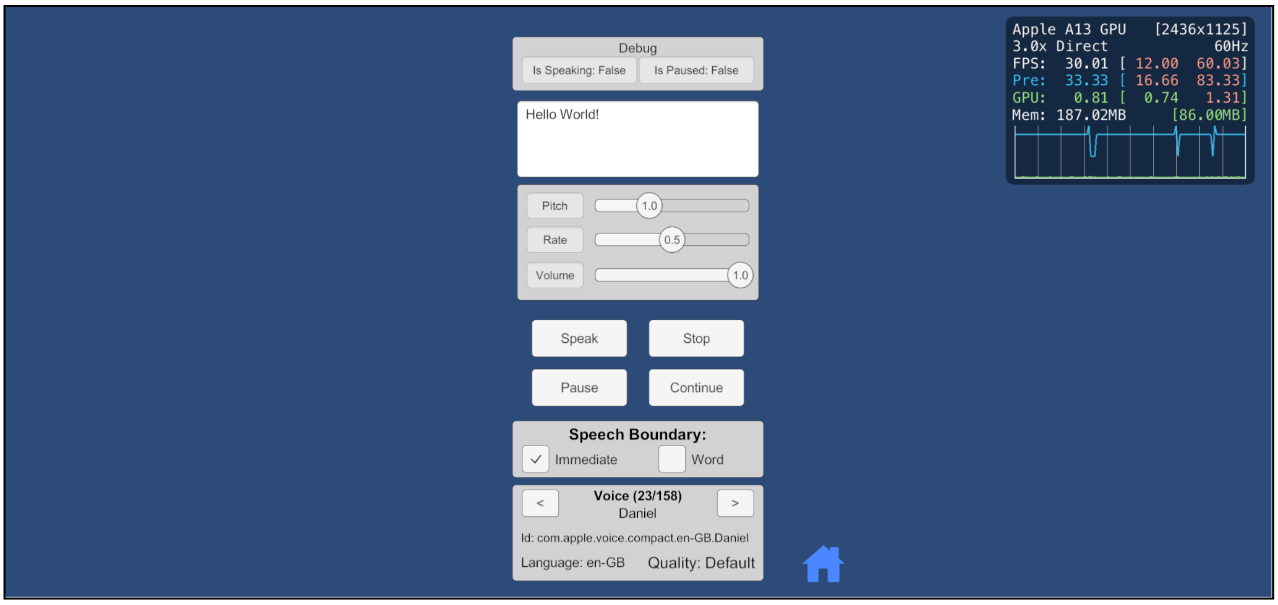


```
Apple A13 GPU    [2436x1125]
3.0x Direct            60Hz
FPS:   16.22 [  2.31  60.03]
Pre:   61.65 [ 16.66 433.32]
GPU:    1.28 [  1.04   1.85]
Mem: 366.69MB       [92.00MB]
```

Object Detection scene is loaded when the user selects the button from the home screen, displaying the following UI and scene layout

4	Main menu should allow the user to navigate to 'settings' scene Succcess criteria points #2, #5 and #11 (User interface design) (Home screen) (Ease of use)	Testing the main menu's navigation to the 'settings' scene is essential for ensuring users can access core features seamlessly, improving usability and engagement with the application.	🟩	Press 'settings' button in home screen	Valid	App displays the 'settings' scene	App displays the 'settings' scene	🟩

Evidence:



```
Apple A13 GPU    [2436x1125]
3.0x Direct            60Hz
FPS:   30.01 [ 12.00  60.03]
Pre:   33.33 [ 16.66  83.33]
GPU:    0.81 [  0.74   1.31]
Mem: 187.02MB       [86.00MB]
```

Debug
Is Speaking: False
Is Paused: False

Hello World!

Pitch 1.0
Rate 0.5
Volume 1.0

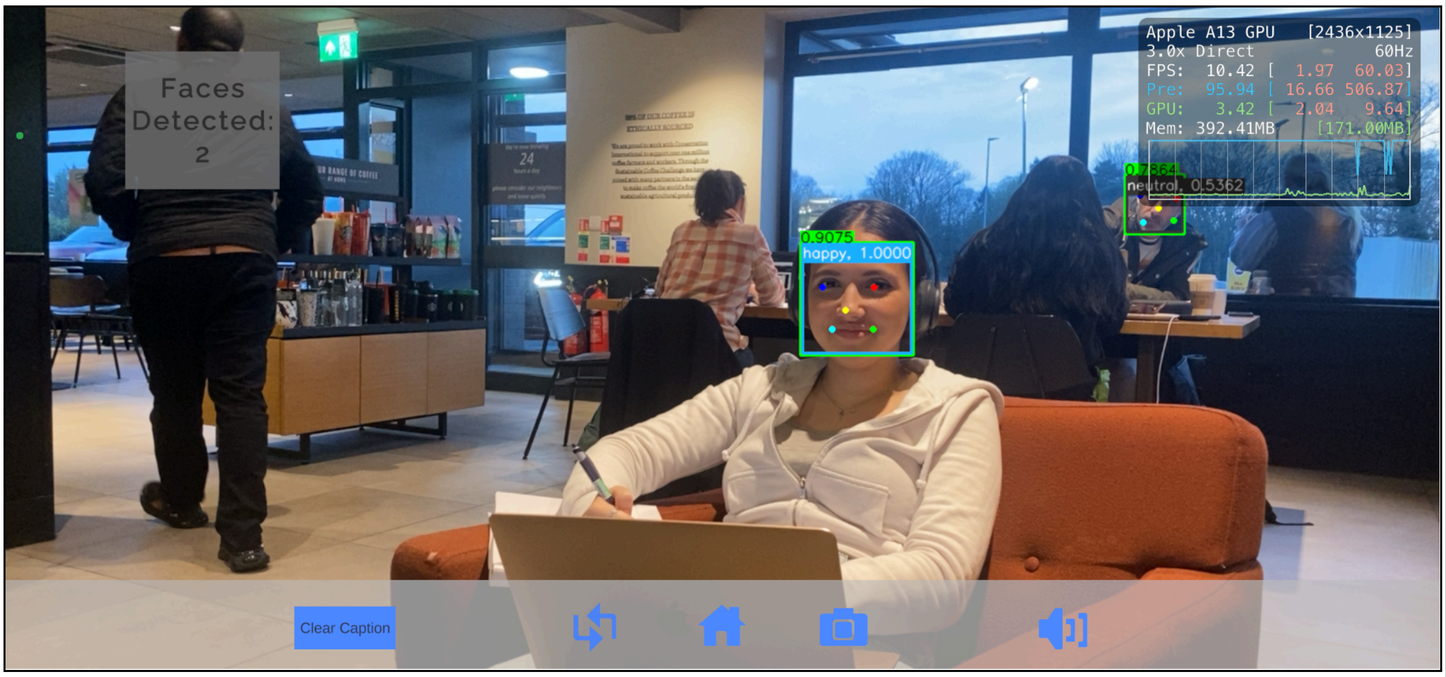
Speech Boundary:
[✓ Immediate] Word

Voice (23/158)
Daniel
Id: com.apple.voice.compact.en-GB.Daniel
Language: en-GB
Quality: Default

The Settings scene is loaded when the user selects the button from the home screen, displaying the following UI and scene layout

5	Camera view appears when in the ‘emotion detection’ scene Success criteria points #1 (Camera functionality when user opens app)	Verifying that the camera view appears in the 'emotion detection' scene is vital for enabling users to capture images in real-time, which is fundamental to the functionality and user interaction with the application.	🟩	Start the ‘emotion detection’ scene	Valid	App displays the output of the camera	App displays the output of the camera	🟩

Evidence:

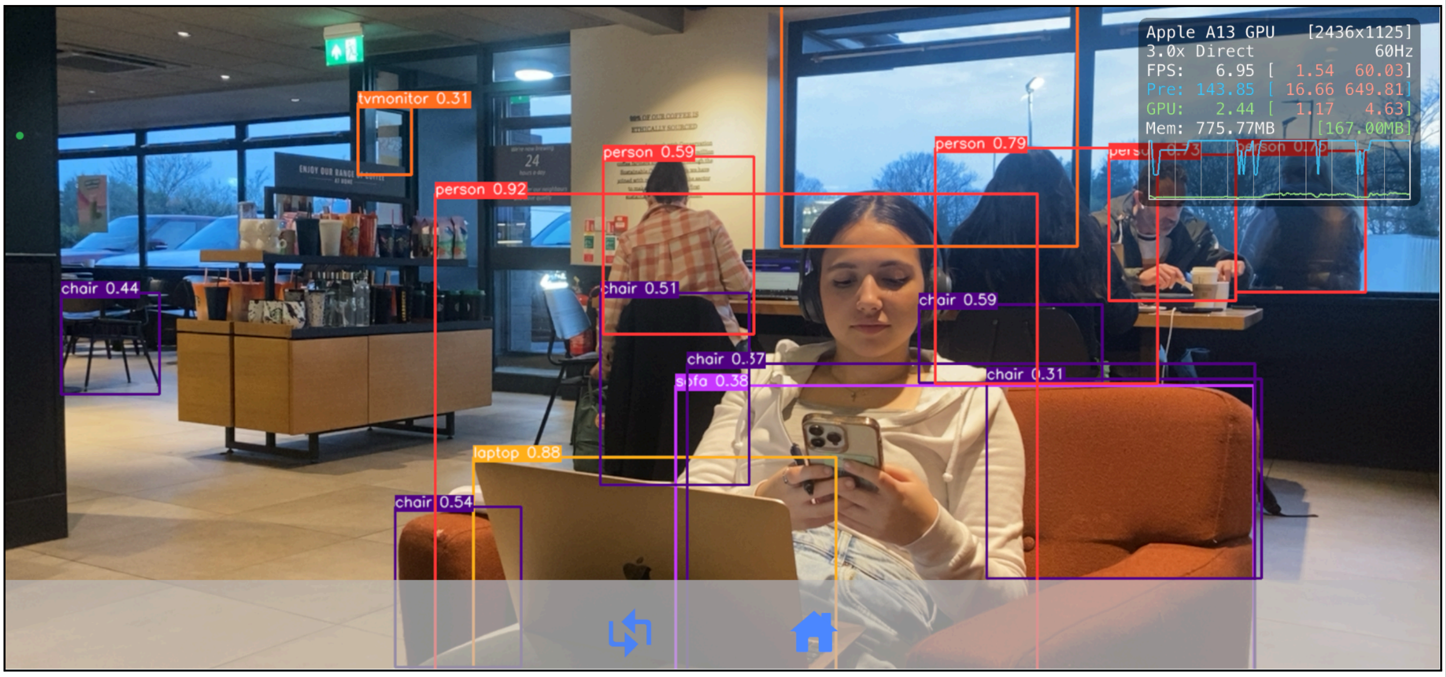


```
Apple A13 GPU    [2436x1125]
3.0x Direct            60Hz
FPS:   10.42 [   1.97  60.03]
Pre:   95.94 [  16.66 506.87]
GPU:    3.42 [   2.04   9.64]
Mem: 392.41MB       [171.00MB]
```

When loaded into the emotion detection scene, the camera view is seen perfectly, in high resolution.

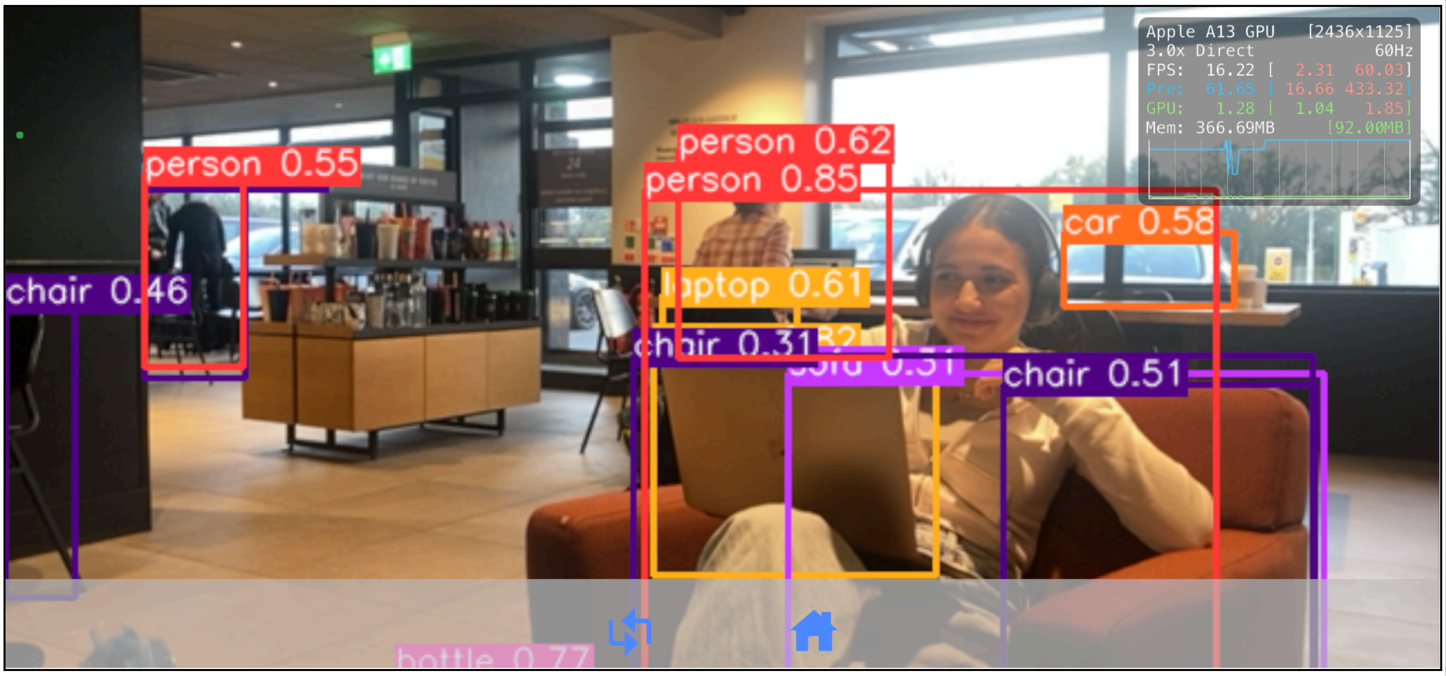
6	Camera functionality within the 'object detection' scene Success criteria points #1, #4 (Camera functionality when user opens app) (Object detection)	Verifying camera functionality and integration within the object detection scene to ensure that the app can effectively use the camera to identify and categorise objects in real-time, a core feature that must perform reliably for the app's intended purpose.	🟨	Start the ‘object detection’ scene	Valid	App displays the output of the camera	App displays the output of the camera	🟩

Evidence:



```
Apple A13 GPU   [2436x1125]
3.0x Direct           60Hz
FPS:    6.95 [  1.54  60.03]
Pre:  143.85 [ 16.66 649.81]
GPU:    2.44 [  1.17   4.63]
Mem: 775.77MB     [167.00MB]
```

When loaded into the emotion detection scene, the camera view is seen perfectly, in high resolution, however the FPS is very low (top right), due to the massive amount of processing needed to be done at a high resolution.



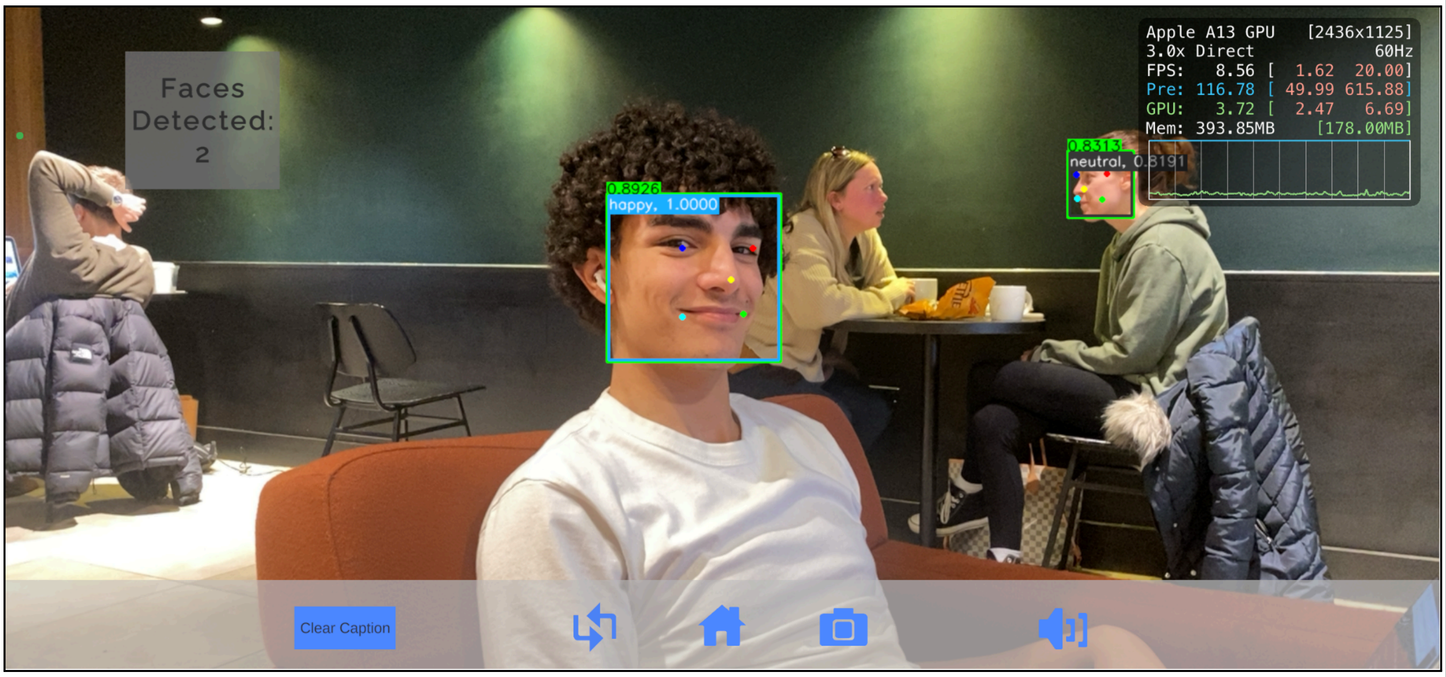
```
Apple A13 GPU   [2436x1125]
3.0x Direct           60Hz
FPS:   16.22 [  2.31  60.03]
Pre:   61.65 [ 16.66 433.32]
GPU:    1.28 [  1.04   1.85]
Mem: 366.69MB      [92.00MB]
```

When I reduce the resolution of the camera, the FPS more than doubles and the memory more than halves, which is critical to smooth operation of my application.

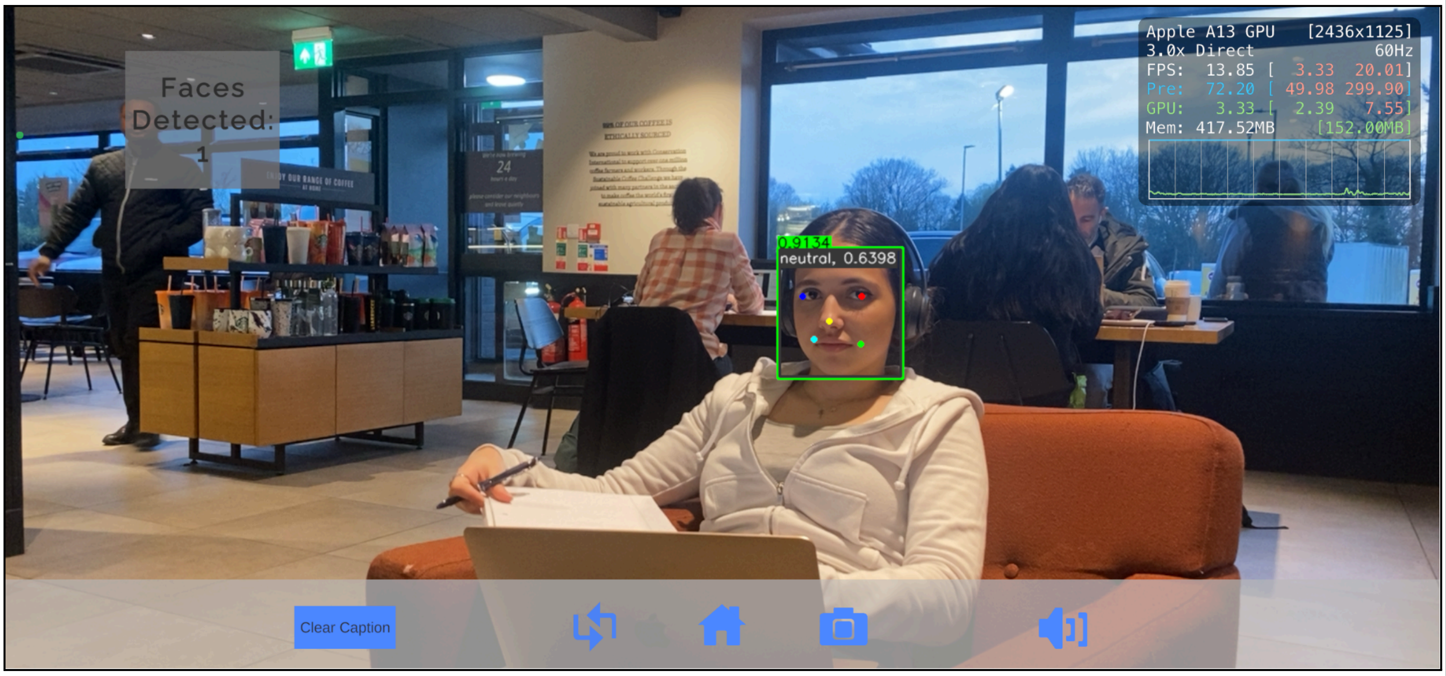
7	Display of facial recognition feature in the 'emotion detection' scene Success criteria points #1, #3, #6	Demonstrating the display and functionality of the facial recognition feature in the emotion detection scene to ensure it	🟩	Start the 'emotion detection' scene and point camera at someone's face.	Valid	App displays a bounding box around faces and prints the detected emotion close the them.	App displays a bounding box around faces and prints the detected emotion close the them.	🟩

	(Camera functionality when user opens app) (Identification of people in camera's view/facial recognition) (Mood recognition based on facial expressions)	operates correctly within this context, accurately identifying and analysing facial expressions for mood assessment.	🟩					🟩

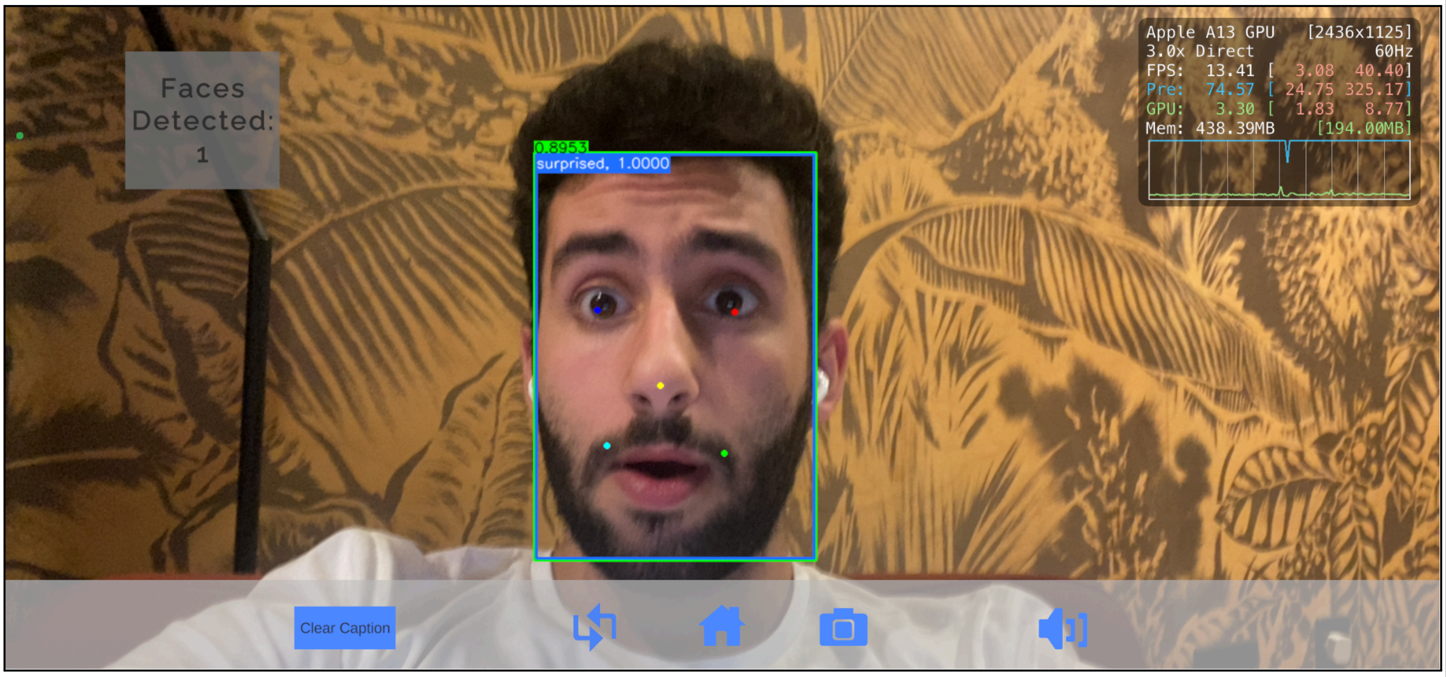
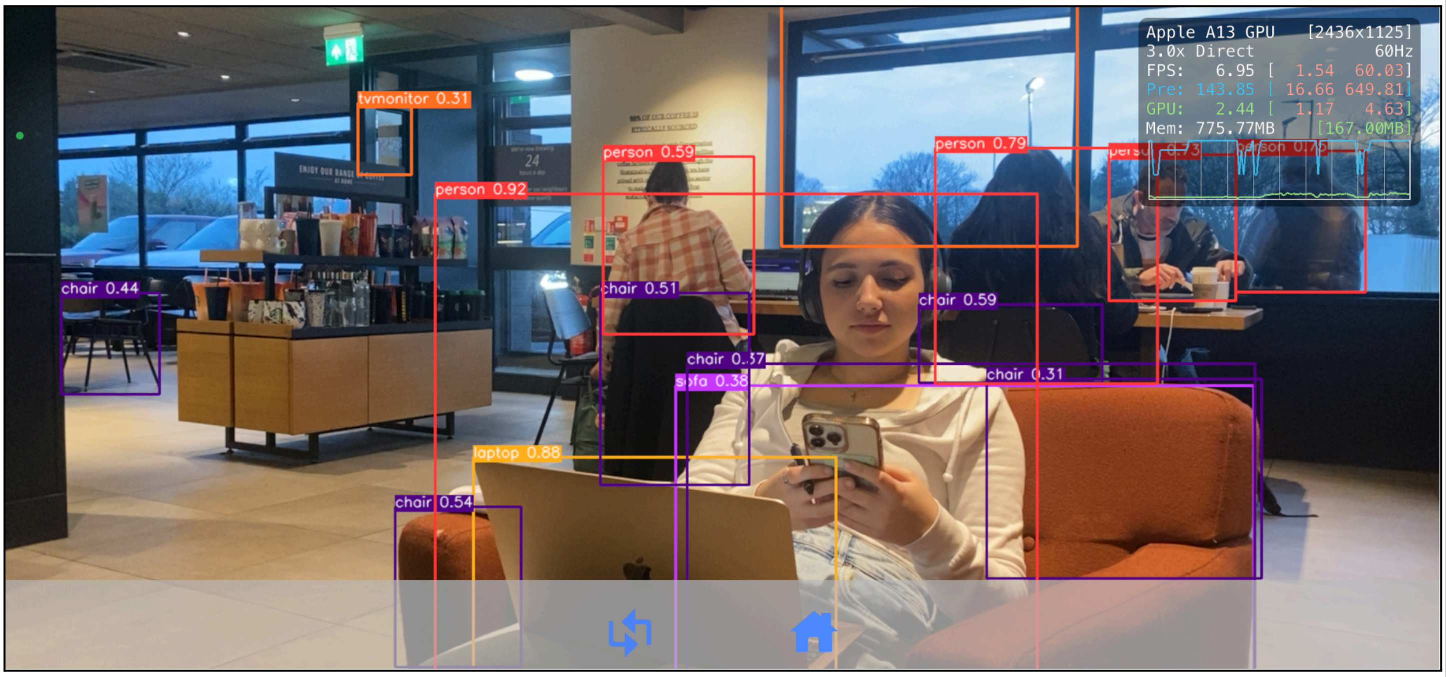
Evidence:



```
Apple A13 GPU    [2436x1125]
3.0x Direct             60Hz
FPS:    8.56 [  1.62  20.00]
Pre:  116.78 [ 49.99 615.88]
GPU:    3.72 [  2.47   6.69]
Mem: 393.85MB     [178.00MB]
```

