Project Name: Facial Recognition App

Student 1 Name: Andrew Nechifor

Student 1 ID: 16415432

Student 2 Name: Mahjabeen Soomro

Student 2 ID: 17362496

Project Supervisor: Suzanne Little

Date finished working on the document: 06/05/2021

Table of Contents

1. Introduction

1.1 Overview

The Facial Recognition App is an application that involves the user opening their camera and being assisted in the act of taking a picture or uploading a picture in order to have a facial recognition model applied to the image. Once the model has analyzed the picture, the results are returned and displayed to the user. The application will inform the user of whether or not they are wearing a mask and if they are wearing the mask correctly or incorrectly. The facial recognition system is developed, trained and tested using a large, public and publicly available online dataset of thousands of images that contains many images of faces with both masks on and masks off. The user will also have the ability to upload their own dataset and have it analysed as well. The goal of this project is to provide users with the ability to detect if someone is wearing a face mask and if that person is wearing the mask correctly or incorrectly.

1.2 Glossary

User interface - The visual way in which a user and a machine system interact.
Machine Learning - A computer system that can learn and evolve through the use of algorithms and data models to perform tasks.
Facial Recognition - The ability of a computer system to recognize facial features in an image.
Application Program Interface - A set of functions and methods that allows the creation of programs or applications.
Frontend - The part of the app with which the user interacts directly.
Backend - The part of the app that is not directly accessed by the user. This will be responsible for receiving, storing and manipulating data. The results of which will be sent back to the frontend.
Integrated Development Environment - This is a system that provides numerous features to software developers for software development
GDPR - This is a set of laws that sets guidelines for the collection and processing of personal information from people who live within the European Union.

2. Research and Motivations

2.1 Research Performed

We began our research on the 5th of October, 2020. We were initially informed of CA Final Year Project ideas supplied by the CA staff by the module co-ordinator. We browsed many of the project ideas in order to understand what the scope of a project could entail and what different areas could be approached by our project. Many different areas of software engineering were covered by all of the project ideas but we were the most captivated by the ideas that utilized large databases of media like photos and videos.

We began discussing what areas we wanted this project to cover and we came to the agreement that facial recognition was the most fascinating subject for both of us. We knew that facial recognition was a vast and evolving area within the field of software engineering. Facial recognition is regularly used in everyday life in the form of the lock screen on people's phones for example. With its constant presence and continuous developments, the research performed needed to be accurate and straight to the point. The global pandemic caused by the novel coronavirus was constantly relevant in our lives and as such, it was taken into account during the researching process. Many articles, chat forums and social media pages were engaged in the conversation of mask usage and facial recognition systems during the research phase of our project. At the time, we googled "facial recognition" and "facial recognition mask" to see many of these articles and discussions.

The results of our initial research gave us vital information in the form of what facial recognition is defined as and what it is not. As well as that, we learned about the sentiments and ethical concerns that some people shared about facial recognition as a system. We noted these findings as we thought they would be useful for us in the future. We decided to further refine our research strings by googling "facial recognition technology" and "facial recognition works" in order to view examples of facial recognition systems and to inspect how these systems operate in detail. This gave us insights into the widespread use of facial recognition, the applications of this technology and the techniques involved like face analysis and machine learning.

After gathering and collecting the results of our research, we firmly chose to work on a project that was involved in the area of facial recognition and to relate the project to the current global state of affairs. Finally, after a couple iterations we conceptualized the idea of using an app to perform facial recognition on a person to detect whether a person is wearing a mask or not and whether they are wearing correctly or not.

2.2 Motivations

Many factors came into play when deciding what our project was going to be and why we chose this particular idea. Conceptualizing an idea and following through with that idea can be difficult tasks to perform without properly justified motivations. First and foremost, we decided to undertake a facial recognition app as our project because the idea felt very relevant and topical for the current situation taking place around the world. Wearing masks in public is now a global and daily part of most people's routine around the world. Previously, countries in Asia regularly wore masks in public but due to the novel coronavirus, that practice has become global. Being able to recognize if someone is wearing a mask and whether that mask is being worn correctly or not is a task humans perform all the time, but transferring this task to an automatic, autonomous system has other benefits that we were motivated by.

Our app has many potential applications in a global business setting. This was another motivation behind our research and project. This facial recognition system can help keep businesses safer and cleaner by analyzing the faces of the customers that enter an establishment to see if they are wearing a mask properly. As well as that, employees can also use this system to ensure that they are correctly wearing a mask. Ensuring every customer and every employee on the premises has an appropriate mask worn can help guarantee a clean and safe environment for the business. This improves the retail experience for both customers and employees. This is not only limited to retail shops, as this can be used in banks, colleges and libraries to ensure the safety of everyone on the premises.

Lastly, we were motivated by our keen interest in the field of machine learning. We were interested in facing the unique challenges and making use of the opportunities that this area gives to developers. Researching the difficulties in developing a facial recognition model motivated us to work in this area. Performing a balancing act of knowing how much to train a model, how much to test a model, how to verify its accuracy and how to integrate the model into a mobile application seemed appealing to both of us.

3. System Architecture

3.1 Overall System Architecture

The application and complete frontend architecture of the project was completely developed using Android Studio. The user interface and functionality were all fully handled using this IDE. Android Studio utilizes the Java object-oriented programming language which helps with the structure and design of the application and the user interface. Within Android Studio are virtual machines and shortcuts for error handling which makes the creation, modification and evaluation of each element of the architecture and user interaction easier to perform.

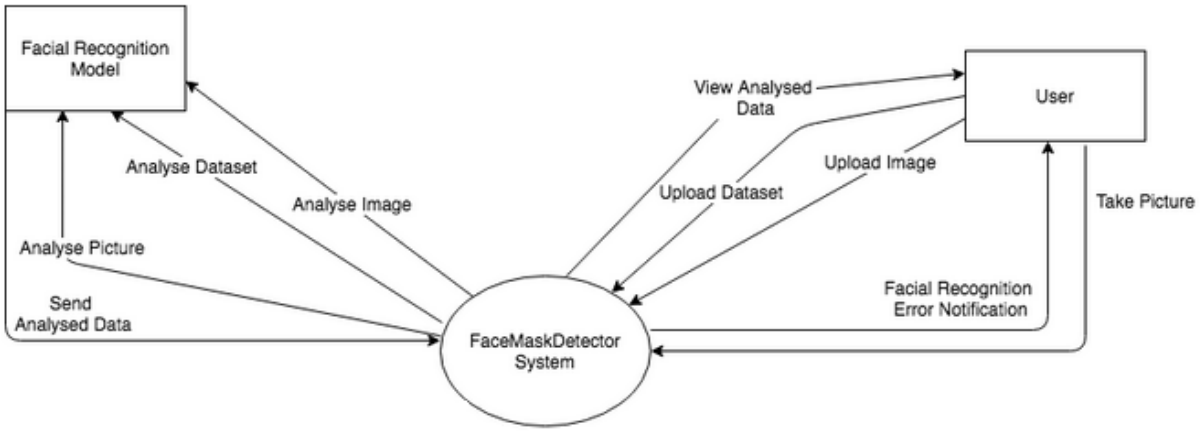
The structure of the user interface was designed to be as understandable and simple as feasibly possible. When the user enters the application for the first time, they enter the home page of the app. They will see the main page with two buttons, one labelled as ‘ Take Picture’ and the other as ‘Select Image’. The user will choose what they wish to do. If the User clicks ‘Take Picture’, the app will bring the user to a gallery view of images which the app has permission to access the gallery images from the device and display it on the app. There will be a camera icon at the top right hand bar which the user will click so the app accesses the device's camera to take a picture. Once taking the picture, the user has a choice to retake a picture if they are not happy with the picture taken or go ahead with the picture. This picture will then be stored in the gallery for the user to select it and then once selected, the training model will be run on the picture and will return a box around the face detected and if a mask has been detected on the face with the percentage of how well a mask has been worn or if there was anything else detected on the face. This depends on the lighting of the picture, what angle the face is in, how many faces are detected, how far away the faces are in the picture etc. If the user decides to select the button to ‘Select Image’, they will be brought to the gallery of images and select an image to run the model on and be returned with the results of if the user is wearing a mask or not.

The design of the backend involves many functions and modules interacting with each other. When the user inputs an image into the facial recognition system, the app has a trained model ready in order to perform the analysis on the picture. When the app performs this analysis, the results are sent to the frontend in order to display the data and any temporary image or data that the backend has processed is automatically deleted.