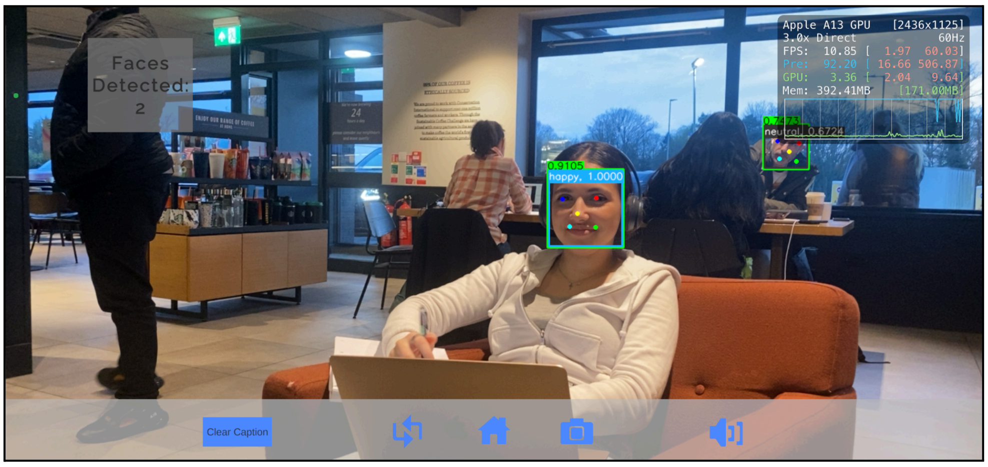


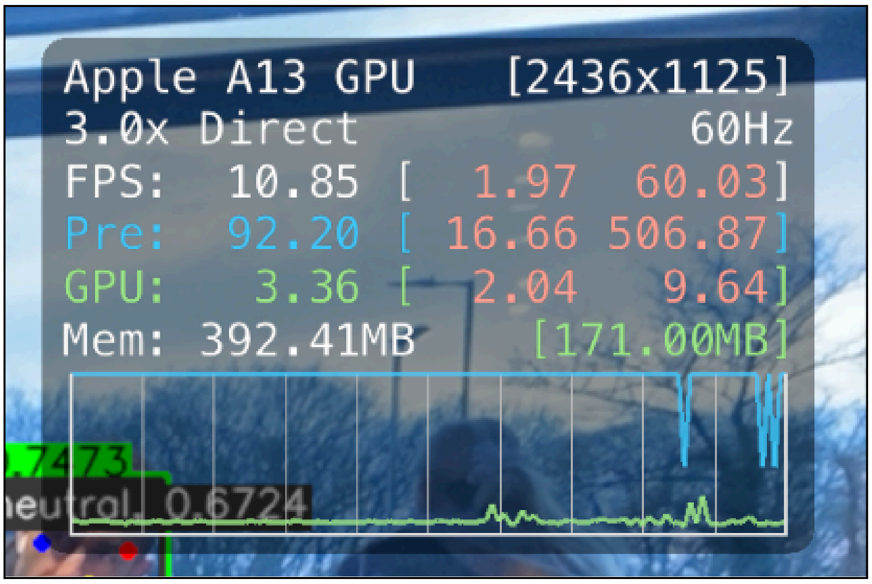
```
Apple A13 GPU    [2436x1125]
3.0x Direct             60Hz
FPS:   13.85 [  3.33  20.01]
Pre:   72.20 [ 49.98 299.90]
GPU:    3.33 [  2.39   7.55]
Mem: 417.52MB     [152.00MB]
```

	 Faces Detected: 1 0.8953 surprised, 1.0000 Clear Caption `Apple A13 GPU [2436x1125]` `3.0x Direct 60Hz` `FPS: 13.41 [ 3.08 40.40]` `Pre: 74.57 [ 24.75 325.17]` `GPU: 3.30 [ 1.83 8.77]` `Mem: 438.39MB [194.00MB]` *Different facial expressions are recognised and displayed on screen*							
8	Objects listed in the 'object detection' scene Success criteria points #4 (Object detection)	Turn on an test the object detection scene to verify that the feature works seamlessly, enhancing the educational and accessibility aspects of the app.	🟨	Press ‘object detection’ button on homescreen	Valid	Object detection scene loads and displays object names besides their bounding boxes.	Object detection scene loads and displays object names besides their bounding boxes.	🟩
	Evidence:  tvmonitor 0.31 person 0.59 person 0.92 person 0.79 chair 0.44 chair 0.51 chair 0.59 chair 0.37 sofa 0.38 chair 0.31 laptop 0.88 chair 0.54 `Apple A13 GPU [2436x1125]` `3.0x Direct 60Hz` `FPS: 6.95 [ 1.54 60.03]` `Pre: 143.85 [ 16.66 649.81]` `GPU: 2.44 [ 1.17 4.63]` `Mem: 775.77MB [167.00MB]` *Different objects are displayed with their names and confidence rating besides the name*							

9	Responsiveness and fluidity of the live camera feed Success criteria points #1, #9 (Camera functionality when user opens app) (Responsiveness - 20fps consistent feed)	Assessing the responsiveness and fluidity of the live camera feed for real-time interaction to ensure the application can handle streaming video at a consistent frame rate, critical for a smooth and engaging user experience in features like live emotion detection or object recognition.	🟩	Starts either the 'emotion detection' or 'object detection' scene, as both use the same logic for the live camera feed	Valid	Camera is fluid and responsive ~20 fps	Camera is fluid and responsive ~20 fps	🟨

Evidence:





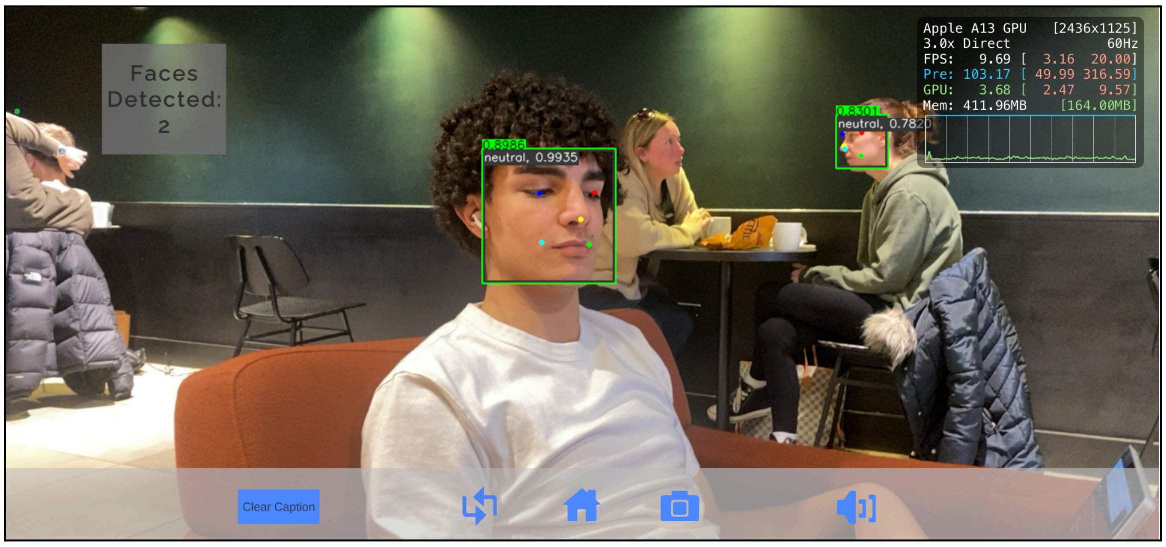
```
Apple A13 GPU    [2436x1125]
3.0x Direct             60Hz
FPS:   10.85 [  1.97  60.03]
Pre:   92.20 [ 16.66 506.87]
GPU:    3.36 [  2.04   9.64]
Mem:  392.41MB    [171.00MB]
```

When maximising the camera resolution, the application runs at approx 10-15fps which is less than my target 20fps. I found through testing that if I reduce the camera quality, I can acheive between 20-30fps, providing a far smoother experience, but I also found that this reduces the accuracy in expression detection and makes application quote unattractive due to the contrast in resolution between UI and camera.

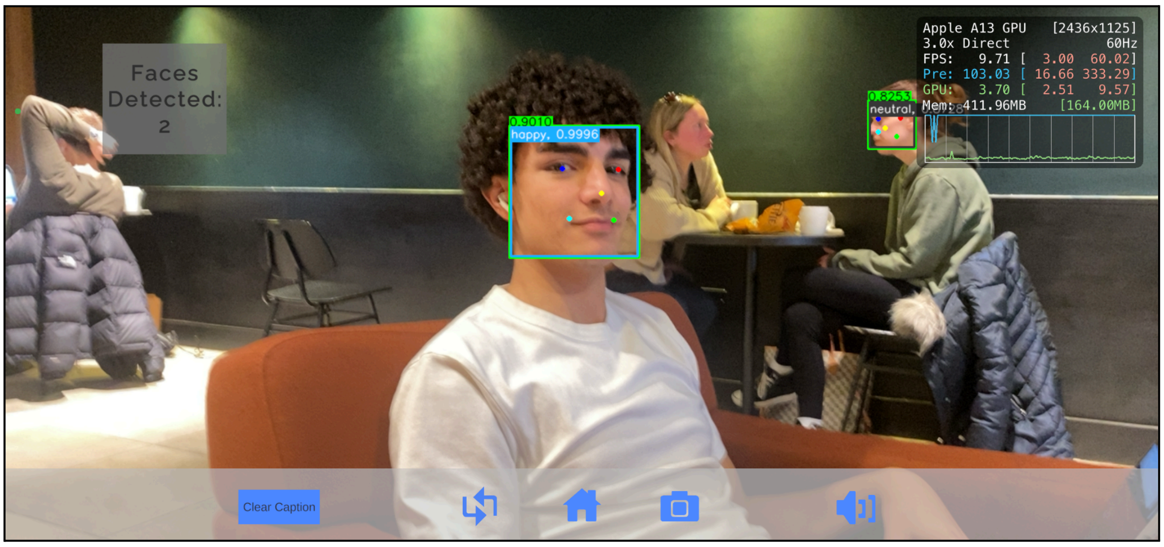
10	Accuracy of facial expression analysis for mood recognition	Validating the accuracy of facial expression analysis for mood	🟩	Start 'emotion detection' scene and point camera at	Valid	'Mood' text updates in real time and accurately	'Mood' text updates in real time and accurately	🟩

	Success criteria points #3, #6, #10 (Identification of people in camera's view/facial recognition) (Mood recognition based on facial expressions) (Accuracy)	recognition to ensure the technology correctly interprets a wide range of human emotions, pivotal for applications relying on emotional intelligence for interaction or feedback.	🟩	someone displaying different moods		detects a person's emotion	detects a person's emotion	🟩

Evidence:



```
Apple A13 GPU   [2436x1125]
3.0x Direct          60Hz
FPS:    9.69 [  3.16  20.00]
Pre:  103.17 [ 49.99 316.59]
GPU:    3.68 [  2.47   9.57]
Mem: 411.96MB     [164.00MB]
```



```
Apple A13 GPU   [2436x1125]
3.0x Direct          60Hz
FPS:    9.71 [  3.00  60.02]
Pre:  103.03 [ 16.66 333.29]
GPU:    3.70 [  2.51   9.57]
Mem: 411.96MB     [164.00MB]
```

These 2 images were taken split seconds apart, displaying the real-time change in emotion detection, immediately switching from 'neutral' to 'happy'

11	User interface intuitiveness and simplicity	Ensuring the user interface is intuitive and simple, facilitating ease of	🟩	Start the application and look at the UI elements on	Valid	UI elements appear on screen. The UI elements	UI elements appear on screen. The UI elements	🟩

	Success criteria points #2, #11 (User interface design) (Ease of use)	use for a wide demographic of users, including those who may not be tech-savvy, to reduce the learning curve and enhance the overall user engagement with the app.	🟩	screen		should be intuitive and simple to follow from the point of view of someone who may not be confident using tech.	should be intuitive and simple to follow from the point of view of someone who may not be confident using tech.	🟩

Evidence:

Home screen buttons

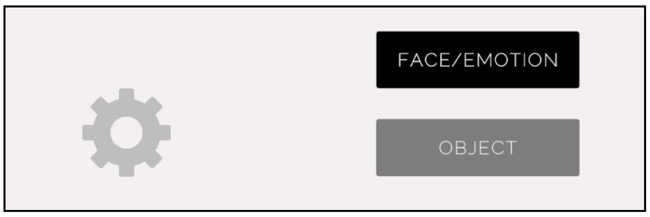
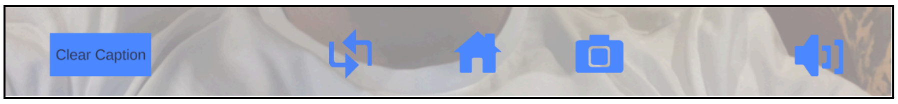
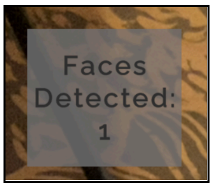


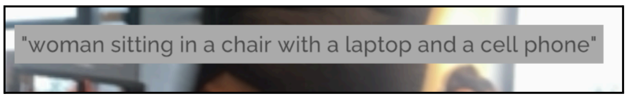
Image captioning button list



Face count popup



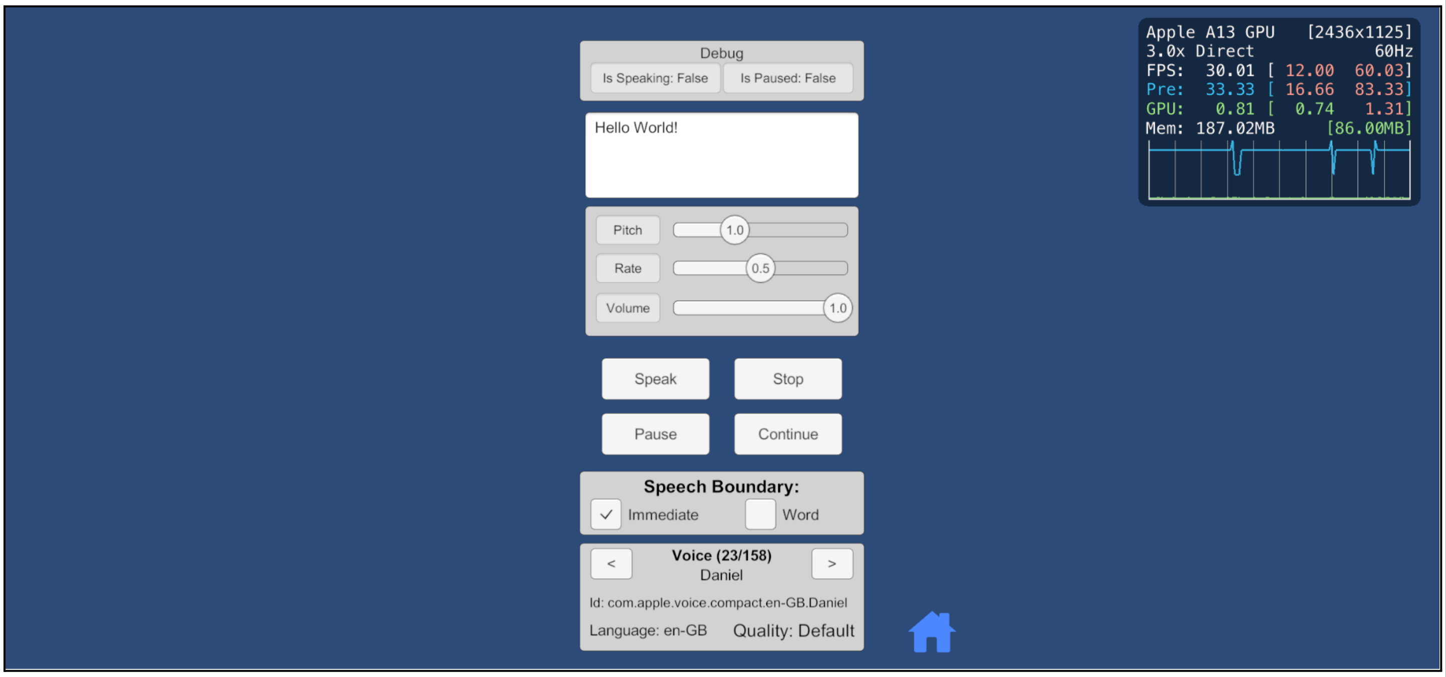
Generated image caption


"woman sitting in a chair with a laptop and a cell phone"

According to my stakeholders and general feedback, the UI and feel of the project seems to be intuitive and simple to follow, providing the user with a pleasant experience, allowing many different demographics to enjoy the app as they please, without confusion.

12	Customization of text-to-speech settings in the 'settings' scene Success criteria points #7, #12 (Text-to-speech output for processed attributes) (Variable text-to-speech voice pitch/volume/speed)	Customising text-to-speech settings in the 'settings' scene to validate the range of personalisation options available to users, ensuring they can adjust voice pitch, volume, and speed to suit their preferences and needs, enhancing accessibility and user experience.	🟨	Vary the sliders and voice carousel in the settings page until preferred TTS voice is as desired	Valid	User is able to change the voice, volume, pitch and speed of TTS speech.	User is able to change the voice, volume, pitch and speed of TTS speech.	🟩

Evidence:



```
Apple A13 GPU    [2436x1125]
3.0x Direct             60Hz
FPS:   30.01 [ 12.00   60.03]
Pre:   33.33 [ 16.66   83.33]
GPU:    0.81 [  0.74    1.31]
Mem:  187.02MB       [86.00MB]
```

Debug

Is Speaking: False | Is Paused: False

Hello World!

Pitch 1.0

Rate 0.5

Volume 1.0

Speech Boundary: ✓ Immediate | Word

Voice (23/158) Daniel

Id: com.apple.voice.compact.en-GB.Daniel

Language: en-GB Quality: Default

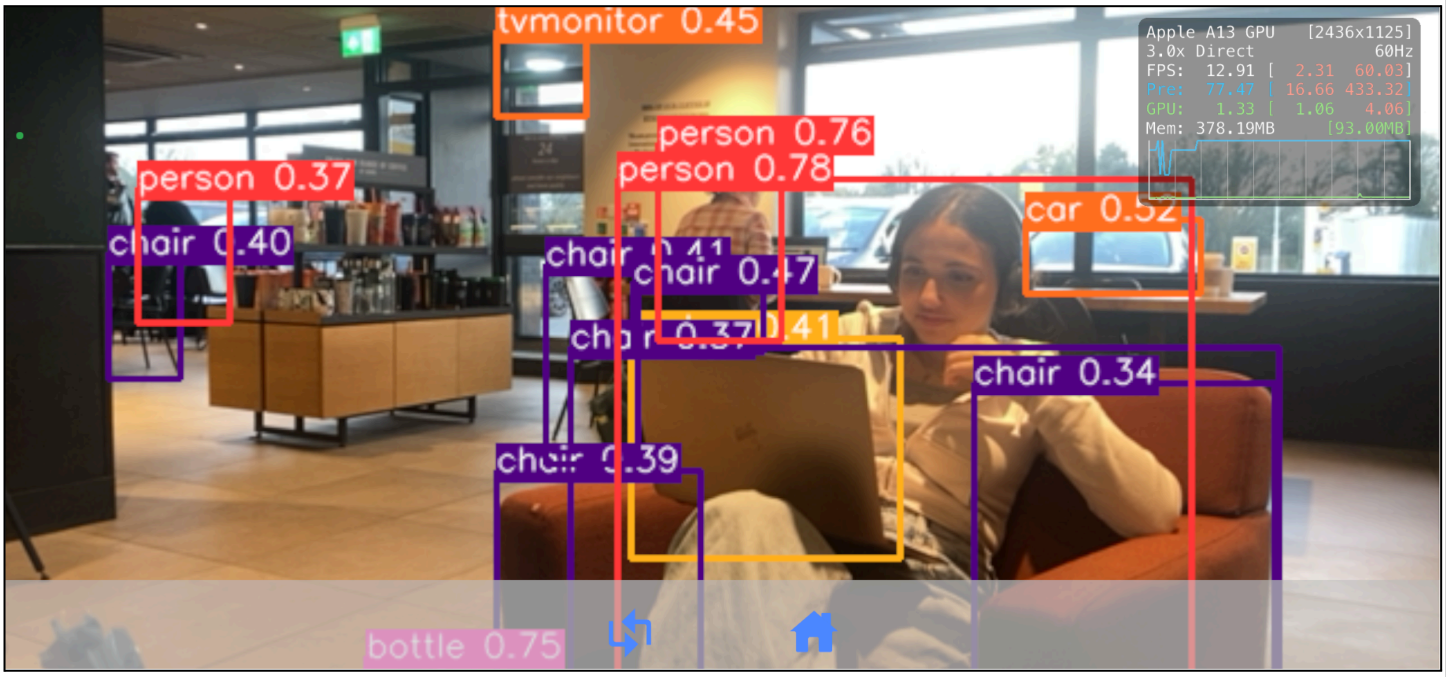
Although the UI for this page may not follow a similar theme to the rest of the project, the functionality remains perfect. The user is able to adjust the TTS voice as they please.