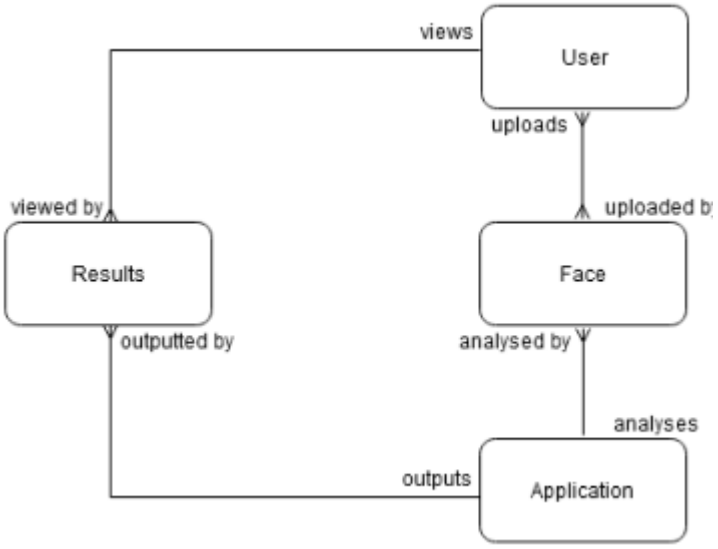
The facial recognition model is trained using an online dataset with thousands of publicly available faces of people with masks both on and off. Bigger sample sizes provide more accurate mean values and smaller margins of error while also aiding in the discovery of outliers that could skew the data that would be caused by a smaller sample size. The model is trained and tested twice with this dataset in order to verify the accuracy of the facial

recognition system. The training and testing data ratio is a split between 85 / 15, we felt that this was a fair split considering the large dataset size we have. This was to avoid the issues of overfitting where systems have good performance on the training data but poor performances on other data. This was also done to avoid underfitting which involves the system having a poor performance on the training data and a poor performance on other data as well.