13	Object detection accuracy and performance Success criteria points #4, #10 (Object detection) (Accuracy)	Checking the accuracy and performance of object detection under a variety of conditions to validate the technology's ability to recognise and categorise objects reliably, which is central to the app's functionality and user engagement.	🟨	Turn on 'object detection' scene and evaluate performance when pointing at many objects in different environments	Valid	Object detection should be accurate and remain highly responsive, even when many objects are in view	Object detection drastically slows down when many objects are in view	🟨

Evidence:

Low resolution object detection (10-15fps), (300-500MB memory)

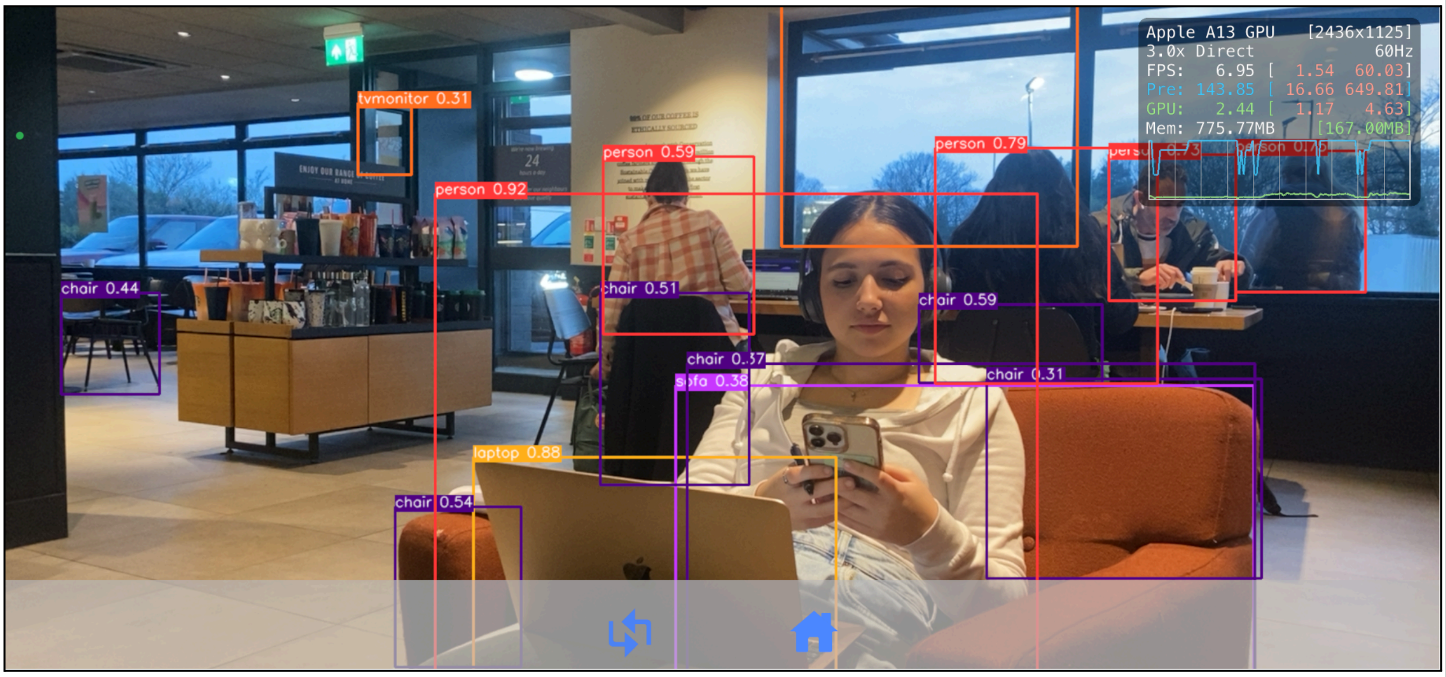
- *This project aims to be aesthetically pleasing and seamless. Having a low resolution output defeats this purpose*



```
Apple A13 GPU   [2436x1125]
3.0x Direct            60Hz
FPS:  12.91 [  2.31  60.03]
Pre:  77.47 [ 16.66 433.32]
GPU:   1.33 [  1.06   4.06]
Mem: 378.19MB     [93.00MB]
```

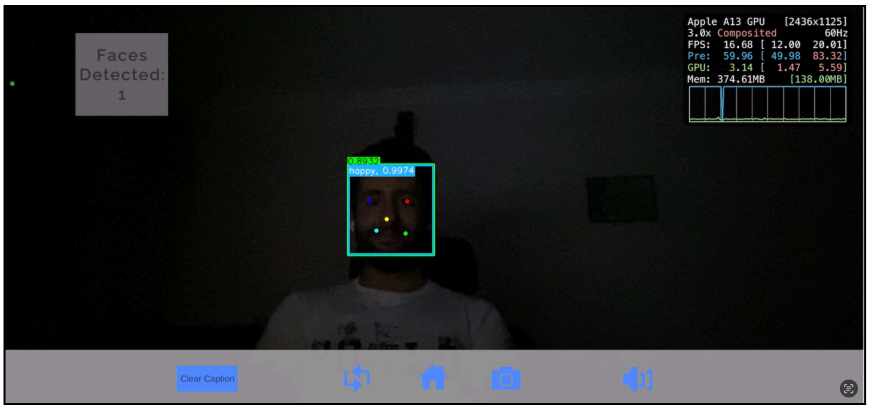
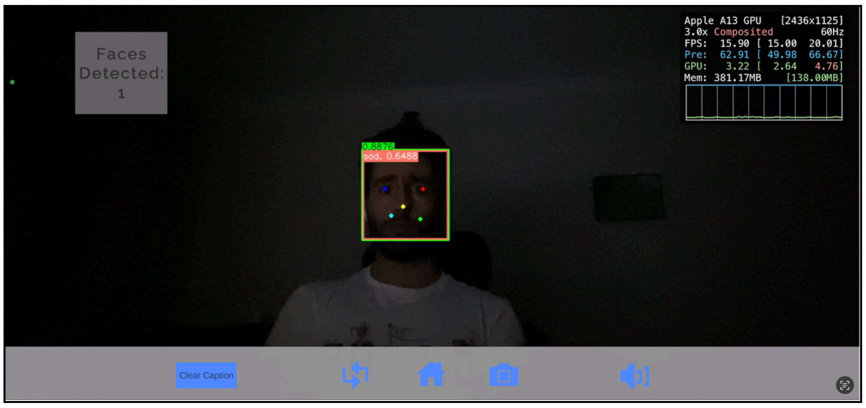
High resolution object detection (5-8fps), (500-800MB memory)

- *Although much better to work with, providing higher accuracy and nicer to look at, using a higher resolution for object detection is highly CPU intensive, causing the phone to dramatically heat up and slow down. FPS is low and memory usage is very high.*

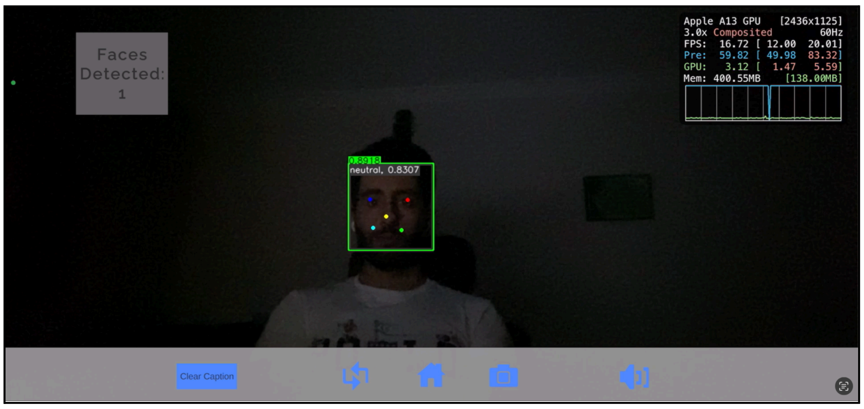


```
Apple A13 GPU   [2436x1125]
3.0x Direct            60Hz
FPS:   6.95 [  1.54  60.03]
Pre: 143.85 [ 16.66 649.81]
GPU:   2.44 [  1.17   4.63]
Mem: 775.77MB    [167.00MB]
```

14	Responsiveness of the main menu and navigation elements Success criteria points #2, #9, #11	Testing the responsiveness of the main menu and navigation elements to ensure that users can quickly and	🟩	Press different buttons and evaluate the response time and loading time of different	Valid	Application should respond to user inputs quickly and with minimal	Application responds to user inputs quickly and with minimal delay	🟩

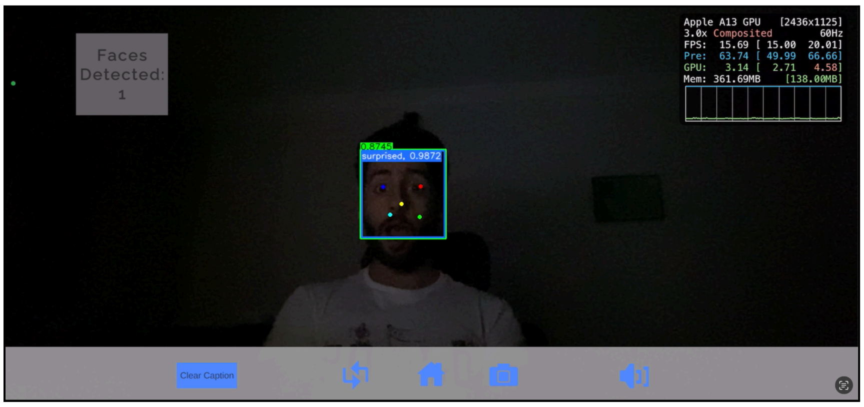
	(User interface design) (Responsiveness - 20fps consistent feed) (Ease of use)	easily access different parts of the application without frustration, critical for usability and user satisfaction.	🟩	scenes and processing.		delay		🟩
	Evidence: Screencast							
15	Evaluation of mood recognition under various lighting conditions Success criteria points #6, #10 (Mood recognition based on facial expressions) (Accuracy)	Evaluating the accuracy of mood recognition under various lighting conditions to ensure the technology is versatile and reliable in different environments, essential for a feature that relies on visual cues for emotion detection.	🟩	Attempt to identify different moods in low light, normal light, and very bright light. Evaluate if the AI is able to distinguish between moods such as happy/sad/angr y	Boundary	In low light, emotion recognition may fail or output incorrect emotions due to a lack of clarity in the image	In low light, emotion recognition may fail or output incorrect emotions due to a lack of clarity in the image	🟩
	Evidence: *Low lighting (attempts 'Happy') (success)*  Apple A13 GPU [2436x1125] 3.0x Composited 60Hz FPS: 16.68 [12.00 20.01] Pre: 59.96 [49.98 83.32] GPU: 3.14 [1.47 5.59] Mem: 374.61MB [138.00MB] 0.8932 happy, 0.9974 *Low lighting (attempts 'Sad') (success)*  Apple A13 GPU [2436x1125] 3.0x Composited 60Hz FPS: 15.90 [15.00 20.01] Pre: 62.91 [49.98 66.67] GPU: 3.22 [2.64 4.76] Mem: 381.17MB [138.00MB] 0.8876 sad, 0.6488 							

Low lighting (attempts 'Neutral') (success)



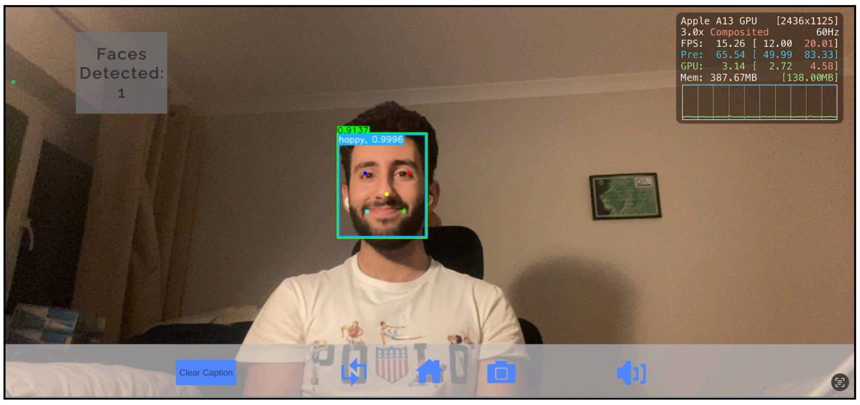
```
Apple A13 GPU   [2436x1125]
3.0x Composited        60Hz
FPS:  16.72 [ 12.00  20.01]
Pre:  59.82 [ 49.98  83.32]
GPU:   3.12 [  1.47   5.59]
Mem: 400.55MB    [138.00MB]
```

Low lighting (attempts 'Surprised') (success)



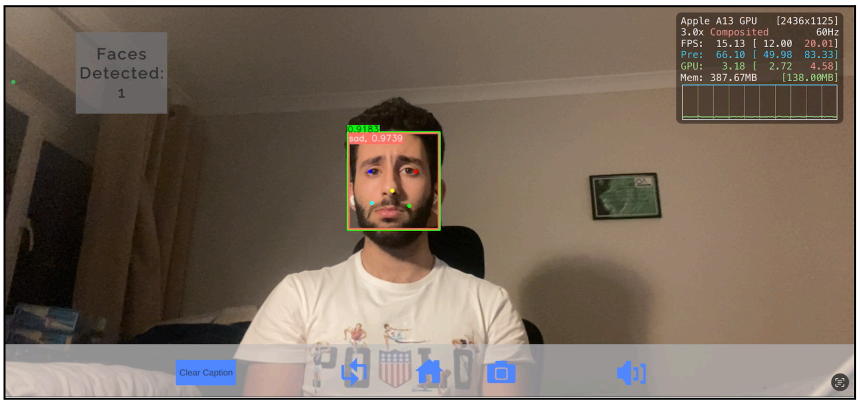
```
Apple A13 GPU   [2436x1125]
3.0x Composited        60Hz
FPS:  15.69 [ 15.00  20.01]
Pre:  63.74 [ 49.99  66.66]
GPU:   3.14 [  2.71   4.58]
Mem: 361.69MB    [138.00MB]
```

Normal lighting (attempts 'Happy') (success)



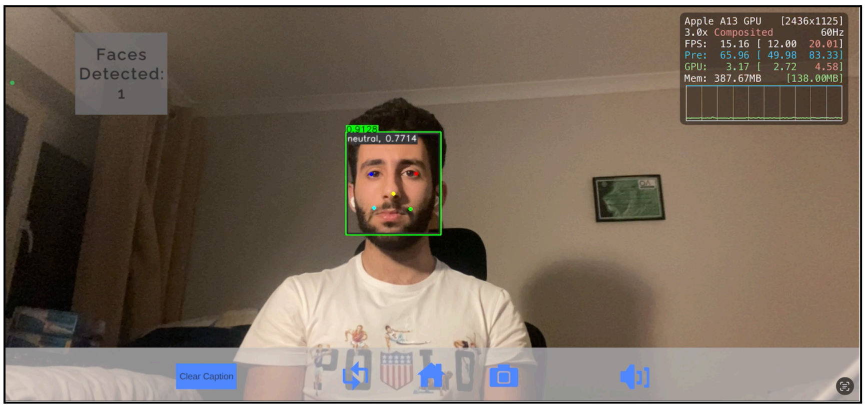
```
Apple A13 GPU   [2436x1125]
3.0x Composited        60Hz
FPS:  15.26 [ 12.00  20.01]
Pre:  65.54 [ 49.99  83.33]
GPU:   3.14 [  2.72   4.58]
Mem: 387.67MB    [138.00MB]
```

Normal lighting (attempts 'Sad') (success)



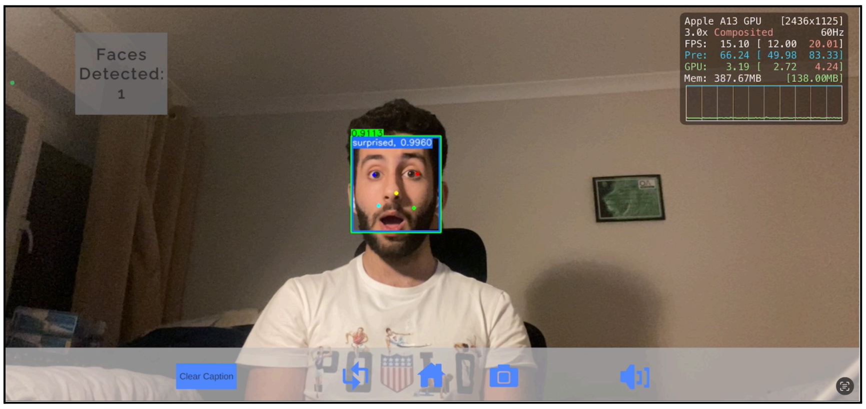
```
Apple A13 GPU   [2436x1125]
3.0x Composited        60Hz
FPS:  15.13 [ 12.00  20.01]
Pre:  66.10 [ 49.98  83.33]
GPU:   3.18 [  2.72   4.58]
Mem: 387.67MB    [138.00MB]
```

Normal lighting (attempts 'Neutral') (success)



```
Apple A13 GPU   [2436x1125]
3.0x Composited        60Hz
FPS:  15.16 [ 12.00  20.01]
Pre:  65.96 [ 49.98  83.33]
GPU:   3.17 [  2.72   4.58]
Mem: 387.67MB    [138.00MB]
```

Normal lighting (attempts 'Surprised') (success)

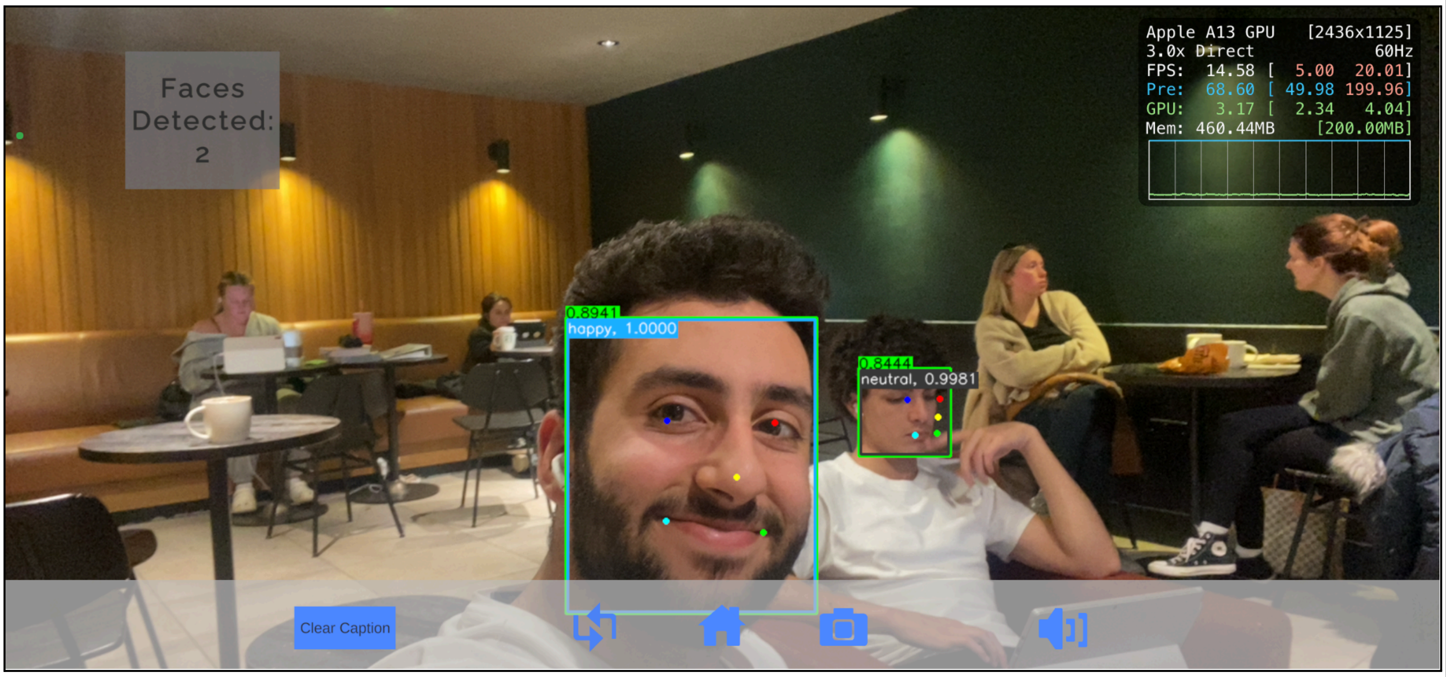


```
Apple A13 GPU   [2436x1125]
3.0x Composited        60Hz
FPS:  15.10 [ 12.00  20.01]
Pre:  66.24 [ 49.98  83.33]
GPU:   3.19 [  2.72   4.24]
Mem: 387.67MB    [138.00MB]
```

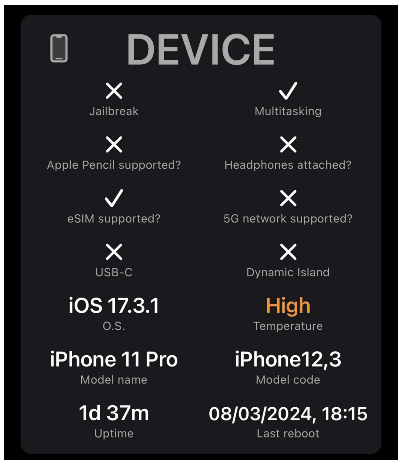
16	Ensuring app stability and performance during extended use Success criteria points #9, #11 (Responsiveness - 20fps consistent feed) (Ease of use)	Monitoring the application's stability and performance during extended use to identify any potential memory leaks, slowdowns, or crashes that could detract from the user experience, aiming for robustness and reliability over time.	🟩	Run the application for a long time, preferably over 10 minutes, with debugging information on screen, monitoring device temperature, memory usage and battery usage.	Boundary	Application should manage resources efficiently, without any memory leaks or bottlenecks.	iPhone gets hot after approx. 2-4 mins of usage. Battery is drained quickly. High memory consumption	🟥

Evidence:

Normal useage in emotion detection mode (no captioning/object detection)

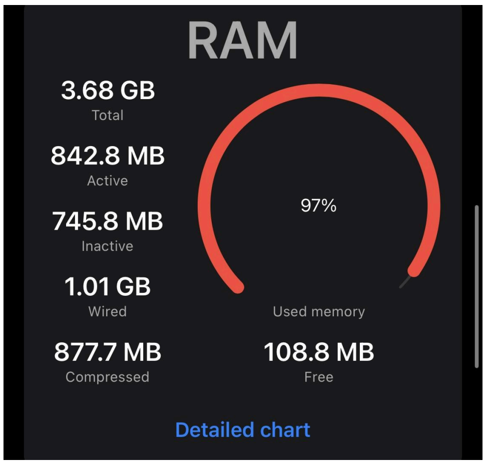


```
Apple A13 GPU    [2436x1125]
3.0x Direct             60Hz
FPS:  14.58 [  5.00  20.01]
Pre:  68.60 [ 49.98 199.96]
GPU:   3.17 [  2.34   4.04]
Mem: 460.44MB     [200.00MB]
```



DEVICE

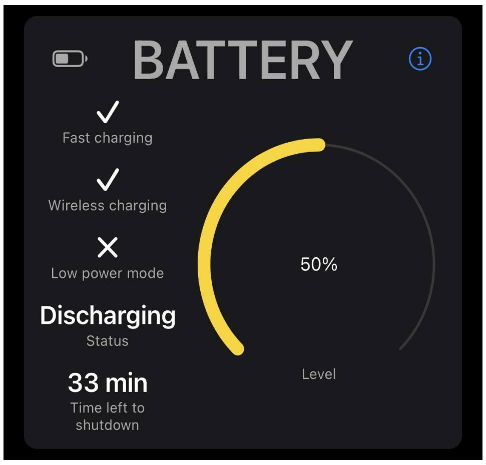
✗ Jailbreak	✓ Multitasking
✗ Apple Pencil supported?	✗ Headphones attached?
✓ eSIM supported?	✗ 5G network supported?
✗ USB-C	✗ Dynamic Island
iOS 17.3.1 O.S.	High Temperature
iPhone 11 Pro Model name	iPhone12,3 Model code
1d 37m Uptime	08/03/2024, 18:15 Last reboot



RAM

3.68 GB	Total
842.8 MB	Active
745.8 MB	Inactive
1.01 GB	Wired
877.7 MB	Compressed
97%	Used memory
108.8 MB	Free

Detailed chart



BATTERY

✓	Fast charging
✓	Wireless charging
✗	Low power mode
Discharging	Status
33 min	Time left to shutdown
50%	Level

After using the application in emotion detection mode for about 2-3 minutes, as seen by the device sensors above, the device temperature is high, RAM usage is extremely high, and the battery, with 50% remaining is estimating to only last another 33 minutes, when usually this would last at least 1-2 hours.

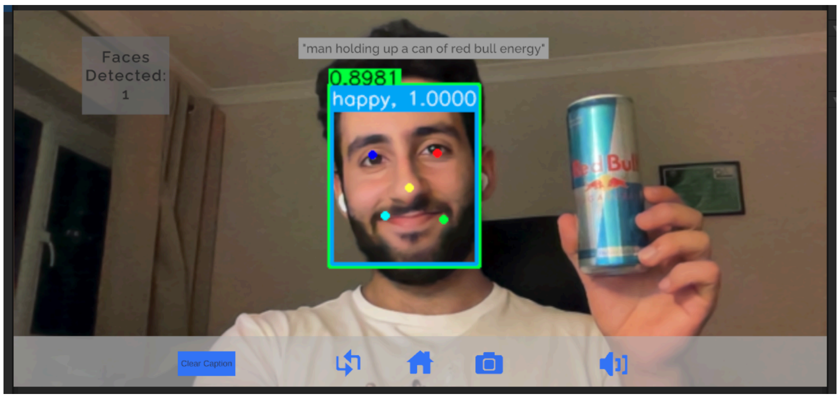
This app seems to be consuming large amounts of power. This could be since all the AI processing for emotion detection and object detection are done on the device itself. It could also be the constant pinging to Google's server, ensuring internet connectivity. There should be a more intuitive way to detect internet connectivity than simply pinging an IP address.

17	Accuracy of text captions generated for captured images Success criteria points #8, #10	Assessing the accuracy of text captions generated for captured images to ensure they are contextually relevant	🟩	Take a wide variety of different images in various environments, evaluating the	Valid	Captions should remain fairly accurate, describing a wide range of given	Captions are fairly accurate, describing a wide range of given scenarios.	🟩

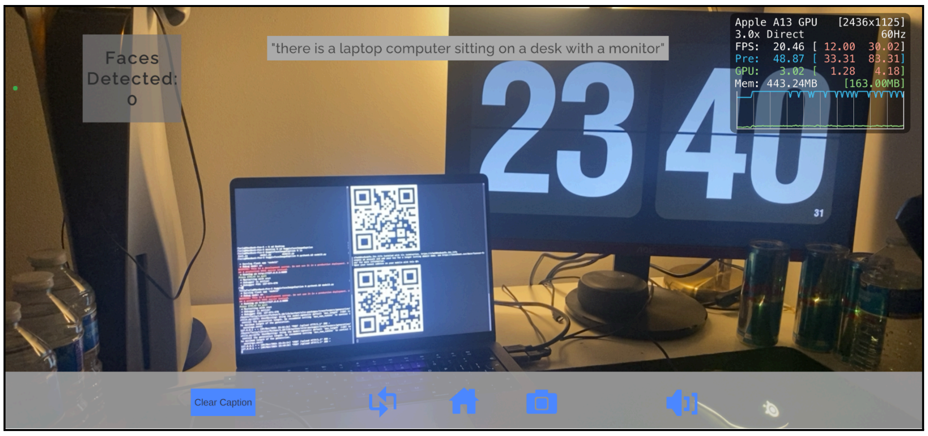
	(Image capture and captioning) (Accuracy)	and precise, enhancing the value of the app's image recognition and captioning capabilities for educational, accessibility, or entertainment purposes.	🟩	response and accuracy of the captions		scenarios.		🟩

Evidence:

Caption states 'man holding up a can of red bull energy'

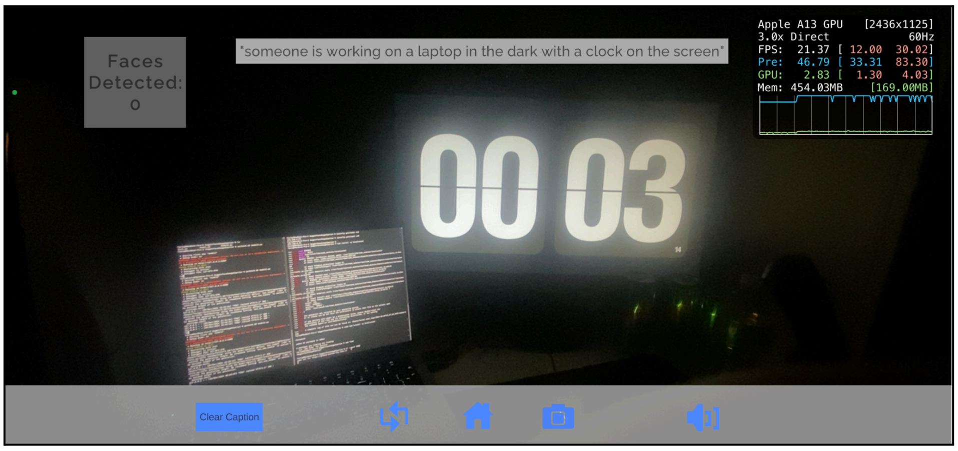


Caption states 'there is a laptop computer sitting on a desk with a monitor'


"there is a laptop computer sitting on a desk with a monitor"
Faces Detected: 0
Clear Caption

```
Apple A13 GPU   [2436x1125]
3.0x Direct            60Hz
FPS:  20.46 [ 12.00  30.02]
Pre:  48.87 [ 33.31  83.31]
GPU:   3.02 [  1.28   4.18]
Mem: 443.24MB    [163.00MB]
```

Caption states 'someone is working on a laptop in the dark with a clock on the screen'

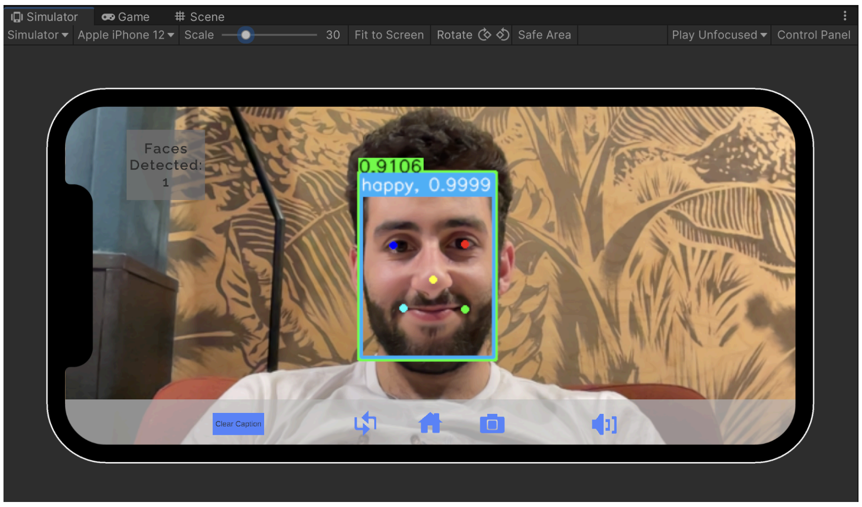

"someone is working on a laptop in the dark with a clock on the screen"
Faces Detected: 0
Clear Caption

```
Apple A13 GPU   [2436x1125]
3.0x Direct            60Hz
FPS:  21.37 [ 12.00  30.02]
Pre:  46.79 [ 33.31  83.30]
GPU:   2.83 [  1.30   4.03]
Mem: 454.03MB    [169.00MB]
```

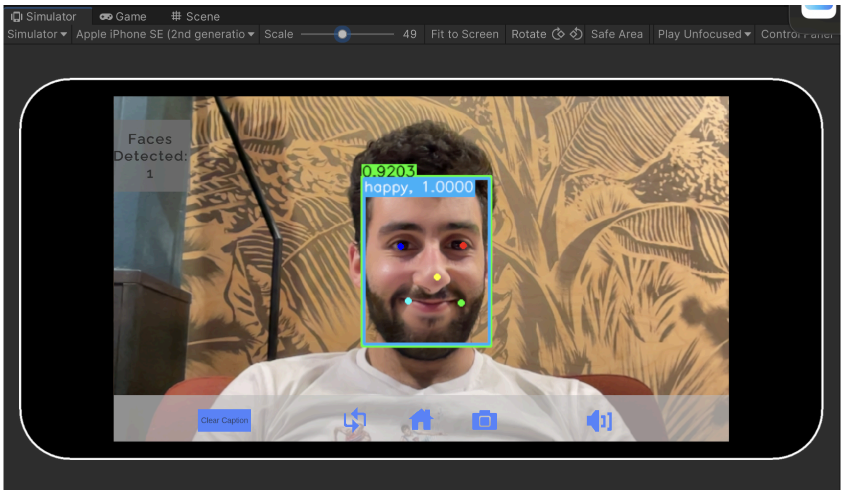
18	User experience and interface consistency across different devices Success criteria points #2, #11 (User interface design) (Ease of use)	Verifying the user experience and interface consistency across different devices, ensuring that the application provides a seamless and uniform experience regardless of screen size or resolution, catering to a wide range of users.	🟩	Build and test on a variety of different iPhones with different processing power and resolutions, evaluating the UI and scaling for each device. (e.g iPhone 7, iPhone 11, iPhone 5)	Valid	Application UI should scale appropriately depending on size and resolution of device.	Different devices may have parts of the UI cut off and different resolutions output UI incorrectly	🟨

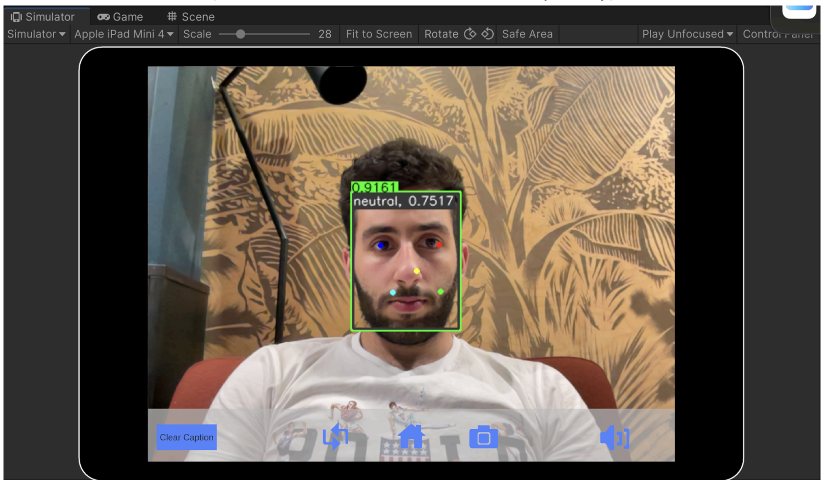
Evidence:

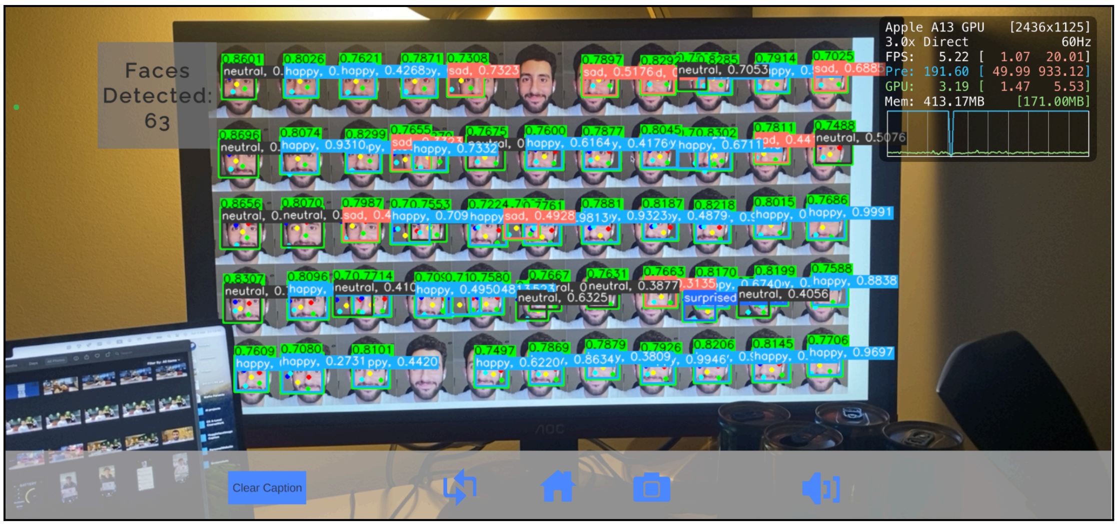
iPhone 12 simulator (UI normal)



iPhone SE simulator (faces detected UI box has no left padding from screen)

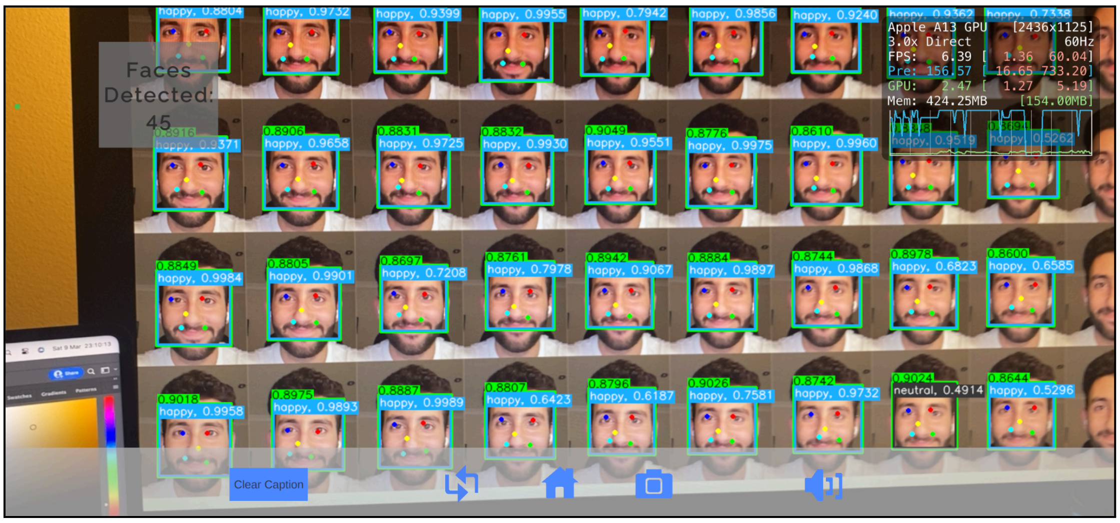


	iPad mini 4 simulator (faces detected UI box cut off completely)  Simulator Game Scene Simulator Apple iPad Mini 4 Scale 28 Fit to Screen Rotate Safe Area Play Unfocused 0.9161 neutral, 0.7517 Clear Caption							
19	Functionality and accuracy of the facial recognition feature in crowded scenes Success criteria points #3, #10 (Identification of people in camera's view/facial recognition) (Accuracy)	Testing the functionality and accuracy of the facial recognition feature in crowded scenes to evaluate its ability to distinguish and analyse multiple faces simultaneously, crucial for real-world applicability and user trust.	🟨	Prepare an image of a large crowd with visible faces and asses the functionality when the AI is given large amounts of data	Boundary	AI should be able to recognise the majority of the faces in the image, however may reach its limit and only detect up to a specific amount.	App freezes and slows down, averaging 1 frame every 5 seconds	🟥
	Evidence: *I created the following collage of 55 photos of myself in order to test the system against a large crowd of people. I loaded this image onto a large monitor and pointed my phone at it and the results are seen afterwards:* *After this image was taken, the app froze and there was no response, which prompted me to restart it. Although there were only 55 faces on the collage, the algorithm detected 63 and placed multiple bounding boxes for 1 face for a number of faces. Furthermore, the primary emotion detected was a mixture of sad, neutral and happy, even though the images are all identical*							



```
Apple A13 GPU    [2436x1125]
3.0x Direct           60Hz
FPS:    5.22 [  1.07  20.01]
Pre:  191.60 [ 49.99 933.12]
GPU:    3.19 [  1.47   5.53]
Mem: 413.17MB    [171.00MB]
```

I tried this again, but I got closer to the monitor so that the features could be extracted easier and found that all of the faces were detected as 'happy'



```
Apple A13 GPU    [2436x1125]
3.0x Direct           60Hz
FPS:    6.39 [  1.36  60.04]
Pre:  156.57 [ 16.65 733.20]
GPU:    2.47 [  1.27   5.19]
Mem: 424.25MB    [154.00MB]
```

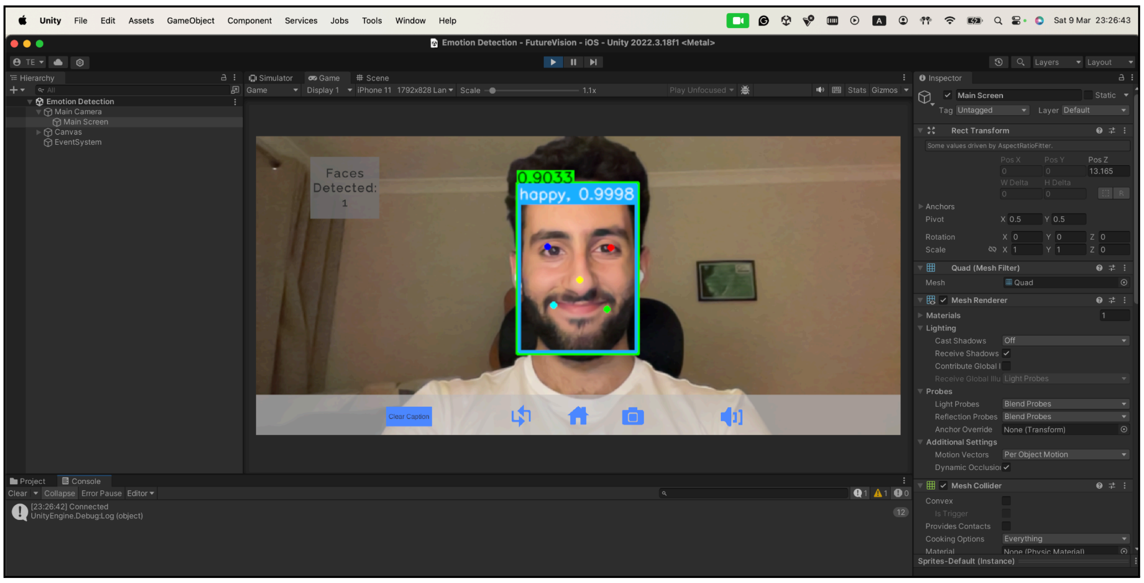
Although the graphics debug box states that the FPS is 5.22 and 6.39 for the above images, in reality, the image on screen updated once every 4-5 seconds

20	Evaluation of app's adaptability to different screen sizes and resolutions Success criteria points #2, #11 (User interface design) (Ease of use)	Assessing the app's adaptability to various screen sizes and resolutions ensures a consistent and optimal user experience across different devices.	🟩	Build and test on a variety of different iPhones with different sizes and resolutions. (e.g iPhone 7, iPhone 11, iPhone 5)	Valid	Application should scale appropriately depending on size and resolution of device.	Different devices may have parts of the camera output and UI cut off and different resolutions output UI and captions incorrectly	🟨
	Evidence: See test 18							
21	Testing the app's performance and	Verifying the app's functionality without	🟨	Turn off device WiFi and	Erroneous and Valid	Image captioning	Image captioning	🟩

	functionality without internet connectivity Success criteria points #1, #11 (Camera functionality when user opens app) (Ease of use)	internet connectivity ensures critical features remain accessible offline, enhancing usability and reliability.	🟨	evaluate useability and features. In particular, ensure the image captioning is turned off		button disappears and remaining features work perfectly.	button disappears and remaining features work perfectly.	🟩

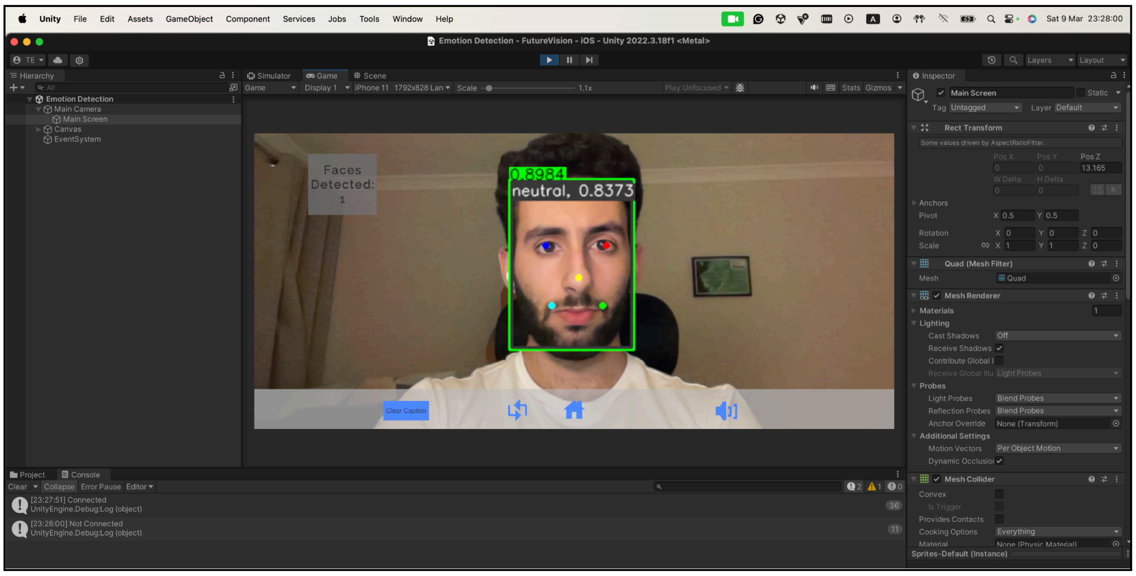
Evidence:

WiFi on (camera button is active, 'Connected' is printed in the console)



```
[23:26:42] Connected
UnityEngine.Debug:Log (object)
```

WiFi off (camera button disappears, 'not connected' printed in console)

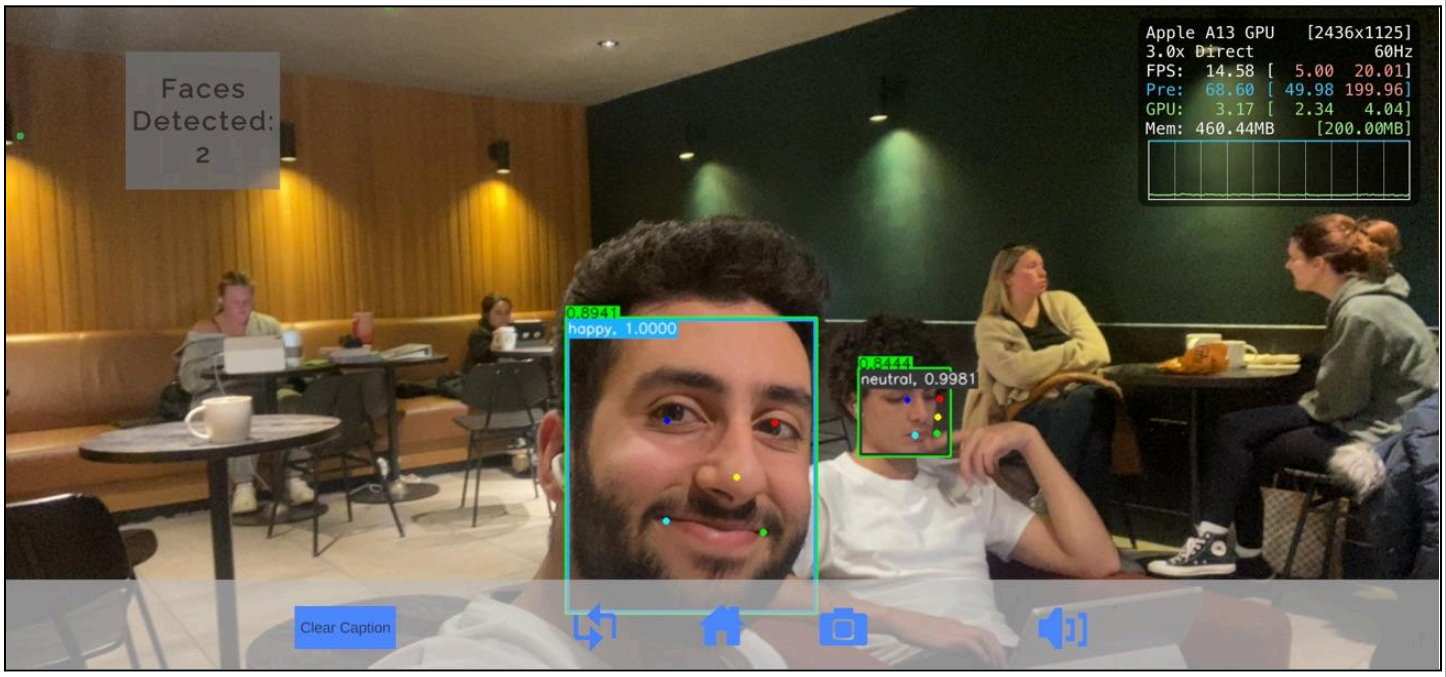


```
[23:27:51] Connected
UnityEngine.Debug:Log (object)
[23:28:00] Not Connected
UnityEngine.Debug:Log (object)
```

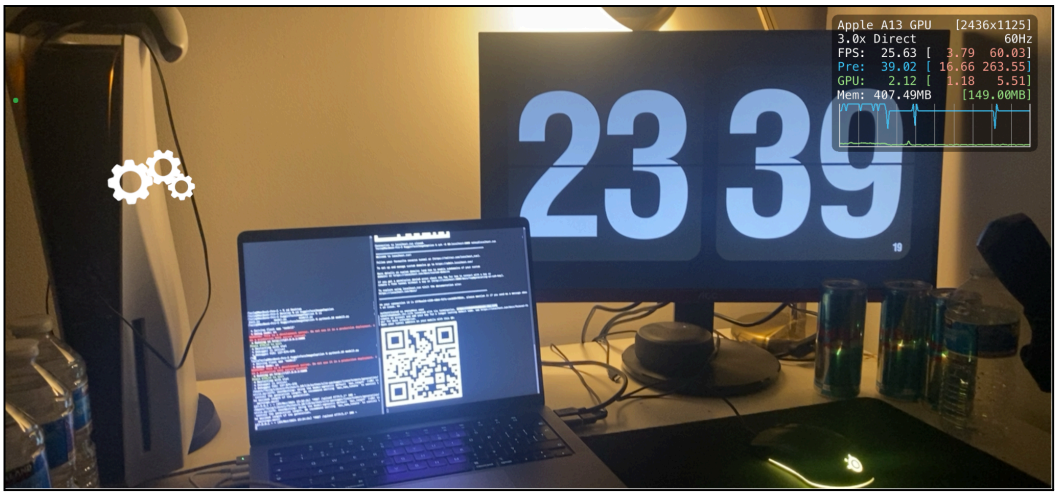
22	Testing voice pitch adjustment in text-to-speech for user personalization Success criteria points #7, #12 (Text-to-speech output for processed attributes) (Variable text-to-speech voice pitch/volume/speed)	Assessing voice pitch adjustment in text-to-speech validates personalization features.	🟥	In the settings page, adjust the TTS slider, type in sample text and press 'test' to hear the audio.	Valid	Users can tailor audio outputs to their preferences and test this.	Users can tailor audio outputs to their preferences and test this.	🟩
	Evidence: screencast							
23	Responsiveness and effectiveness of the user interface during peak processing tasks Success criteria points #2, #9, #11 (User interface design) (Responsiveness - 20fps consistent feed) (Ease of use)	Evaluating the UI's responsiveness and effectiveness during peak processing tasks ensures the application remains user-friendly and efficient under heavy load conditions.	🟩	Within the object detection, or emotion detection script, include many faces/objects which creates many bounding boxes and evaluate the responsiveness.	Boundary	During peak processing tasks, the app should slow down and reduce in responsivenes s due to a bottleneck in system resources	During peak processing tasks,app severely slows down, increases device temperature and occasionally crashes due to high memory useage	🟥
	Evidence: See test 19							
24	Test the text-to-speech feature with the maximum and minimum allowed pitch, volume, and speed settings. Success criteria points: #7, #12 (Text-to-speech output for processed attributes) (Variable text-to-speech voice pitch/volume/speed)	Evaluate the limits of text-to-speech customization settings to ensure that extreme values do not cause unintelligibility or application errors.	🟥	Within the settings page, turn the slider to the maximum and minimum value, then test the audio.	Boundary	Audio should be clear and easy to understand, at an acceptable volume.	Audio clear and easy to understand, at an acceptable volume.	🟩
	Evidence: screencast							

Extra Tests (Post Development Testing)

These are some extra tests that I have included after development, that were not included in the original testing plan

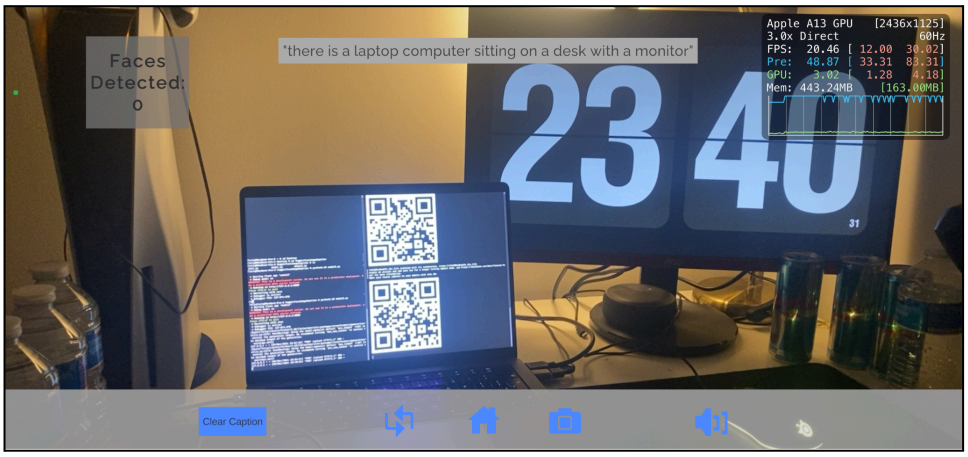
Test No.	Aspect being tested/Link to success criteria	Justification	Importance?	How to perform test	Type of test (valid, invalid, boundary)	Expected result	Actual result	Success?
25	Flip Camera button works as expected Succcess criteria points #11 (Ease of use)	Being able to use the front camera in this project is important as it allows users to caption and take pictures with friends, and perhaps look at themselves, giving this app a sense of playfulness and a greater audience.	🟨	Press 'flip camera' button	Valid	Camera switches from main back camera to front camera	Camera switches from main back camera to front camera	🟩
	Evidence: *Front camera working as expected in high resolution.*  Faces Detected: 2 0.8941 happy, 1.0000 0.8444 neutral, 0.9981 Clear Caption Apple A13 GPU [2436x1125] 3.0x Direct 60Hz FPS: 14.58 [5.00 20.01] Pre: 68.60 [49.98 199.96] GPU: 3.17 [2.34 4.04] Mem: 460.44MB [200.00MB]							
26	'Clear Caption' clears the screen of any caption Succcess criteria points #11 (Ease of use)	Being able to clear the caption off the screen allows greater accessibility to the user as they are able to see the full screen without any distractions.	🟥	Press 'clear captions' button	Valid	Caption disappears and the top of the screen becomes clear	Caption disappears and the top of the screen becomes clear	🟩

	Evidence: *screencast*							
27	Tutorial/Instructions on how to use the app, detailing the functions of each button etc. Success criteria points #11 (Ease of use)	Having a tutorial is quite useful as many people may not understand what the button symbols mean and could be afraid of testing them out due to the lack of communication. A tutorial explaining what the app does and the corresponding buttons will help iron out any confusion	🟩	Intial startup.	Valid	Text-based/video-based tutorial pops up on screen, prompting the user to read and understand what each button/functionality does	Due to time constraints, I was unable to implement this feature into the project	🟥
	Evidence: *Due to time constraints, I was unable to implement this feature into the project*							
28	Auto-Rotate function works, allowing the user to use the app in landscape or portrait whenever they please Succcess criteria points #11 (Ease of use)	Having the choice of how to use the app provides greater flexibility to the user and thus, improves their overall experience. This is why this feature is quite important	🟨	Rotate phone	Valid	Camera and outputs rotate with the iPhone, providing a seamless change.	Due to time constraints and various unavoidable bugs when trying to fix this, I was unable to implement this feature into the project	🟥
	Evidence: *Due to time constraints and various bugs when trying to fix this, I was unable to implement this feature into the project and as a result, have locked the app to stay in landscape mode where it works perfectly fine.*							
29	Caption Loading icon Succcess criteria points #2 (User interface design)	When captioning an image, it can occasionally take a long time. As a result, having an animated icon to indicate that some sort of processing is being taken place saves the user from confusion.	🟥	Press 'caption' button	Valid	When sending a screenshot to be captioned, a spinning cog appears, indicating that a task is being processed.	When sending a screenshot to be captioned, a spinning cog appears, indicating that a task is being processed.	🟩
	Evidence: *White spinning cog appears when captioning an image and the other UI elements disappear in order to provide 'abstraction' to the captioning AI*							



```
Apple A13 GPU    [2436x1125]
3.0x Direct             60Hz
FPS:   25.63 [  3.79  60.03]
Pre:   39.02 [ 16.66 263.55]
GPU:    2.12 [  1.18   5.51]
Mem: 407.49MB     [149.00MB]
```

After a caption is returned, all the UI elements reappear and the cog disappears



Faces Detected: 0

"there is a laptop computer sitting on a desk with a monitor"

```
Apple A13 GPU    [2436x1125]
3.0x Direct             60Hz
FPS:   20.46 [ 12.00  30.02]
Pre:   48.87 [ 33.31  83.31]
GPU:    3.02 [  1.28   4.18]
Mem: 443.24MB     [163.00MB]
```

Robustness

In evaluating the robustness of my AI application, which encompasses face detection, emotion recognition, and image captioning functionalities, a comprehensive series of tests were conducted to ensure its operational integrity across a wide array of scenarios. Initially, the app was subjected to a diverse testing environment involving individuals exhibiting a range of emotions—happiness, sadness, neutrality, anger, surprise, and disgust—in varying group sizes, from solitary figures to gatherings of up to 15 people. Although the app demonstrated commendable performance in recognizing and processing emotions under these conditions, a notable decline in response time was observed as the number of faces increased, with the system showing signs of strain beyond ten to fifteen faces, occasionally leading to crashes. This highlighted a potential limitation in handling large groups, though it remains efficient for the majority of practical applications.

To ensure the app's versatility, it was also tested under varying lighting conditions, from dim to brightly lit environments. The system adeptly handled these variations, maintaining high accuracy across all tested emotional states, underscoring its robustness in face and emotion detection tasks.

The image captioning feature, powered by a pre-trained model on the expansive COCO dataset, was anticipated to offer significant accuracy due to its extensive training background. While specific tests for this feature were not detailed, the reliance on a robust dataset suggests confidence in its capability to generate relevant descriptions across a wide range of scenes and objects.

During operation, the app utilizes an iPhone camera to process images in real time, thus bypassing issues related to invalid data inputs such as non-image files or corrupted images. However, under high-load conditions, the app exhibited performance challenges, attributed to the intensive processing demands placed on the device. This aspect reveals a significant area for improvement, possibly suggesting a shift towards server-based processing to alleviate the computational load on the device and enhance the overall user experience.

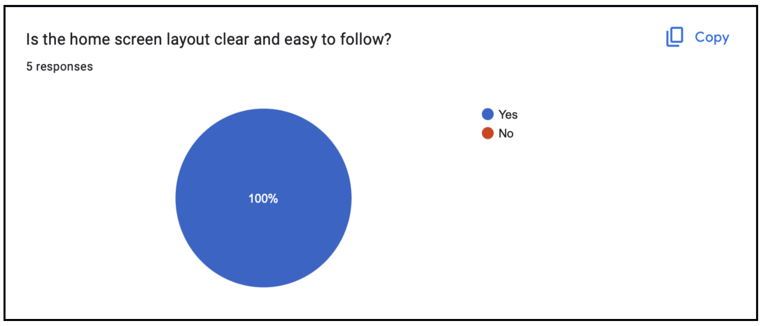
In summary, the robustness evaluation of my AI app reveals a system capable of performing effectively under a variety of conditions with limitations primarily in handling large groups and under intense processing demands. Future enhancements would focus on optimizing performance under high-load scenarios, possibly through server-side processing, and refining face detection capabilities to better manage scenarios involving numerous individuals, thereby ensuring a seamless and efficient user experience across all functionalities.

Useability Testing

Menu and UI

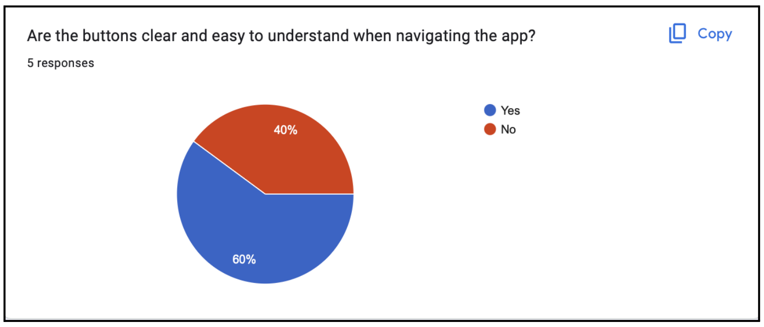
The following questions relate to the menus and UI within my app that the user will interact with to access different parts of the program such as creating a ‘object detection’, or ‘emotion detection’, accessing the settings page etc.

Is the homes screen layout clear and easy to follow?



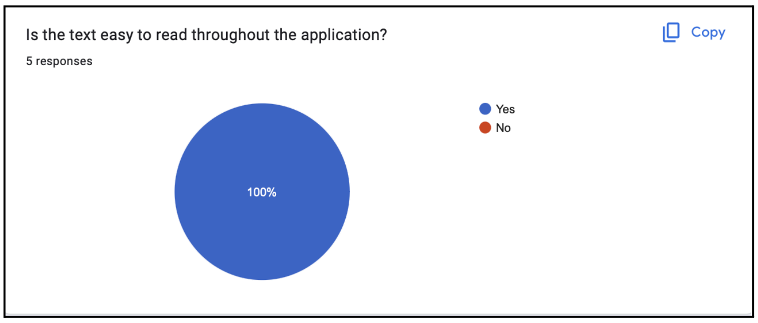
Since the user sees the main screen of the app initially, it must be easy to navigate so that the individual does not become frustrated or puzzled by it. With 100% of respondents indicating "yes," the main menu is in fact straightforward, according to the comments I received on my Google Form. Therefore, if I had more time, I wouldn't need to alter the main menu's layout.

Are the buttons clear and easy to understand when navigating the app?



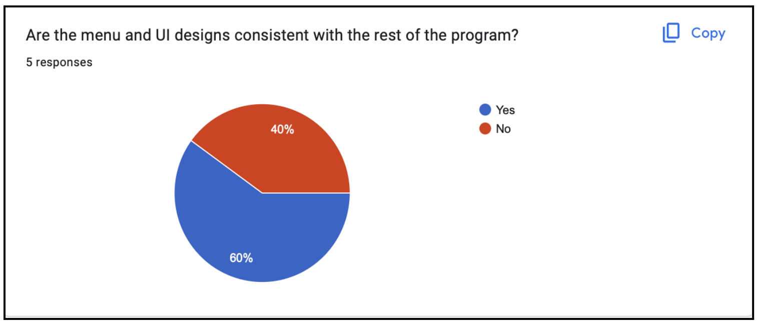
The clarity and ease of use of an application's interactive elements are vital for a seamless user experience. In the feedback gathered, 60% of respondents affirmed that the buttons within the app are clear and easy to understand, suggesting a majority positive experience. However, a notable 40% indicated the opposite, which points to a potential area for improvement. While the results are predominantly positive, the dissenting opinions highlight the necessity for a closer evaluation of the app's button design and labels. To ensure a more intuitive navigation for all users, it may be worthwhile to investigate the specific concerns of those who responded negatively and consider adjustments that could enhance the app's overall usability. It is essential to strive for a design that accommodates the needs and preferences of a wider user base, particularly in elements as fundamental as navigation buttons.

Is the text easy to read throughout the application?



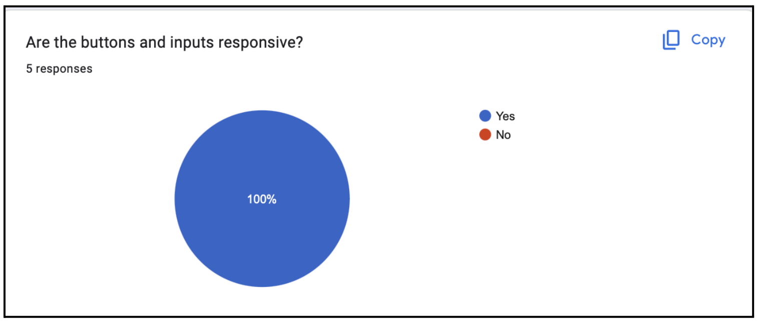
Text legibility is a fundamental component of application design, impacting user engagement and overall satisfaction. The feedback received unanimously indicates that the text within the app is easy to read, with 100% of the respondents giving a positive "yes" response. This suggests that the current font size, style, and contrast are well-suited to the users' needs, facilitating a smooth and accessible reading experience across the app. This strong consensus among the limited pool of respondents is encouraging; however, it's also important to remain open to further feedback as the user base grows. Continuous monitoring and testing with a more diverse group of users can ensure that the text remains readable for new users with varying visual preferences and needs.

Are the menu and UI designs consistent with the rest of the program?



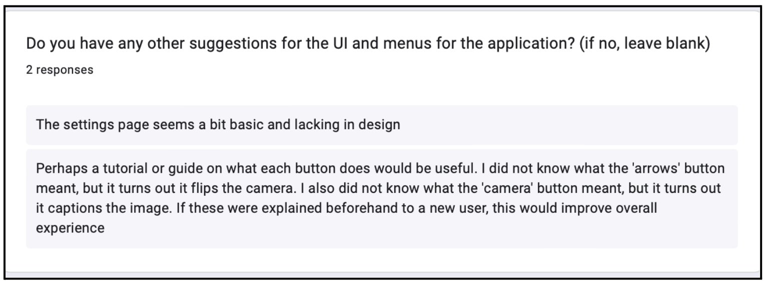
Consistency in UI design is key to providing a seamless user experience. According to the feedback, a majority of 60% of users reported that the menu and UI designs were consistent across the app. However, 40% of respondents perceived a lack of consistency, which could be attributed to the use of a pre-built settings page that does not entirely match the custom-designed UI of the rest of the application. This deviation in design quality is acknowledged and stems from time constraints during the development process. Moving forward, it is recognized that investing time in redesigning the settings page to match the aesthetic and functional quality of the custom UI would likely enhance user satisfaction and create a fully cohesive experience throughout the app. Addressing this inconsistency will be a priority to ensure that all elements of the app meet the high standards set by the majority of the custom user interface.

Are the buttons and inputs responsive?



Responsiveness is a crucial factor in the usability of any application, directly affecting the efficiency and satisfaction of the user experience. The feedback obtained from the stakeholders reveals a unanimous agreement with 100% of the respondents indicating that the buttons and inputs are responsive. This outcome suggests that the application performs well in terms of interaction speed and feedback, which is essential for users who expect immediate and reliable responses to their actions within the app. The positive feedback on responsiveness indicates that, technically, the app's interactive elements are optimized for user engagement. It's important to maintain these performance standards as the app scales and to continue to gather user feedback to ensure responsiveness remains consistent across various devices and usage scenarios.

Do you have any other suggestions for the UI and menus for the application?



Do you have any other suggestions for the UI and menus for the application? (if no, leave blank)

2 responses

The settings page seems a bit basic and lacking in design

Perhaps a tutorial or guide on what each button does would be useful. I did not know what the 'arrows' button meant, but it turns out it flips the camera. I also did not know what the 'camera' button meant, but it turns out it captions the image. If these were explained beforehand to a new user, this would improve overall experience

User feedback is invaluable for refining an application's UI and enhancing its functionality. From the two responses provided, we can discern a common theme around the need for clarity and depth in design.

The first response critiques the settings page for its basic appearance and lack of design complexity, which echoes earlier feedback and suggests that a redesign could be beneficial to align with the more polished aspects of the application.

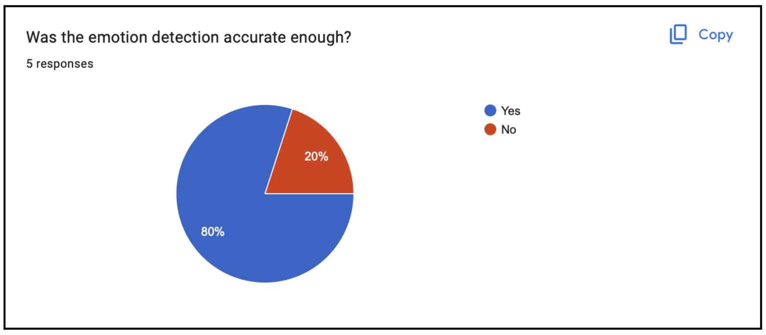
The second response requests additional guidance for users, indicating a gap in the app's intuitiveness. Specifically, it points to the need for a tutorial or guide that explains the function of each button, such as the 'arrows' for flipping the camera and the 'camera' for captioning the image. This indicates that while the design may be visually pleasing, its functionality is not immediately obvious to all users.

Addressing these concerns by enhancing the settings page design and implementing a user guide or tutorial would likely improve user experience. It would provide a more cohesive design language throughout the app and empower users with the knowledge to fully utilize the app’s features from the outset.

App Functionality and Features

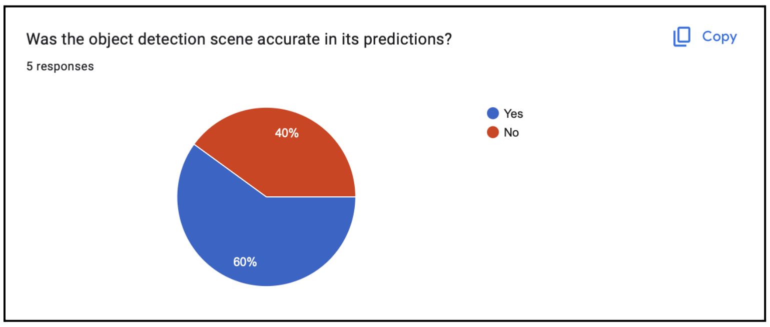
The following questions are related to the functionality and the features of the application, ranging from the settings page functionality, to the image captioning

Was the emotion detection accurate?



In the realm of app features, emotion detection accuracy is crucial for user trust and engagement. The feedback provided by users indicates that 80% found the emotion detection feature to be accurate enough, reflecting a high level of satisfaction with this functionality. However, the remaining 20% expressed that the accuracy was not sufficient, suggesting there is room for improvement in the algorithm or its implementation. While the positive response from the majority is promising, the critical feedback from a minority is important to consider for future updates. It might be beneficial to investigate the circumstances or types of interactions where the emotion detection falls short. Enhancing this feature based on specific user experiences could lead to a more robust and reliable emotion detection system, further increasing user satisfaction and broadening the app's appeal.

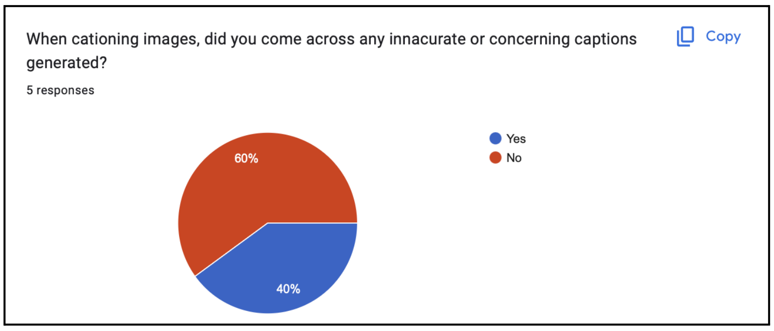
Was the object detection scene accurate in its predictions?



Feature accuracy is crucial for applications relying on artificial intelligence, like object and emotion detection. For the object detection feature, the feedback indicates that 60% of respondents find the predictions accurate. This suggests that while the feature generally performs well, there is room for improvement, as a notable 40% of users did not find the object detection accurate enough.

Addressing the concerns of the minority could involve refining the algorithms or providing better training data to improve accuracy. It’s also worthwhile to consider user expectations and experiences on a deeper level to understand the specific inaccuracies they’re encountering. This might not only lead to technical enhancements but also adjustments in how the app sets expectations for its performance. It's imperative to aim for as close to 100% accuracy as possible to enhance user dependency on the application.

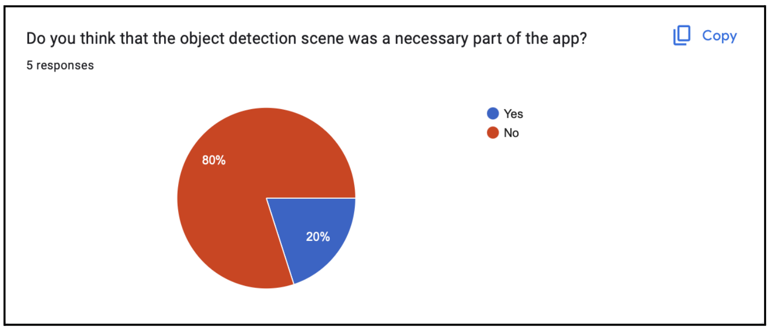
When captioning images, did you come across any inaccurate or concerning captions generated?



The accuracy of caption generation in image-related applications is significant for user engagement. The feedback received shows that 60% of users did not encounter any inaccuracies or concerns with the captions generated by the app, which is a positive indicator of the system's reliability. However, 40% did experience issues with the generated captions, suggesting that there are certain discrepancies between the app’s output and user expectations or reality.

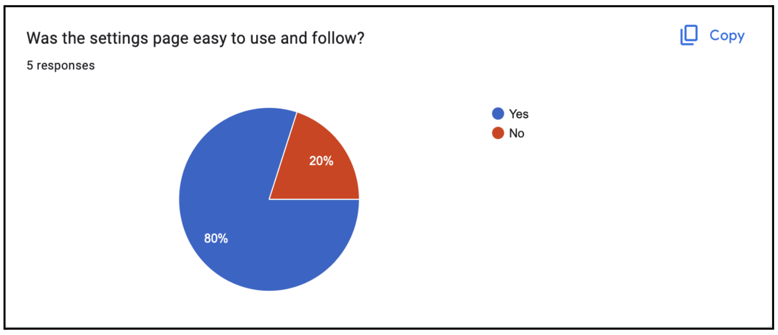
This feedback highlights an area that could benefit from further development. Enhancing the captioning algorithm, possibly by expanding the training dataset or incorporating more sophisticated natural language processing techniques, might improve the accuracy. Additionally, implementing a feature for users to provide immediate feedback on caption accuracy could help in quickly identifying and rectifying specific issues. This approach would not only improve the app's performance but also engage users in the refinement process, potentially increasing satisfaction with subsequent updates.

Do you think that the object detection scene was a necessary part of the app?



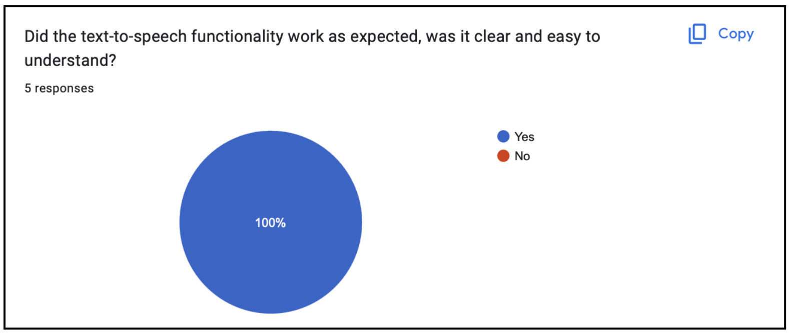
Reflecting on the functionality of the object detection scene within the app, the user feedback indicates a significant viewpoint; 80% of respondents do not consider it a necessary element of the application. This suggests that for the vast majority, the object detection feature may not contribute to their primary use case or enhance their user experience in a meaningful way. Conversely, the remaining 20% do see its value, hinting at specific use cases or preferences where this feature plays a vital role. Moving forward, it would be prudent to delve deeper into understanding the needs and expectations of the user base. This could involve re-evaluating the prominence given to object detection in the user interface or possibly redefining its role within the app’s ecosystem, ensuring that the app's features are closely aligned with the needs of its users.

Was the settings page easy to use and follow?



Navigability and clarity on the settings page are essential for a positive user experience. The feedback indicates that the majority, 80%, found the settings page easy to use and follow. This is a strong endorsement of the page's current layout and functionality. However, there is still a notable minority, 20%, that did not find the settings page user-friendly. This suggests that while the settings page functions well for most, there could be specific elements that are not as intuitive or clear as they could be for all users. It might be worthwhile to investigate these particular cases to understand the challenges faced by the minority. Improving the settings page by addressing the issues highlighted by the 20% could lead to enhanced usability and a more universally positive response from the user base.

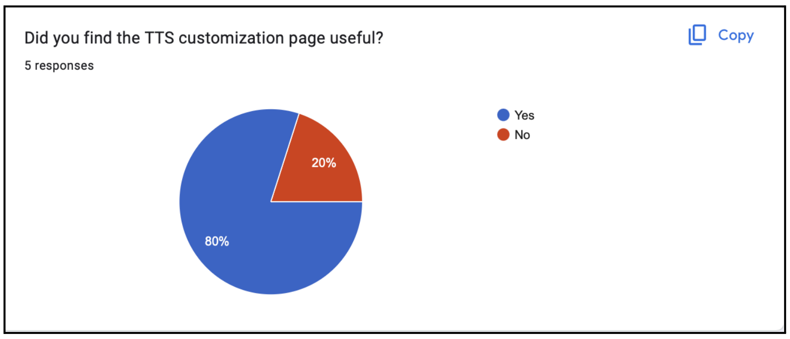
Did the text-to-speech functionality work as expected, was it clear and easy to understand?


Did the text-to-speech functionality work as expected, was it clear and easy to understand?

5 responses

The text-to-speech feature of the app appears to meet users' expectations splendidly, with all respondents (100%) confirming that it worked as expected and was clear and easy to understand. This unanimous feedback underscores the effectiveness and user-friendliness of the text-to-speech functionality, suggesting it's a well-implemented aspect of the application. A consistent and reliable performance in such features is crucial as it can greatly enhance the accessibility of the app, allowing a broader audience to benefit from its use. It is encouraging to see such positive feedback, and maintaining this level of functionality will be important as the app continues to evolve.

Did you find the TTS customization page useful?

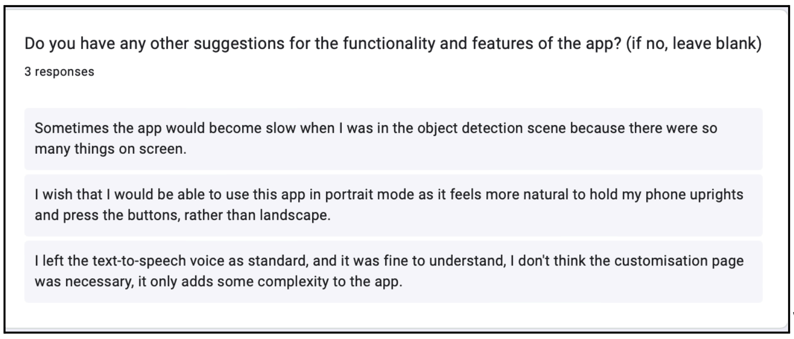

Did you find the TTS customization page useful?

5 responses

The Text-to-Speech (TTS) customization page is considered useful by the majority of respondents, with 80% affirming its usefulness. This reflects a strong approval for the customization features provided, suggesting that users appreciate the ability to personalize their TTS experience. Tailoring the speech output to individual preferences can greatly enhance user satisfaction, as it allows for adjustments in voice, speed, and other parameters to match user needs.

The remaining 20% who did not find the customization page useful might indicate specific areas for improvement or perhaps a need for more guidance on how to effectively use the customization options. It could also point to different expectations about what 'customization' should entail. Understanding the needs and preferences of this minority group could provide valuable insights into how the customization page can be improved or made more intuitive. Ensuring that the TTS customization options are both accessible and comprehensible to all users will contribute to a more inclusive app experience.

Do you have any other suggestions for the functionality and features of the app?



Do you have any other suggestions for the functionality and features of the app? (if no, leave blank)

3 responses

Sometimes the app would become slow when I was in the object detection scene because there were so many things on screen.

I wish that I would be able to use this app in portrait mode as it feels more natural to hold my phone uprights and press the buttons, rather than landscape.

I left the text-to-speech voice as standard, and it was fine to understand, I don't think the customisation page was necessary, it only adds some complexity to the app.

\

User feedback is crucial for continuous improvement, and these three responses offer actionable suggestions for the app’s functionality and features:

Performance issues are noted during heavy usage within the object detection scene. Users have experienced slowdowns when the screen is populated with multiple items. This feedback suggests that optimizing the performance, possibly by streamlining the object detection processing or improving the app's resource management, could enhance the user experience.

A preference for portrait mode indicates a desire for more flexible user interaction. This user finds holding their phone in an upright position and pressing buttons more natural, which suggests an opportunity to improve the app's orientation options, potentially making it more accessible and comfortable for a broader user base.

While the text-to-speech feature is deemed understandable, at least one user feels that the customization of this feature adds unnecessary complexity. This insight points towards a need for balancing advanced options with maintaining simplicity. It could be beneficial to consider a more simplified version of the TTS customization for users who prefer a more straightforward experience, perhaps offering basic and advanced settings.

Taking these suggestions into account could lead to a more refined, user-friendly app, catering to the diverse needs and preferences of its users.

Review on useability

Our comprehensive analysis of user feedback presents a nuanced view of the app's usability. The stakeholders' responses offer valuable insights, reflecting a commendably high level of satisfaction with several core aspects, alongside constructive criticism in specific areas. Notably, the unanimous approval of the text readability and the responsiveness of buttons and inputs underscores the app’s proficiency in fundamental usability metrics. Such positive responses resonate with the aim to deliver a user-friendly experience, which is further corroborated by the majority finding the settings page navigable and the text-to-speech functionality meeting expectations.

However, this positive sentiment is tempered by concerns over the consistency and intuitive nature of the UI, particularly regarding the settings page, which was noted to lack the finesse of the app's custom-designed elements. Additionally, although the object detection feature was technologically sound, a significant 80% did not perceive it as necessary, highlighting a possible misalignment with user needs. The TTS customization page, while deemed useful by the majority, also faced scrutiny from users who favored simplicity over complex personalization options.

A recurring theme of desiring simplicity and ease of use was also evident, with suggestions for portrait mode capabilities suggesting that ergonomic considerations could further refine the user experience. Users expressed the need for optimizations, particularly within the object detection scene, to maintain app performance even when dealing with high on-screen object density.

In synthesizing the responses, it is clear that while the app excels in accessibility and clarity of content, it should strive for a balance between advanced features and maintaining simplicity. To enhance usability, the next steps would include streamlining the settings interface, providing user guidance for unfamiliar features, and ensuring that every aspect of the app, especially additional features like object detection and TTS customization, aligns seamlessly with user expectations. The app’s strong foundation sets the stage for these targeted improvements to bolster usability further, ensuring the app remains both functional and enjoyable for all users.

Evaluating Each Success Criteria

Importance:
RED - critical for application functionality
ORANGE - Moderately important for app functionality
GREEN - Additional, yet unnecessary features for overall app aim

Completion:
RED - Incomplete
ORANGE - Partially complete
GREEN - Completed

ID	Importance	Completion	Criteria	Evidence	Final Evaluation
1	🟥	🟩	Camera functionality when user opens app	Tests 5, 6, 7, 9, 21 (Post development Testing)	This was perhaps the most critical part of the project and undoubtedly one of the most complex parts also. This criterion included me learning how to communicate with the device camera through Unity and converting it to a mappable texture, which is then copied onto a gameObject renderer. This made up the majority of the time spent during iteration 1. This section was fully completed, with the ability to switch cameras seamlessly for both the object desdtection and emotion detection.
2	🟧	🟧	User interface design	Tests 1, 2, 3, 4, 11, 14, 18, 20, 23, 29 (Post development Testing)	Although most of the UI elements were completed. Parts of the app were not consistent with the rest of the theme, in particular, the settings page. Furthermore, I feel as though The UI on the homescreen was cleaner and more aesthetically pleasing than the UI in the main 'emotion detection' and 'object detection' scenes. Perhaps if I had time I would work on this further and tailor this to look more professional and interlinked with the rest of the app.
3	🟥	🟩	Identification of people in camera's	Tests 7, 10, 19 (Post development	Being able to identify people and faces in Unity proved to be a challenge. During my research

	🟥	🟩	view/facial recognition	Testing)	phase, I was under the impression that I was simply able to import an AI model into Unity's built-in barracuda AI processing plugin, however as it turned out, this was close to impossible in the given time constraints, which resulted in me researching and selecting a separate plugin that has many pre-trained and loaded models with their corresponding scripts. The most challenging part was perhaps being able to communicate these pre-built scripts with the camera texture that I had rendered from earlier. In conclusion, this project-critical section of the application was fully complete and exceed my expectations in it's accuracy and responsiveness when processing locally on a mobile phone with relatively low processing power.
4	🟩	🟩	Object detection	Tests 6, 8, 13 (Post development Testing)	Although this section was entirely optional, I came across an object detection pre-trained model alongside the facial recognition model and thought to include an additional feature, which may not be useful to many, but some users may find amusement and curiosity in the power of artificial intelligence being able to identify every day objects. Since I already programmed the logic for the pre-built OpenAI scripts to communicate with my camera output scripts, this criterion was much more simple than some of the other ones, which allowed me to fully complete this section of the app, despite its lack of importance.
5	🟥	🟩	Home screen	Test 1, 2, 3, 4 (Post development Testing)	My app is entirely based on accessibility and open useability. Every user should immediately be greeted with a pleasant home screen, providing them with valuable direction and insight into the general app. Although this wasn't difficult to program. The UI for the home screen set the layout

	🟥	🟩			for the rest of the app, which made the design for this, including the slogan, quite time consuming yet nonetheless, I was able to fully complete it with complete positive feedback.
6	🟥	🟩	Mood recognition based on facial expressions	Test 7, 10, 15 (Post development Testing)	Mood recognition is a critical aspect of my project, providing the backbone for many of the other features. This was fully completed as it had to be perfect. The mood recognition received overwhelmingly positive feedback and even throughout testing, I was very impressed with its accuracy, even in very different and complex environments.
7	🟧	🟩	Text-to-speech output for processed attributes	screencast	I wanted to include Text-To-Speech as an extra feature that demonstrated accessibility, which is what this project aims to bring for everyone. Despite what I had originally thought, the text-to-speech was far more difficult than I had assumed as I had never worked with Apple's built in TTS kit. This feature of the project demanded large amounts of research into how I could implement this into my work. Something that slowed me down significantly, is that in order to test its functionality, I was forced to build it onto my phone each and every time since the libraries are not accessible on my laptop. This was very time consuming, especially for someone who is learning to use this while developing, although I was pleased that I was able to complete this whole optional section of my success criteria
8	🟥	🟩	Image capture and captioning	Test 17 (Post development	Image captioning provides this app with another dimension through computer vision. With image

	🟥	🟩		Testing)	captioning, people are able to understand and simulate their surroundings at the touch of a button, without even needing to see what's around them. I was fixated on building the image captioning model myself, however after weeks of development, I had hit a roadblock and was unable to continue, and while I was close to giving up, I found huggingFace, which provided me with an alternative, although much simpler and less rewarding method of image captioning by using their pre-trained models and a Python library. However, I was determined to include this feature in my project so I settled for it. If I had more time, I would have started from scratch and learned to train, evaluate and export my own image captioning model, which I can then use for server-side processing.
9	🟥	🟧	Responsiveness (20fps consistent feed)	Test 9, 14, 16, 23 (Post development Testing) and screencast	Since my project aims to be accessible and useable by all smoothly and without confusion, I wanted every aspect of this app to run smoothly and highly responsive, with an ideal target framrate of 20 frames per second. However, the pre-trained OpenAI models for objectdetection and facedetection could not scale well when camera resolution increased. As a result, the system framrate dropped to an average of around 10fps all around. While this is still useable and manageable, it doesn't provide the smooth elegant responsiveness that I was aiming for. Upon further research, it was made clear to me that Unity is not designed for AI processing, whether they are pre-trained or not. If next time I want high framerate and responsiveness, I should

	🟥	🟧			attempt to build the iPhone app via another engine or programming language such as Swift.
10	🟥	🟧	Accuracy	Tests 10, 13, 15, 17, 19 (Post development Testing)	Ideally, every aspect of this app should be highly accurate as one of the aims is to potentially help people who many require assistance from visual aid and thus I do not want to be providing inaccurate and confusing information as this may cause confusion and other problems. In order to increase accuracy, I would need to increase the resolution of the cameras and the rate at which the images are sent for processing, however this will result in a large drop in performance and framerate. As a result, some sacrifices must be made. In order to maintain framerate, some accuracy will be lost and vice versa, however in the grand scheme of the overall project, I think this criterion came out to be relatively successful, especially in the emotion detection scene, when focused on 1-3 people.
11	🟥	🟧	Ease of use	Tests 2, 3, 4, 11, 14, 16, 18, 20, 21, 23, 25, 26, 27, 28 (Post development Testing)	In order to be as accessible to everyone as possible, my app needs to be easy to use and understand, without confusion. I don't feel as though I adequately completed this criterion. According to some of the feedback I received stated that some users were confused by the buttons due to the lack of tutorial or text. If I had more time for this project, I would have also included a tutorial on the functionality of the app and what each button's purpose is.
12	🟩	🟩	Variable text-to-speech voice pitch/volume/speed	screencast	Although not necessary for the project, the I wanted to improve the ease of access even further by allowing the users to customise the TTS voice however they please.

	🟩	🟩			With the ability to change the voice, pitch, and speaking rate, for when the image captions are loaded. This section was fully complete as the TTS library that I imported for iPhone already included the TTS customization scene, which I used as the settings page due to the lack of time to work on a better, more polished settings page.

Limitations and Maintenance

Limitations

In assessing the challenges encountered during the development of the application, several limitations have surfaced that provide essential learning points for future iterations. These limitations, categorized under specific subheadings, present opportunities for both technical improvements and a better alignment with user expectations.

Rotation and UI challenges:

- The project faced significant rotation and scaling limitations, particularly notable in the inconsistent behavior of UI and camera orientation. The UI elements, including crucial features like emotion detection indicators, would often rotate appropriately with the device but the camera feed itself would not, resulting in upside-down visuals. Additionally, device disparity, especially with iPads, led to UI elements being cropped out of view, indicating a need for more adaptive UI scaling. These challenges suggest a gap in initial planning for cross-device compatibility and point towards a necessary overhaul of the rotation management system to ensure a consistent and user-friendly interface across all devices.

Object Detection and Power efficiency:

- Object detection emerged as the primary power-consuming process, due to the high demands it places on the device's CPU and memory. The intensive processing of multiple objects significantly drained the battery, leading to inefficiency compared to other apps with similar functionalities. Acknowledging this as a first venture into AI and camera/image processing tasks, the experience has been a substantial learning curve. Future iterations would benefit from exploring methods to optimize object detection, perhaps by simplifying the process or selectively processing objects to minimize the computational load.

Network dependency for image captioning:

- The app's reliance on network speed for image captioning has resulted in variable performance. While a fast connection could yield a caption in as little as 5-7 seconds, a weaker signal could extend this to a full minute. During this period, the UI becomes unresponsive, leaving the user with no option but to wait. The absence of a 'cancel captioning' feature, due to time constraints in development, reduces user experience by eliminating user control during this process. Implementing a cancel option would restore agency to the user, allowing them to opt-out of prolonged waiting times and enhancing the app's responsiveness and usability in varying network conditions.

Caption Accuracy Versus Processing Time:

- In striving for accuracy in image captioning, a variety of pre-trained models were evaluated. The chosen model strikes a balance between accuracy and processing time, providing reasonable results within an acceptable timeframe. More accurate models exist, but their longer processing times were deemed impractical for the app's purpose. The trade-off between precision and speed is a common challenge in AI applications, and continuous testing under varying conditions—like poor lighting or lower resolutions—is essential for further refinement. Future developments might also consider incorporating user feedback mechanisms to improve captioning algorithms based on real-world usage and performance.

Ease of access issues:

- Ease of access was compromised in the app due to the reliance on iconography without accompanying text descriptions, leading to user confusion regarding feature functionalities. The absence of a 'help' section or tutorial due to time constraints exacerbated this issue, as users were left without guidance. To remedy this, future updates could prioritize the development of an intuitive help page or an interactive tutorial that walks users through the app’s features, thus fostering a more accessible and user-friendly environment.

Maintenance

Maintenance and regular updates are essential for the continued success and reliability of any application. For this project, a systematic approach will be adopted to ensure that the app remains functional, efficient, and user-friendly, while adapting to evolving user needs and technological advancements.

Feature updates and optimization:
One of the key areas of focus will be on the feature set and overall performance of the app. This includes routine optimization of the AI algorithms to enhance speed and reduce power consumption. Regular benchmarking against newer models and methods will be essential to ensure that object detection and image captioning remain both accurate and efficient. Addressing user feedback, such as the demand for portrait mode or the inclusion of a 'cancel captioning' feature, will be prioritized to ensure the app evolves in line with user expectations.

UI/UX Enhancements:
Feedback highlighted the need for clearer UI elements and ease of navigation. Future updates will aim to integrate text descriptions alongside icons for better clarity, and the development of a 'help' page or interactive tutorial is slated for the next cycle of updates. Responsiveness across various devices and orientations will be addressed, with a focus on improving rotation handling and UI scaling to cater to a wide range of devices, including tablets.

Technical Maintenance:
Regular technical maintenance will include updates to ensure compatibility with the latest operating systems and device standards. Efforts will also be made to reduce the app's power consumption by refining the codebase for better CPU and memory management. Ensuring scalability of the backend to handle an increasing number of requests will be crucial, especially as the user base grows.

Content and Accessibility Updates:
To maintain the app's relevance, content such as object libraries for detection and caption databases will be continuously updated. This ensures the app can recognize and correctly caption the latest objects and scenes users might encounter. Additionally, accessibility features will be enhanced to support users with different abilities, making sure the app is inclusive and compliant with accessibility standards. This includes updates to the text-to-speech system, and user feedback. I would also like to add haptic feedback at some point in the future, as an optional method of communication, perhaps to indicate when a caption is returned so that a user with visual impairments would know when to request the caption be spoken aloud via the TTS.

Monitoring and Analytics:
Implementing robust monitoring and analytics will inform the maintenance strategy. By understanding how users interact with the app and where they encounter problems, updates can be strategically targeted to areas that will have the most significant impact on the user experience.

In essence, the maintenance strategy for this project is to be as dynamic and user-focused as the app itself. By continually iterating and improving upon the app’s foundation, the goal is to maintain its standing as a helpful and dependable tool for users, adapting not only to the technological landscape but also to the changing needs of its audience.

Final Evaluation

Reflecting upon the journey of this project, I am confident in considering it a commendable success. Evaluating against the success criteria set at the project's inception, it is gratifying to note that nearly all the intended functionalities were effectively implemented within the project timeline. Although some criteria were scaled back due to the project's scope, the achievements in functionality and the resultant product are aspects that instill a strong sense of pride.

The success of the project is further substantiated by the affirmative feedback from stakeholders. Engaging with them consistently throughout the development process, adhering to the rapid application development methodology, proved to be beneficial. Regular interactions yielded critical evaluation points and constructive feedback, which shaped the iterative improvements of the application. The overall positive responses, particularly highlighted by the user experience scores from the final usability testing, reinforce the project's success.

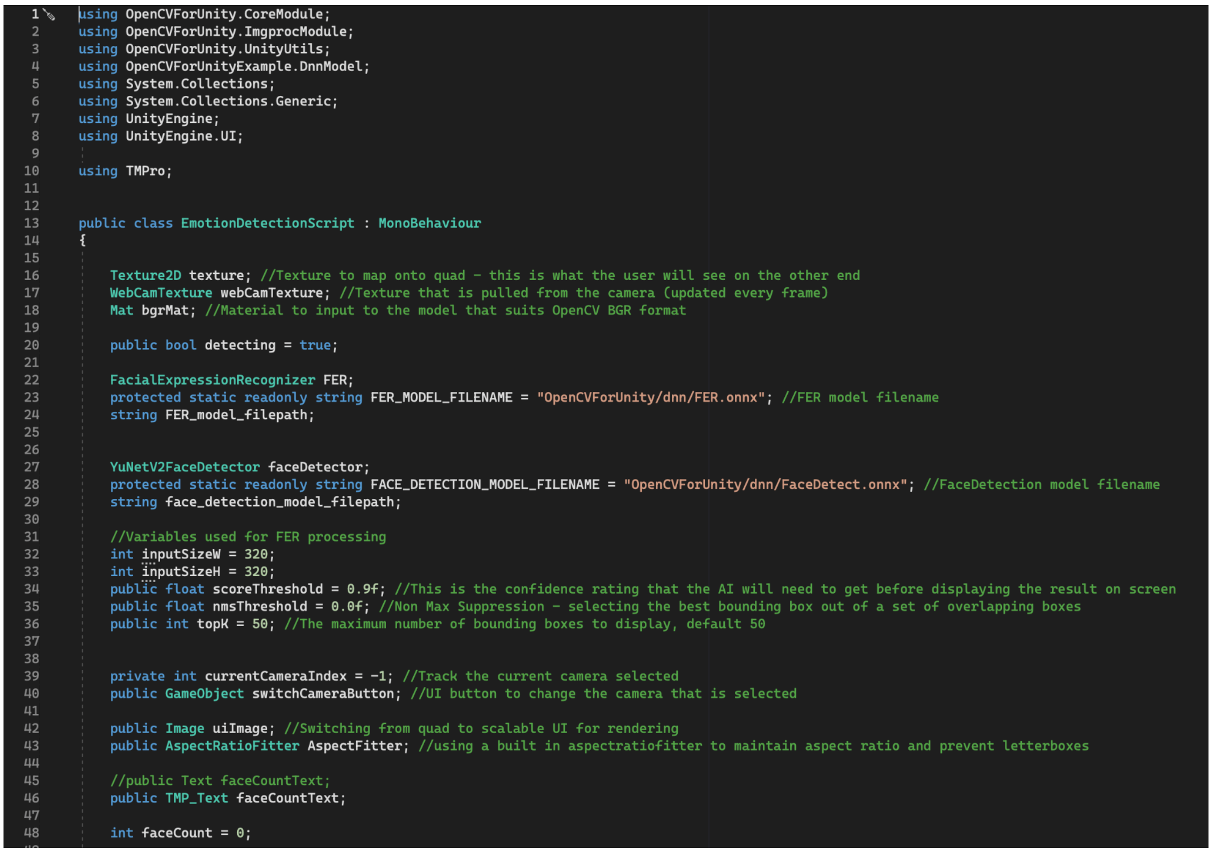
The design robustness and user-centric approach have been pivotal. The application was crafted with the end-user in mind, ensuring an intuitive user interface and a seamless user experience. These aspects were not by chance but by design, and they underscore the careful consideration of usability in the app's development.

In conclusion, Even though this project was successful, I think I misjudged how much time and work it would require, which prevented me from producing the polished app I had hoped for. The project stands as a testament to skill development, user engagement, and technical execution. While there were challenges, such as time constraints and the ambitious nature of the project, the overall outcome has been overwhelmingly positive. The application not only meets the set criteria but does so in a way that affirms my dedication and the project's potential for future growth and refinement.

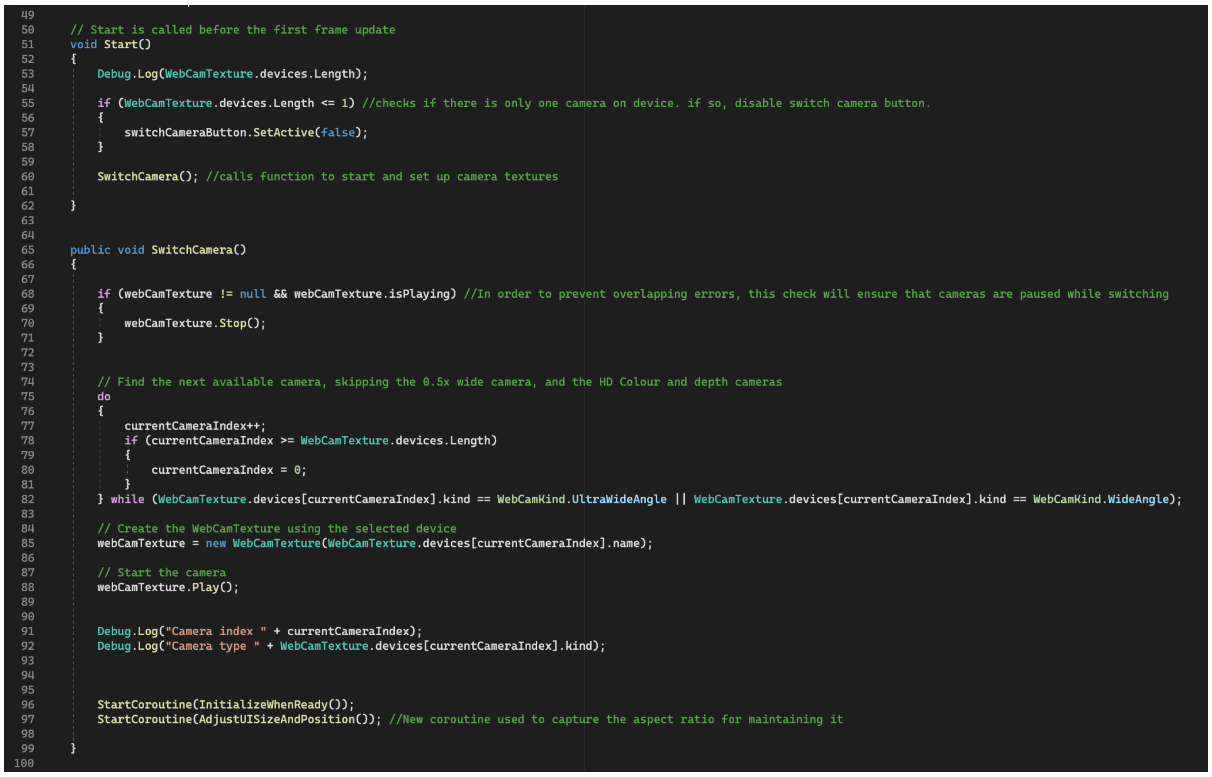
Appendices

Code Listing

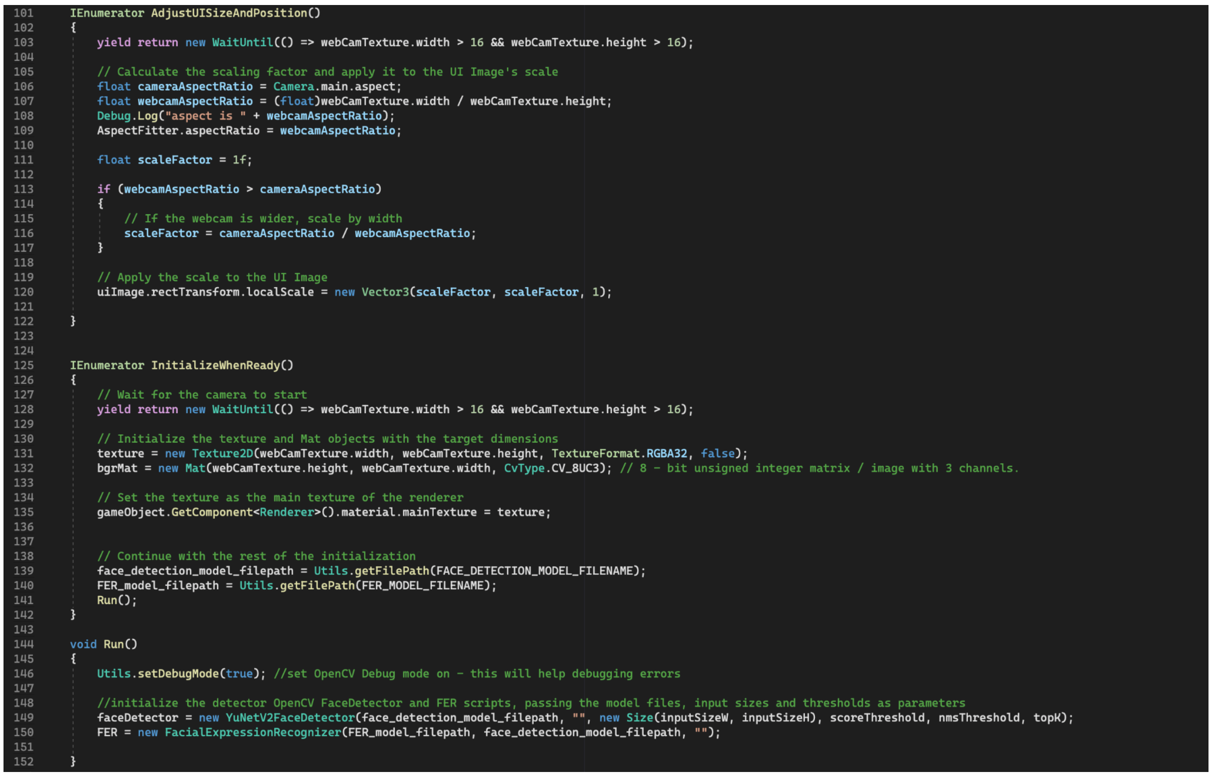
EmotionDetectionScript.cs



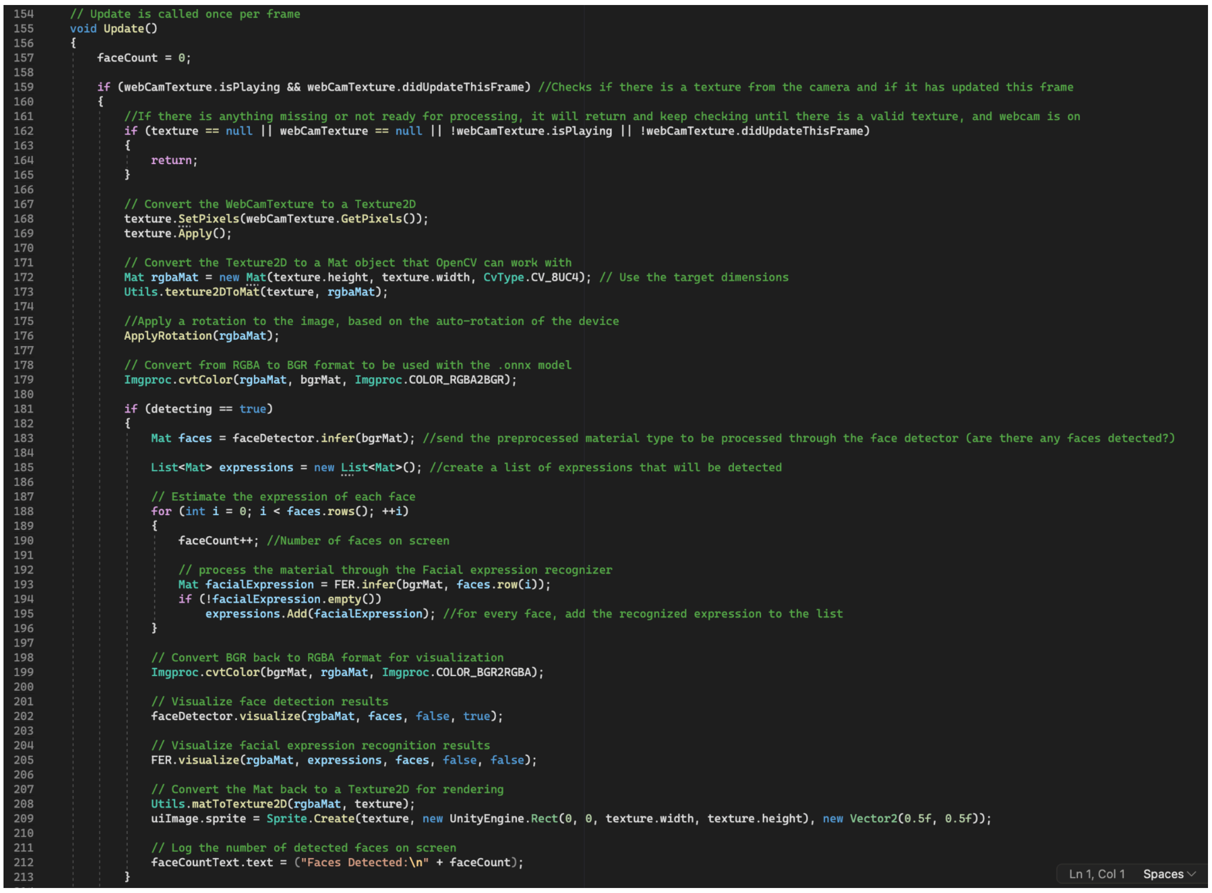
```csharp
using OpenCVForUnity.CoreModule;
using OpenCVForUnity.ImgprocModule;
using OpenCVForUnity.UnityUtils;
using OpenCVForUnityExample.DnnModel;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

using TMPro;


public class EmotionDetectionScript : MonoBehaviour
{

    Texture2D texture; //Texture to map onto quad - this is what the user will see on the other end
    WebCamTexture webCamTexture; //Texture that is pulled from the camera (updated every frame)
    Mat bgrMat; //Material to input to the model that suits OpenCV BGR format

    public bool detecting = true;

    FacialExpressionRecognizer FER;
    protected static readonly string FER_MODEL_FILENAME = "OpenCVForUnity/dnn/FER.onnx"; //FER model filename
    string FER_model_filepath;


    YuNetV2FaceDetector faceDetector;
    protected static readonly string FACE_DETECTION_MODEL_FILENAME = "OpenCVForUnity/dnn/FaceDetect.onnx"; //FaceDetection model filename
    string face_detection_model_filepath;

    //Variables used for FER processing
    int inputSizeW = 320;
    int inputSizeH = 320;
    public float scoreThreshold = 0.9f; //This is the confidence rating that the AI will need to get before displaying the result on screen
    public float nmsThreshold = 0.0f; //Non Max Suppression - selecting the best bounding box out of a set of overlapping boxes
    public int topK = 50; //The maximum number of bounding boxes to display, default 50


    private int currentCameraIndex = -1; //Track the current camera selected
    public GameObject switchCameraButton; //UI button to change the camera that is selected

    public Image uiImage; //Switching from quad to scalable UI for rendering
    public AspectRatioFitter AspectFitter; //using a built in aspectratiofitter to maintain aspect ratio and prevent letterboxes

    //public Text faceCountText;
    public TMP_Text faceCountText;

    int faceCount = 0;
```



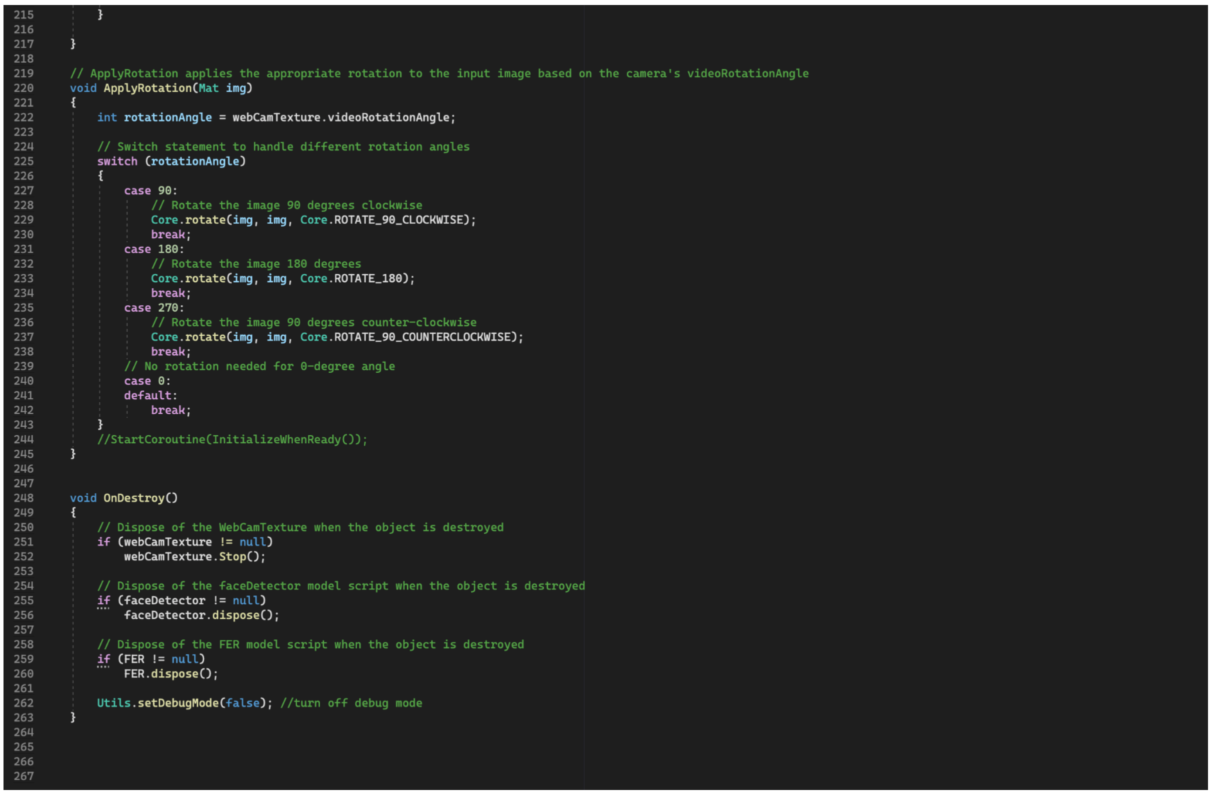
```
    // Start is called before the first frame update
    void Start()
    {
        Debug.Log(WebCamTexture.devices.Length);

        if (WebCamTexture.devices.Length <= 1) //checks if there is only one camera on device. if so, disable switch camera button.
        {
            switchCameraButton.SetActive(false);
        }

        SwitchCamera(); //calls function to start and set up camera textures

    }


    public void SwitchCamera()
    {

        if (webCamTexture != null && webCamTexture.isPlaying) //In order to prevent overlapping errors, this check will ensure that cameras are paused while switching
        {
            webCamTexture.Stop();
        }


        // Find the next available camera, skipping the 0.5x wide camera, and the HD Colour and depth cameras
        do
        {
            currentCameraIndex++;
            if (currentCameraIndex >= WebCamTexture.devices.Length)
            {
                currentCameraIndex = 0;
            }
        } while (WebCamTexture.devices[currentCameraIndex].kind == WebCamKind.UltraWideAngle || WebCamTexture.devices[currentCameraIndex].kind == WebCamKind.WideAngle);

        // Create the WebCamTexture using the selected device
        webCamTexture = new WebCamTexture(WebCamTexture.devices[currentCameraIndex].name);

        // Start the camera
        webCamTexture.Play();


        Debug.Log("Camera index " + currentCameraIndex);
        Debug.Log("Camera type " + WebCamTexture.devices[currentCameraIndex].kind);



        StartCoroutine(InitializeWhenReady());
        StartCoroutine(AdjustUISizeAndPosition()); //New coroutine used to capture the aspect ratio for maintaining it

    }

```



```
    IEnumerator AdjustUISizeAndPosition()
    {
        yield return new WaitUntil(() => webCamTexture.width > 16 && webCamTexture.height > 16);

        // Calculate the scaling factor and apply it to the UI Image's scale
        float cameraAspectRatio = Camera.main.aspect;
        float webcamAspectRatio = (float)webCamTexture.width / webCamTexture.height;
        Debug.Log("aspect is " + webcamAspectRatio);
        AspectFitter.aspectRatio = webcamAspectRatio;

        float scaleFactor = 1f;

        if (webcamAspectRatio > cameraAspectRatio)
        {
            // If the webcam is wider, scale by width
            scaleFactor = cameraAspectRatio / webcamAspectRatio;
        }

        // Apply the scale to the UI Image
        uiImage.rectTransform.localScale = new Vector3(scaleFactor, scaleFactor, 1);

    }


    IEnumerator InitializeWhenReady()
    {
        // Wait for the camera to start
        yield return new WaitUntil(() => webCamTexture.width > 16 && webCamTexture.height > 16);

        // Initialize the texture and Mat objects with the target dimensions
        texture = new Texture2D(webCamTexture.width, webCamTexture.height, TextureFormat.RGBA32, false);
        bgrMat = new Mat(webCamTexture.height, webCamTexture.width, CvType.CV_8UC3); // 8 - bit unsigned integer matrix / image with 3 channels.

        // Set the texture as the main texture of the renderer
        gameObject.GetComponent<Renderer>().material.mainTexture = texture;


        // Continue with the rest of the initialization
        face_detection_model_filepath = Utils.getFilePath(FACE_DETECTION_MODEL_FILENAME);
        FER_model_filepath = Utils.getFilePath(FER_MODEL_FILENAME);
        Run();
    }

    void Run()
    {
        Utils.setDebugMode(true); //set OpenCV Debug mode on - this will help debugging errors

        //initialize the detector OpenCV FaceDetector and FER scripts, passing the model files, input sizes and thresholds as parameters
        faceDetector = new YuNetV2FaceDetector(face_detection_model_filepath, "", new Size(inputSizeW, inputSizeH), scoreThreshold, nmsThreshold, topK);
        FER = new FacialExpressionRecognizer(FER_model_filepath, face_detection_model_filepath, "");

    }
```

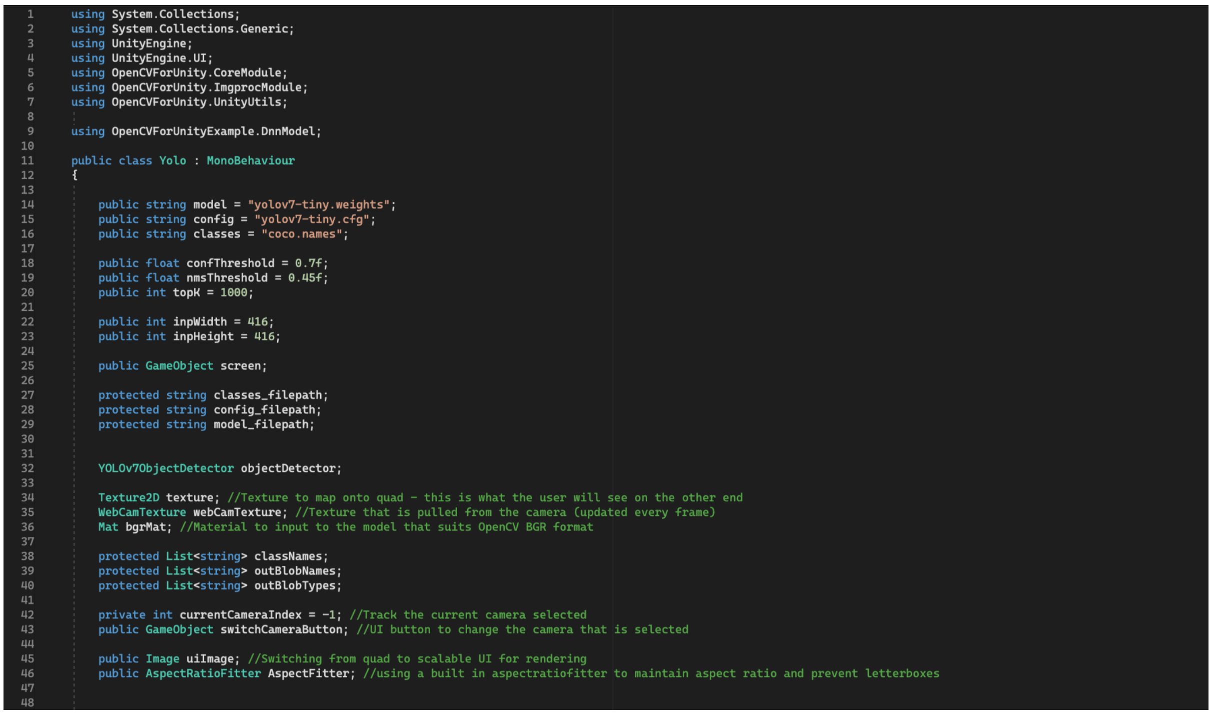


```
    // Update is called once per frame
    void Update()
    {
        faceCount = 0;

        if (webCamTexture.isPlaying && webCamTexture.didUpdateThisFrame) //Checks if there is a texture from the camera and if it has updated this frame
        {
            //If there is anything missing or not ready for processing, it will return and keep checking until there is a valid texture, and webcam is on
            if (texture == null || webCamTexture == null || !webCamTexture.isPlaying || !webCamTexture.didUpdateThisFrame)
            {
                return;
            }

            // Convert the WebCamTexture to a Texture2D
            texture.SetPixels(webCamTexture.GetPixels());
            texture.Apply();

            // Convert the Texture2D to a Mat object that OpenCV can work with
            Mat rgbaMat = new Mat(texture.height, texture.width, CvType.CV_8UC4); // Use the target dimensions
            Utils.texture2DToMat(texture, rgbaMat);

            //Apply a rotation to the image, based on the auto-rotation of the device
            ApplyRotation(rgbaMat);

            // Convert from RGBA to BGR format to be used with the .onnx model
            Imgproc.cvtColor(rgbaMat, bgrMat, Imgproc.COLOR_RGBA2BGR);

            if (detecting == true)
            {
                Mat faces = faceDetector.infer(bgrMat); //send the preprocessed material type to be processed through the face detector (are there any faces detected?)

                List<Mat> expressions = new List<Mat>(); //create a list of expressions that will be detected

                // Estimate the expression of each face
                for (int i = 0; i < faces.rows(); ++i)
                {
                    faceCount++; //Number of faces on screen

                    // process the material through the Facial expression recognizer
                    Mat facialExpression = FER.infer(bgrMat, faces.row(i));
                    if (!facialExpression.empty())
                        expressions.Add(facialExpression); //for every face, add the recognized expression to the list
                }

                // Convert BGR back to RGBA format for visualization
                Imgproc.cvtColor(bgrMat, rgbaMat, Imgproc.COLOR_BGR2RGBA);

                // Visualize face detection results
                faceDetector.visualize(rgbaMat, faces, false, true);

                // Visualize facial expression recognition results
                FER.visualize(rgbaMat, expressions, faces, false, false);

                // Convert the Mat back to a Texture2D for rendering
                Utils.matToTexture2D(rgbaMat, texture);
                uiImage.sprite = Sprite.Create(texture, new UnityEngine.Rect(0, 0, texture.width, texture.height), new Vector2(0.5f, 0.5f));

                // Log the number of detected faces on screen
                faceCountText.text = ("Faces Detected:\n" + faceCount);
            }
```

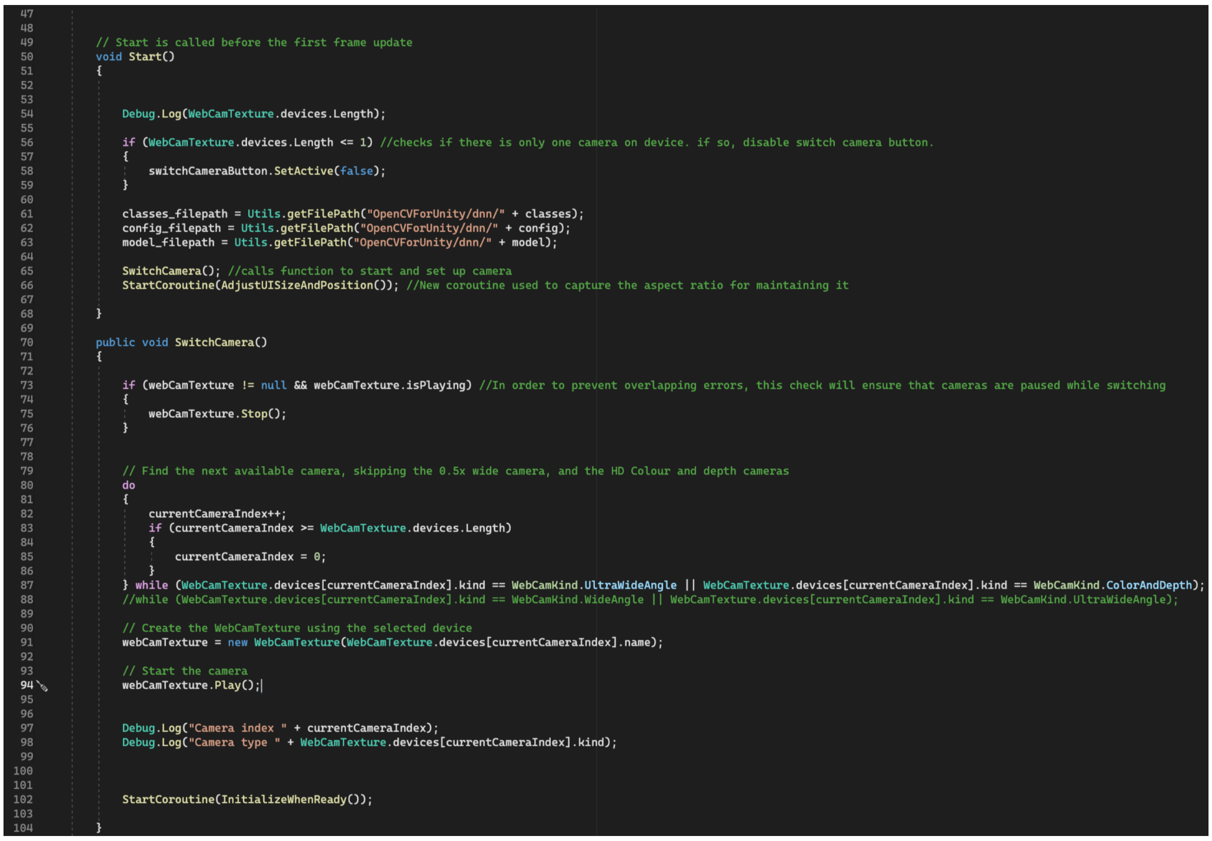


```
        }

    }

    // ApplyRotation applies the appropriate rotation to the input image based on the camera's videoRotationAngle
    void ApplyRotation(Mat img)
    {
        int rotationAngle = webCamTexture.videoRotationAngle;

        // Switch statement to handle different rotation angles
        switch (rotationAngle)
        {
            case 90:
                // Rotate the image 90 degrees clockwise
                Core.rotate(img, img, Core.ROTATE_90_CLOCKWISE);
                break;
            case 180:
                // Rotate the image 180 degrees
                Core.rotate(img, img, Core.ROTATE_180);
                break;
            case 270:
                // Rotate the image 90 degrees counter-clockwise
                Core.rotate(img, img, Core.ROTATE_90_COUNTERCLOCKWISE);
                break;
            // No rotation needed for 0-degree angle
            case 0:
            default:
                break;
        }
        //StartCoroutine(InitializeWhenReady());
    }


    void OnDestroy()
    {
        // Dispose of the WebCamTexture when the object is destroyed
        if (webCamTexture != null)
            webCamTexture.Stop();

        // Dispose of the faceDetector model script when the object is destroyed
        if (faceDetector != null)
            faceDetector.dispose();

        // Dispose of the FER model script when the object is destroyed
        if (FER != null)
            FER.dispose();

        Utils.setDebugMode(false); //turn off debug mode
    }
```

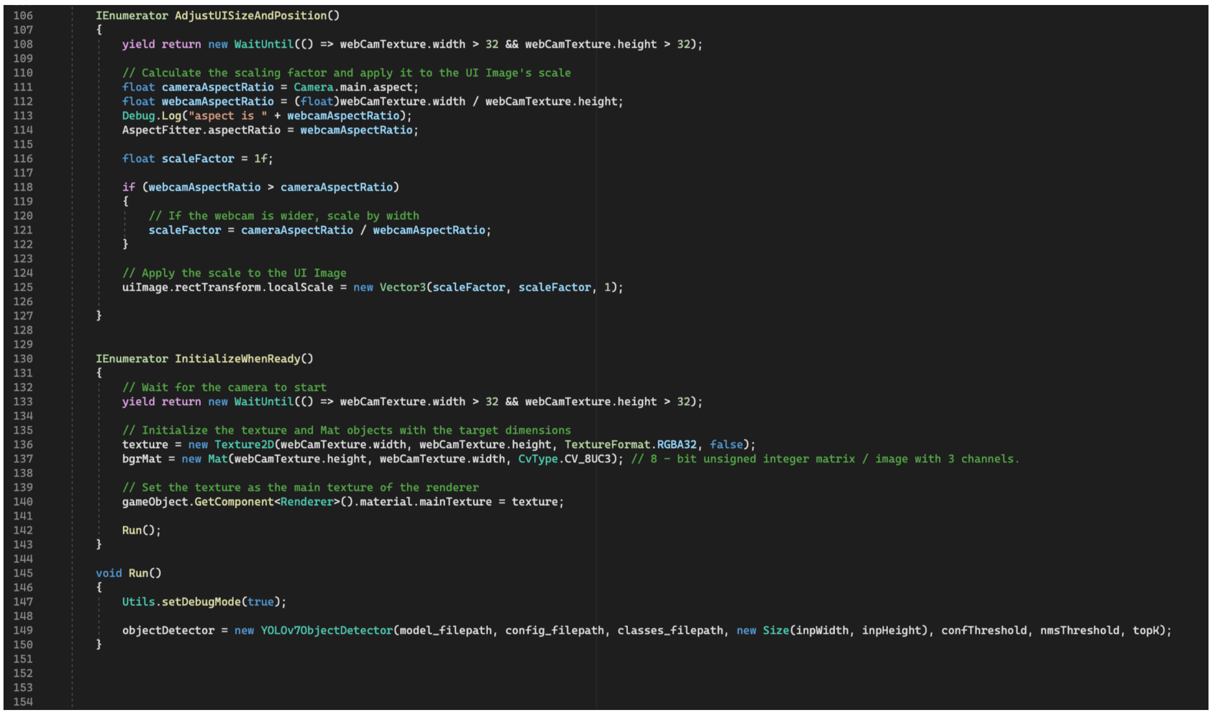
Yolo.cs



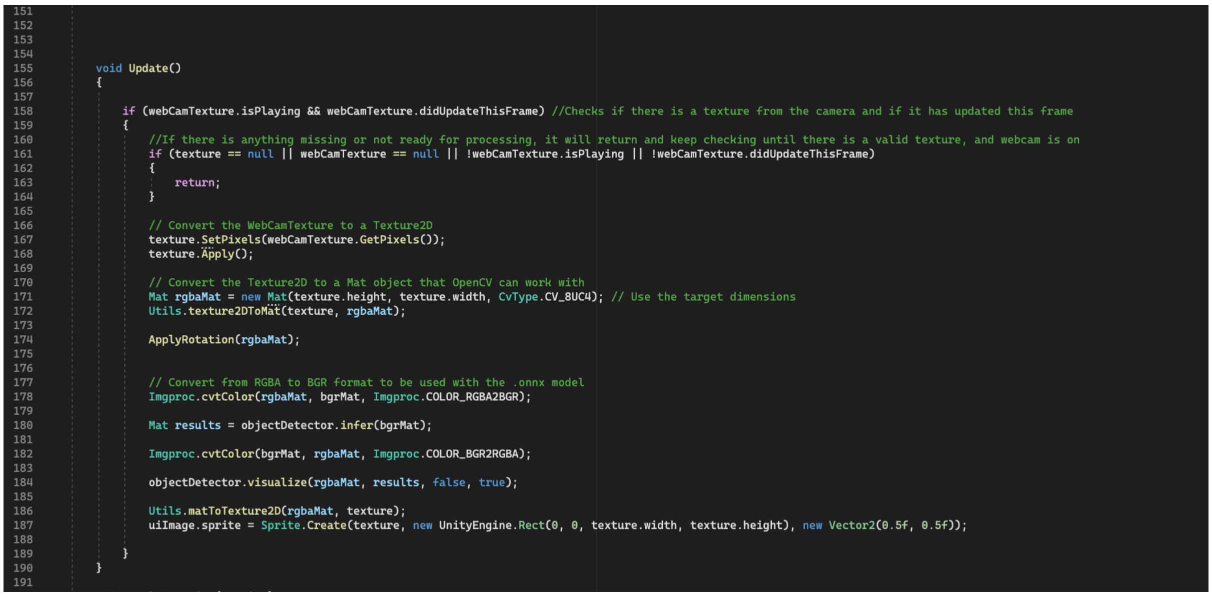
```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using OpenCVForUnity.CoreModule;
using OpenCVForUnity.ImgprocModule;
using OpenCVForUnity.UnityUtils;

using OpenCVForUnityExample.DnnModel;

public class Yolo : MonoBehaviour
{

    public string model = "yolov7-tiny.weights";
    public string config = "yolov7-tiny.cfg";
    public string classes = "coco.names";

    public float confThreshold = 0.7f;
    public float nmsThreshold = 0.45f;
    public int topK = 1000;

    public int inpWidth = 416;
    public int inpHeight = 416;

    public GameObject screen;

    protected string classes_filepath;
    protected string config_filepath;
    protected string model_filepath;


    YOLOv7ObjectDetector objectDetector;

    Texture2D texture; //Texture to map onto quad - this is what the user will see on the other end
    WebCamTexture webCamTexture; //Texture that is pulled from the camera (updated every frame)
    Mat bgrMat; //Material to input to the model that suits OpenCV BGR format

    protected List<string> classNames;
    protected List<string> outBlobNames;
    protected List<string> outBlobTypes;

    private int currentCameraIndex = -1; //Track the current camera selected
    public GameObject switchCameraButton; //UI button to change the camera that is selected

    public Image uiImage; //Switching from quad to scalable UI for rendering
    public AspectRatioFitter AspectFitter; //using a built in aspectratiofitter to maintain aspect ratio and prevent letterboxes

```



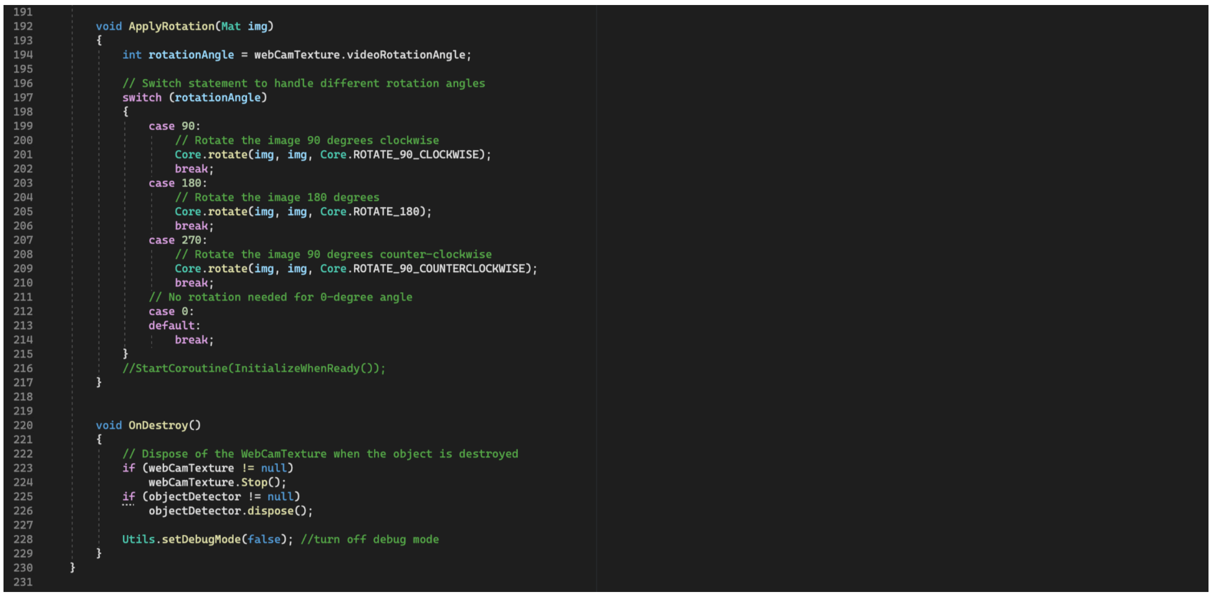
```csharp


    // Start is called before the first frame update
    void Start()
    {


        Debug.Log(WebCamTexture.devices.Length);

        if (WebCamTexture.devices.Length <= 1) //checks if there is only one camera on device. if so, disable switch camera button.
        {
            switchCameraButton.SetActive(false);
        }

        classes_filepath = Utils.getFilePath("OpenCVForUnity/dnn/" + classes);
        config_filepath = Utils.getFilePath("OpenCVForUnity/dnn/" + config);
        model_filepath = Utils.getFilePath("OpenCVForUnity/dnn/" + model);

        SwitchCamera(); //calls function to start and set up camera
        StartCoroutine(AdjustUISizeAndPosition()); //New coroutine used to capture the aspect ratio for maintaining it

    }

    public void SwitchCamera()
    {

        if (webCamTexture != null && webCamTexture.isPlaying) //In order to prevent overlapping errors, this check will ensure that cameras are paused while switching
        {
            webCamTexture.Stop();
        }


        // Find the next available camera, skipping the 0.5x wide camera, and the HD Colour and depth cameras
        do
        {
            currentCameraIndex++;
            if (currentCameraIndex >= WebCamTexture.devices.Length)
            {
                currentCameraIndex = 0;
            }
        } while (WebCamTexture.devices[currentCameraIndex].kind == WebCamKind.UltraWideAngle || WebCamTexture.devices[currentCameraIndex].kind == WebCamKind.ColorAndDepth);
        //while (WebCamTexture.devices[currentCameraIndex].kind == WebCamKind.WideAngle || WebCamTexture.devices[currentCameraIndex].kind == WebCamKind.UltraWideAngle);

        // Create the WebCamTexture using the selected device
        webCamTexture = new WebCamTexture(WebCamTexture.devices[currentCameraIndex].name);

        // Start the camera
        webCamTexture.Play();


        Debug.Log("Camera index " + currentCameraIndex);
        Debug.Log("Camera type " + WebCamTexture.devices[currentCameraIndex].kind);



        StartCoroutine(InitializeWhenReady());

    }
```



```csharp
    IEnumerator AdjustUISizeAndPosition()
    {
        yield return new WaitUntil(() => webCamTexture.width > 32 && webCamTexture.height > 32);

        // Calculate the scaling factor and apply it to the UI Image's scale
        float cameraAspectRatio = Camera.main.aspect;
        float webcamAspectRatio = (float)webCamTexture.width / webCamTexture.height;
        Debug.Log("aspect is " + webcamAspectRatio);
        AspectFitter.aspectRatio = webcamAspectRatio;

        float scaleFactor = 1f;

        if (webcamAspectRatio > cameraAspectRatio)
        {
            // If the webcam is wider, scale by width
            scaleFactor = cameraAspectRatio / webcamAspectRatio;
        }

        // Apply the scale to the UI Image
        uiImage.rectTransform.localScale = new Vector3(scaleFactor, scaleFactor, 1);

    }


    IEnumerator InitializeWhenReady()
    {
        // Wait for the camera to start
        yield return new WaitUntil(() => webCamTexture.width > 32 && webCamTexture.height > 32);

        // Initialize the texture and Mat objects with the target dimensions
        texture = new Texture2D(webCamTexture.width, webCamTexture.height, TextureFormat.RGBA32, false);
        bgrMat = new Mat(webCamTexture.height, webCamTexture.width, CvType.CV_8UC3); // 8 - bit unsigned integer matrix / image with 3 channels.

        // Set the texture as the main texture of the renderer
        gameObject.GetComponent<Renderer>().material.mainTexture = texture;

        Run();
    }

    void Run()
    {
        Utils.setDebugMode(true);

        objectDetector = new YOLOv7ObjectDetector(model_filepath, config_filepath, classes_filepath, new Size(inpWidth, inpHeight), confThreshold, nmsThreshold, topK);
    }




```

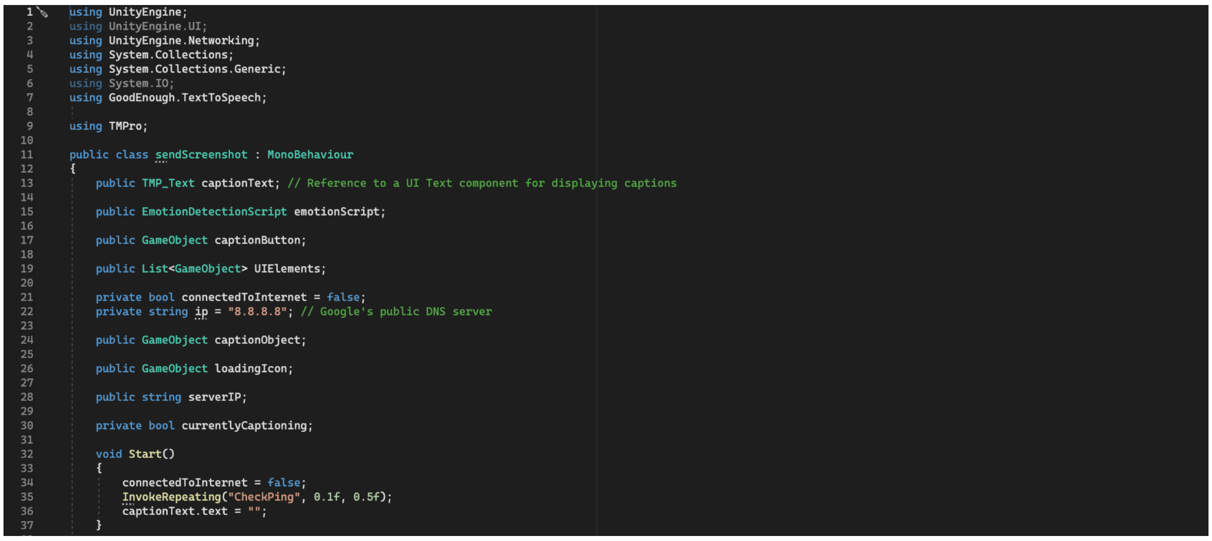


```csharp




    void Update()
    {

        if (webCamTexture.isPlaying && webCamTexture.didUpdateThisFrame) //Checks if there is a texture from the camera and if it has updated this frame
        {
            //If there is anything missing or not ready for processing, it will return and keep checking until there is a valid texture, and webcam is on
            if (texture == null || webCamTexture == null || !webCamTexture.isPlaying || !webCamTexture.didUpdateThisFrame)
            {
                return;
            }

            // Convert the WebCamTexture to a Texture2D
            texture.SetPixels(webCamTexture.GetPixels());
            texture.Apply();

            // Convert the Texture2D to a Mat object that OpenCV can work with
            Mat rgbaMat = new Mat(texture.height, texture.width, CvType.CV_8UC4); // Use the target dimensions
            Utils.texture2DToMat(texture, rgbaMat);

            ApplyRotation(rgbaMat);


            // Convert from RGBA to BGR format to be used with the .onnx model
            Imgproc.cvtColor(rgbaMat, bgrMat, Imgproc.COLOR_RGBA2BGR);

            Mat results = objectDetector.infer(bgrMat);

            Imgproc.cvtColor(bgrMat, rgbaMat, Imgproc.COLOR_BGR2RGBA);

            objectDetector.visualize(rgbaMat, results, false, true);

            Utils.matToTexture2D(rgbaMat, texture);
            uiImage.sprite = Sprite.Create(texture, new UnityEngine.Rect(0, 0, texture.width, texture.height), new Vector2(0.5f, 0.5f));

        }
    }

```

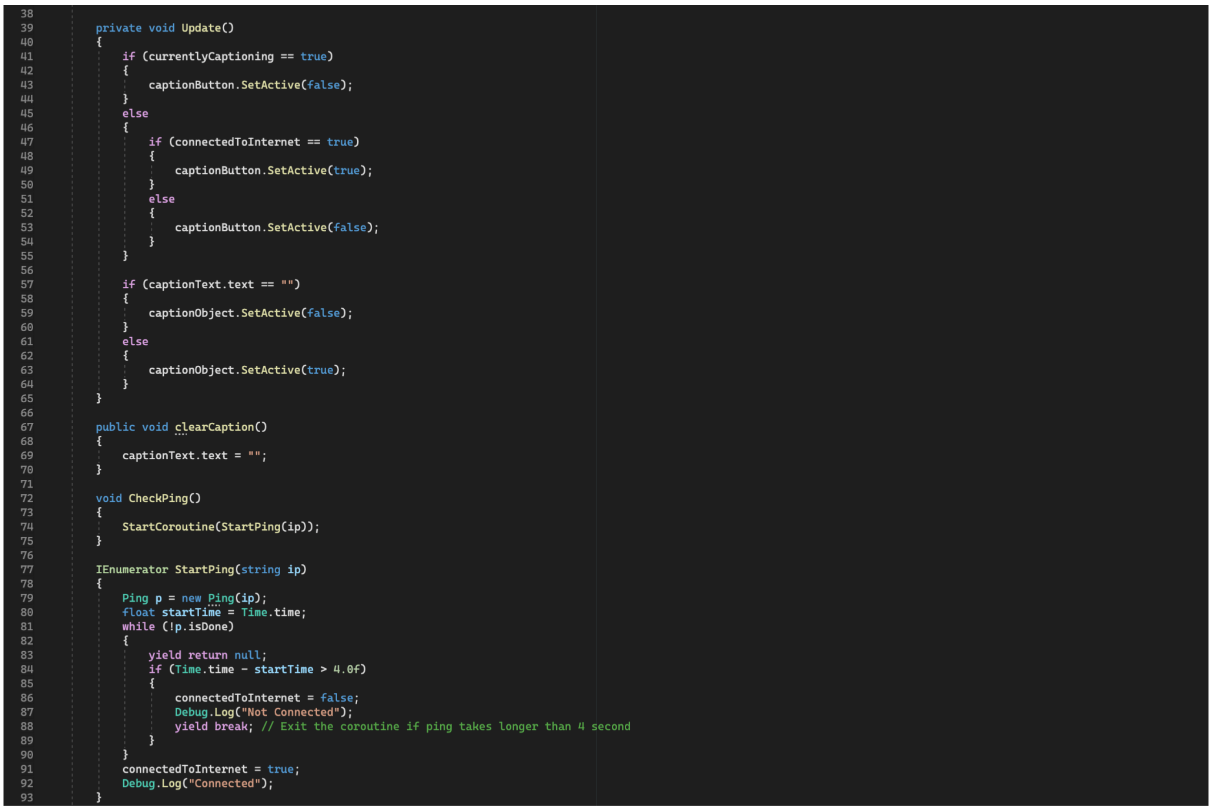


```
    void ApplyRotation(Mat img)
    {
        int rotationAngle = webCamTexture.videoRotationAngle;

        // Switch statement to handle different rotation angles
        switch (rotationAngle)
        {
            case 90:
                // Rotate the image 90 degrees clockwise
                Core.rotate(img, img, Core.ROTATE_90_CLOCKWISE);
                break;
            case 180:
                // Rotate the image 180 degrees
                Core.rotate(img, img, Core.ROTATE_180);
                break;
            case 270:
                // Rotate the image 90 degrees counter-clockwise
                Core.rotate(img, img, Core.ROTATE_90_COUNTERCLOCKWISE);
                break;
            // No rotation needed for 0-degree angle
            case 0:
            default:
                break;
        }
        //StartCoroutine(InitializeWhenReady());
    }


    void OnDestroy()
    {
        // Dispose of the WebCamTexture when the object is destroyed
        if (webCamTexture != null)
            webCamTexture.Stop();
        if (objectDetector != null)
            objectDetector.dispose();

        Utils.setDebugMode(false); //turn off debug mode
    }
}
```

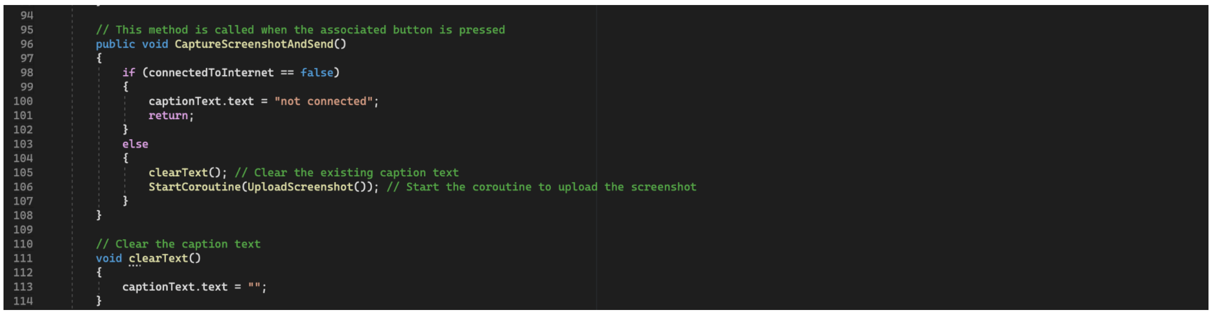
sendScreenshot.cs



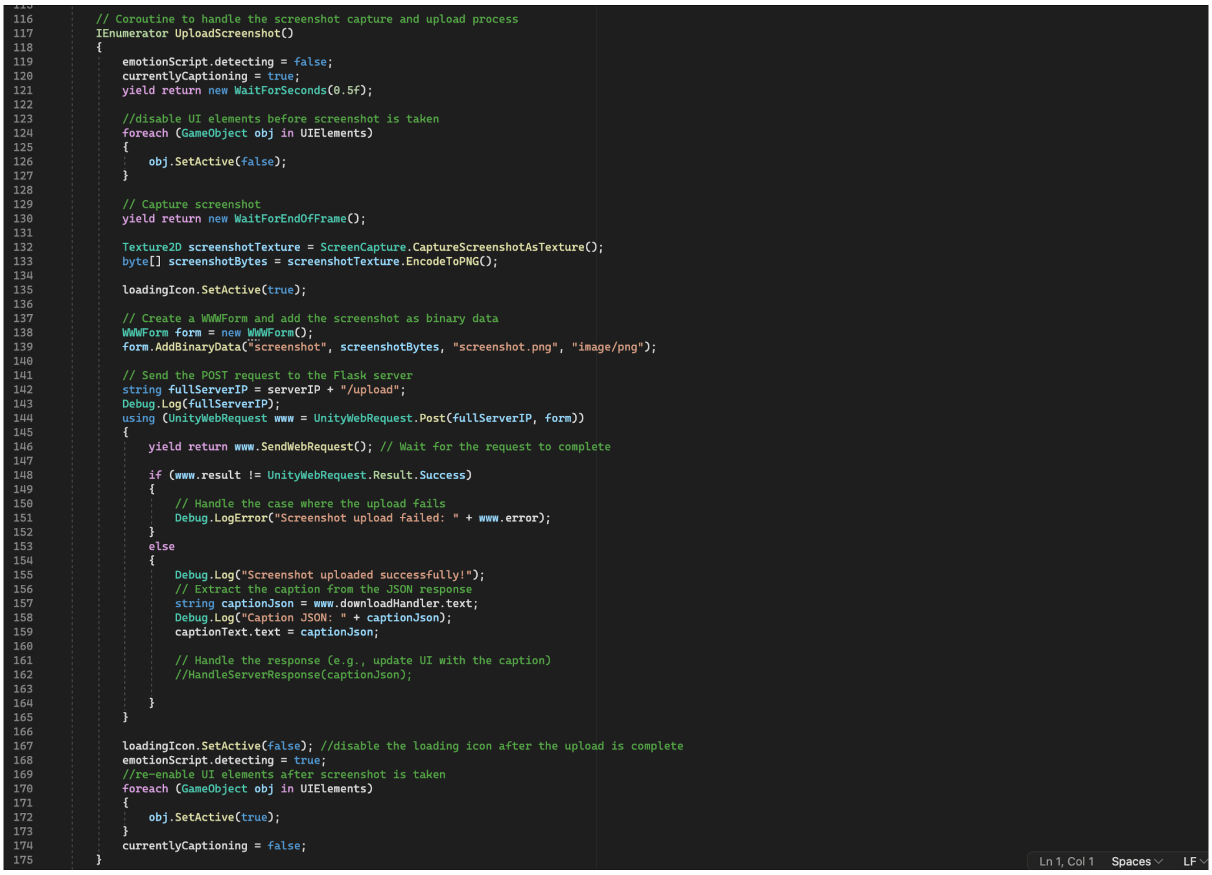
```csharp
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Networking;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using GoodEnough.TextToSpeech;

using TMPro;

public class sendScreenshot : MonoBehaviour
{
    public TMP_Text captionText; // Reference to a UI Text component for displaying captions

    public EmotionDetectionScript emotionScript;

    public GameObject captionButton;

    public List<GameObject> UIElements;

    private bool connectedToInternet = false;
    private string ip = "8.8.8.8"; // Google's public DNS server

    public GameObject captionObject;

    public GameObject loadingIcon;

    public string serverIP;

    private bool currentlyCaptioning;

    void Start()
    {
        connectedToInternet = false;
        InvokeRepeating("CheckPing", 0.1f, 0.5f);
        captionText.text = "";
    }
```



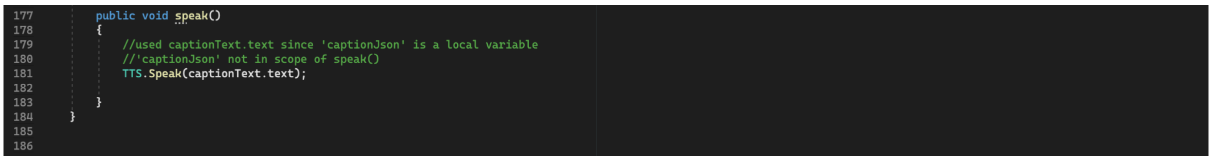
```csharp

    private void Update()
    {
        if (currentlyCaptioning == true)
        {
            captionButton.SetActive(false);
        }
        else
        {
            if (connectedToInternet == true)
            {
                captionButton.SetActive(true);
            }
            else
            {
                captionButton.SetActive(false);
            }
        }

        if (captionText.text == "")
        {
            captionObject.SetActive(false);
        }
        else
        {
            captionObject.SetActive(true);
        }
    }

    public void clearCaption()
    {
        captionText.text = "";
    }

    void CheckPing()
    {
        StartCoroutine(StartPing(ip));
    }

    IEnumerator StartPing(string ip)
    {
        Ping p = new Ping(ip);
        float startTime = Time.time;
        while (!p.isDone)
        {
            yield return null;
            if (Time.time - startTime > 4.0f)
            {
                connectedToInternet = false;
                Debug.Log("Not Connected");
                yield break; // Exit the coroutine if ping takes longer than 4 second
            }
        }
        connectedToInternet = true;
        Debug.Log("Connected");
    }
```



```csharp

    // This method is called when the associated button is pressed
    public void CaptureScreenshotAndSend()
    {
        if (connectedToInternet == false)
        {
            captionText.text = "not connected";
            return;
        }
        else
        {
            clearText(); // Clear the existing caption text
            StartCoroutine(UploadScreenshot()); // Start the coroutine to upload the screenshot
        }
    }

    // Clear the caption text
    void clearText()
    {
        captionText.text = "";
    }
```

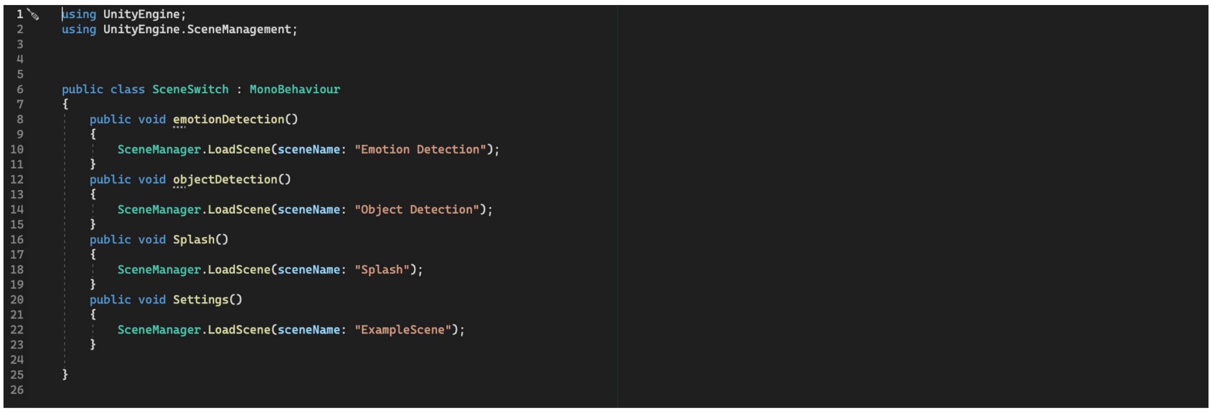


```
    // Coroutine to handle the screenshot capture and upload process
    IEnumerator UploadScreenshot()
    {
        emotionScript.detecting = false;
        currentlyCaptioning = true;
        yield return new WaitForSeconds(0.5f);

        //disable UI elements before screenshot is taken
        foreach (GameObject obj in UIElements)
        {
            obj.SetActive(false);
        }

        // Capture screenshot
        yield return new WaitForEndOfFrame();

        Texture2D screenshotTexture = ScreenCapture.CaptureScreenshotAsTexture();
        byte[] screenshotBytes = screenshotTexture.EncodeToPNG();

        loadingIcon.SetActive(true);

        // Create a WWWForm and add the screenshot as binary data
        WWWForm form = new WWWForm();
        form.AddBinaryData("screenshot", screenshotBytes, "screenshot.png", "image/png");

        // Send the POST request to the Flask server
        string fullServerIP = serverIP + "/upload";
        Debug.Log(fullServerIP);
        using (UnityWebRequest www = UnityWebRequest.Post(fullServerIP, form))
        {
            yield return www.SendWebRequest(); // Wait for the request to complete

            if (www.result != UnityWebRequest.Result.Success)
            {
                // Handle the case where the upload fails
                Debug.LogError("Screenshot upload failed: " + www.error);
            }
            else
            {
                Debug.Log("Screenshot uploaded successfully!");
                // Extract the caption from the JSON response
                string captionJson = www.downloadHandler.text;
                Debug.Log("Caption JSON: " + captionJson);
                captionText.text = captionJson;

                // Handle the response (e.g., update UI with the caption)
                //HandleServerResponse(captionJson);

            }
        }

        loadingIcon.SetActive(false); //disable the loading icon after the upload is complete
        emotionScript.detecting = true;
        //re-enable UI elements after screenshot is taken
        foreach (GameObject obj in UIElements)
        {
            obj.SetActive(true);
        }
        currentlyCaptioning = false;
    }
```



```
    public void speak()
    {
        //used captionText.text since 'captionJson' is a local variable
        //'captionJson' not in scope of speak()
        TTS.Speak(captionText.text);

    }
}
```

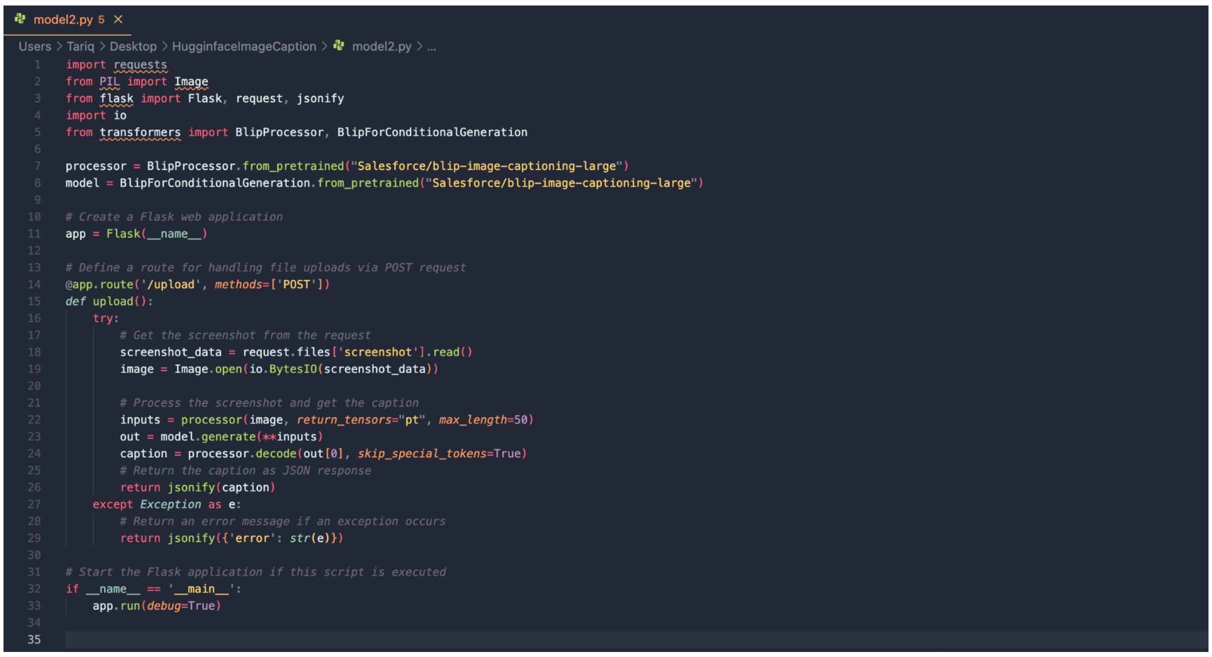
SceneSwitch.cs



```
using UnityEngine;
using UnityEngine.SceneManagement;



public class SceneSwitch : MonoBehaviour
{
    public void emotionDetection()
    {
        SceneManager.LoadScene(sceneName: "Emotion Detection");
    }
    public void objectDetection()
    {
        SceneManager.LoadScene(sceneName: "Object Detection");
    }
    public void Splash()
    {
        SceneManager.LoadScene(sceneName: "Splash");
    }
    public void Settings()
    {
        SceneManager.LoadScene(sceneName: "ExampleScene");
    }

}

```

Model2.py

(updated image captioning model)


model2.py 5
Users > Tariq > Desktop > HugginfaceImageCaption > model2.py > ...

```python
import requests
from PIL import Image
from flask import Flask, request, jsonify
import io
from transformers import BlipProcessor, BlipForConditionalGeneration

processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-large")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-large")

# Create a Flask web application
app = Flask(__name__)

# Define a route for handling file uploads via POST request
@app.route('/upload', methods=['POST'])
def upload():
    try:
        # Get the screenshot from the request
        screenshot_data = request.files['screenshot'].read()
        image = Image.open(io.BytesIO(screenshot_data))

        # Process the screenshot and get the caption
        inputs = processor(image, return_tensors="pt", max_length=50)
        out = model.generate(**inputs)
        caption = processor.decode(out[0], skip_special_tokens=True)
        # Return the caption as JSON response
        return jsonify(caption)
    except Exception as e:
        # Return an error message if an exception occurs
        return jsonify({'error': str(e)})

# Start the Flask application if this script is executed
if __name__ == '__main__':
    app.run(debug=True)

```

Note

The majority of the code for this project was continuously updated and changed. In particular, the original image captioning model was completely designed and set to be trained by myself, although due to unforeseen circumstances, I was unable to get it to work, as a result, a large amount of code is missing from the code listing, however it is all available to see in the iterative developments, along with updates throughout every other script involved.

Screencast

For the screencast, please visit this link (https://youtu.be/jL8sWUxPo5E)