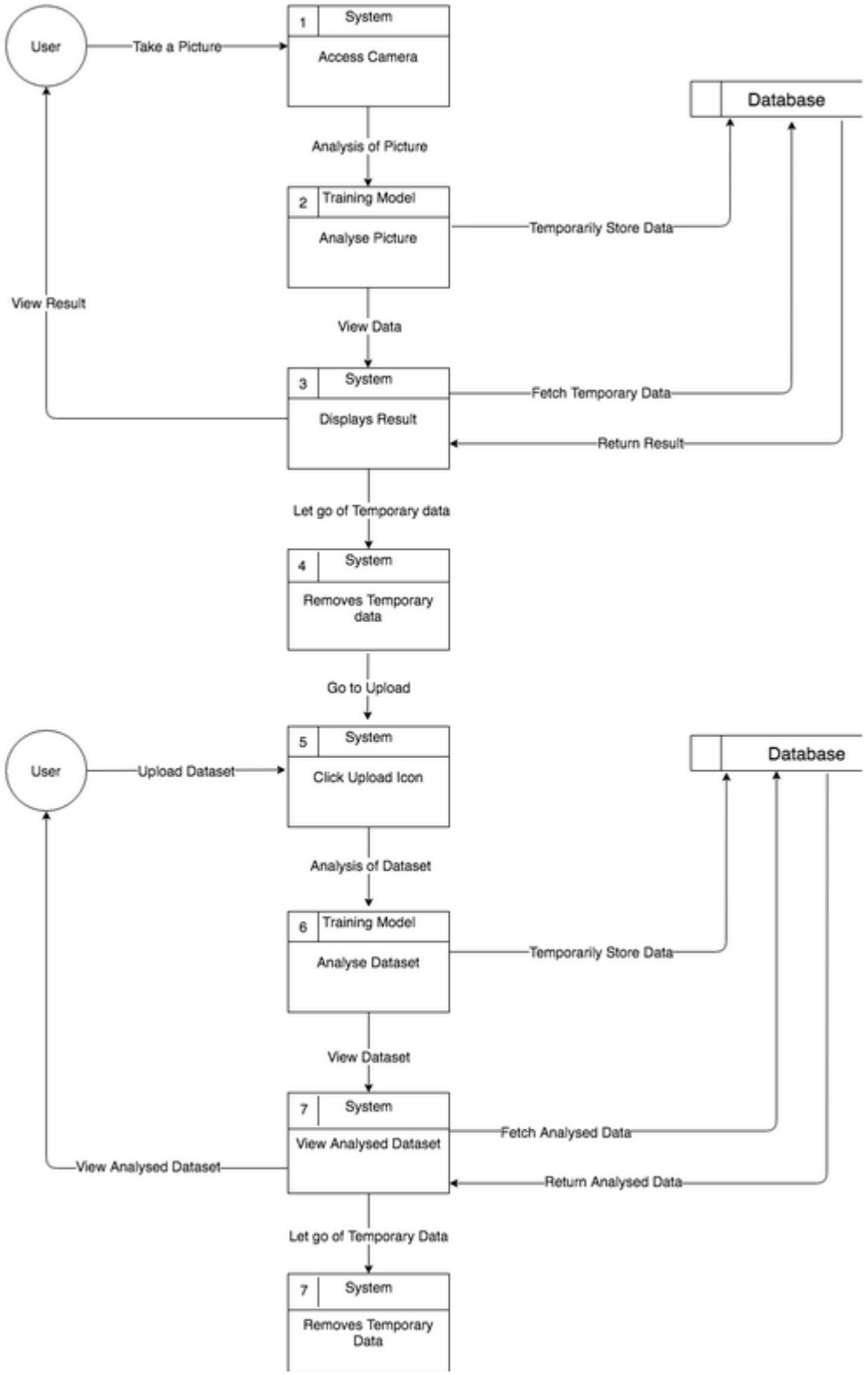
Context Diagram



Logical Data Structure



Data Flow Diagram



4. Implementation

4.1 Implementation Choices

A major implementation choice that we made was the decision to carry out the facial recognition calculations of the application itself. Instead of taking the popular approach of incorporating a server that performs the facial recognition for the application, our app performs the facial recognition on the uploaded images on the phone itself. The biggest factor that affected this decision was the ethical concerns we needed to consider for this project. Handling sensitive data like the facial images of a person in the most optimal appropriate manner was a top priority for this project. Servers can cause extra security risk in the form of images potentially being stolen as they are being sent on their way to the server or from the server itself. Eliminating this risk by not using servers and having the application conduct the operations on the mobile device creates a safer environment for any data and for the user. The user no longer needs to worry about someone accessing the application and taking their data and no longer requires a constant and stable internet connection to use the app as well. Another advantage to this approach is the removal of the costs and labour associated with establishing and maintaining servers. Servers can be resource intensive and computationally expensive to constantly run and maintain on smaller or cheaper home machines which would increase the cost of the continuous development, maintenance and evaluation of the facial recognition system and the application. Extra clarity can be achieved from implementing the facial recognition system onto the application itself as well. Any bug fixing or error handling can be performed on the app, instead of attempting to find whether an issue is server side or client side or both. In order to achieve an adequate standard of quality to our project, it was necessary to make this implementation choice of performing the facial recognition system on the application itself. However, we must acknowledge the downsides of this implementation choice. This increases the minimum amount of processing and computing power required from the mobile device in order to utilise the application. Resources like the memory allocated or how much CPU usage space is required will now need to increase. These are the tradeoffs we knew we were making and given the nature of the project, we believe that the benefits far outweigh the disadvantages stated for this implementation choice.

Another significant implementation choice was our decision to have the application automatically delete any data or images received after the user has performed their tasks. Taking into account the ethical considerations for any project is vital before commencing the development of an app. Handling the personal data submitted by a user, in this case a picture of their face, is a key ethical consideration that must be examined. Avoiding the storage of any personal data and removing any private information submitted by the user after they have finished using the app are the implementation choices we decided to apply to our project. This allows us to have an ethical project that still maintains the core functionality of the app itself.

Finally, the last important implementation choice we carried out was the structure and design of the user interface of the application. We decided to create a user interface that was as simple and easy to use as possible. This was done to increase the quality of the user experience through high comfortability and high navigability for the person using the app. This was done by having two buttons for the different options to choose from for the user. There was nothing too complicated for the user to do but take a picture or select an image and upload it so the model can perform the analysis. The gallery of the device accessed by the app to allow for users to select images they have stored on their device was for the ease of the user or for the user to be able to take a picture in the app instead of having to go take a picture from the actual camera of the device to then select the image in the app for the ease of the user and then the picture taken to store that in the device's memory and then be able to perform an analysis on that picture.

4.2 Sample Code

Analysis of facial recognition system code

The facial recognition code started off with creating a face detector in the onActivityResult() of the MainActivity file. Using the FaceDetector.builder(), it was built and then if it was not isOperational, then return the message that it failed to operate. Next the faces would have to be detected from the bitmap of the image whereby a frame was built using Frame.Builder() and to send the bitmap of the image. Once the face is detected, call the frame and build it around the face. This code is seen in the facedetector image below.

Each face that was defined was then checked with the size of the face, left, right, top and bottom of the face was checked. The Paint() function was then called to create a box in the style of a stroke around the face, i.e. a line box around the face. The result class was called to detect if a mask was worn on the face detected previously in the image i.e. to run the model on the image. If a mask is worn, then a green box will appear around the detected face but if a mask is not worn then a red box will appear around the face with the label, mask or no mask and a percentage. This code can be seen in the defined faces image below.

The result class consisted of loading the model.tflite, and using the Interpreter on the model. The labels.txt were also loaded. The type of data and the shape of the input and output image was checked with .dataType() and .shape(). Buffer was called and then the preprocessing of the image began. The buffer was used to load the image with .load(input) to apply the preprocessing and then the model was run on the image. The results of the model were received and the output returned as a float using .mapWithFloatValue. This can be seen in the loadmodel image below.

Screenshots of facial recognition code

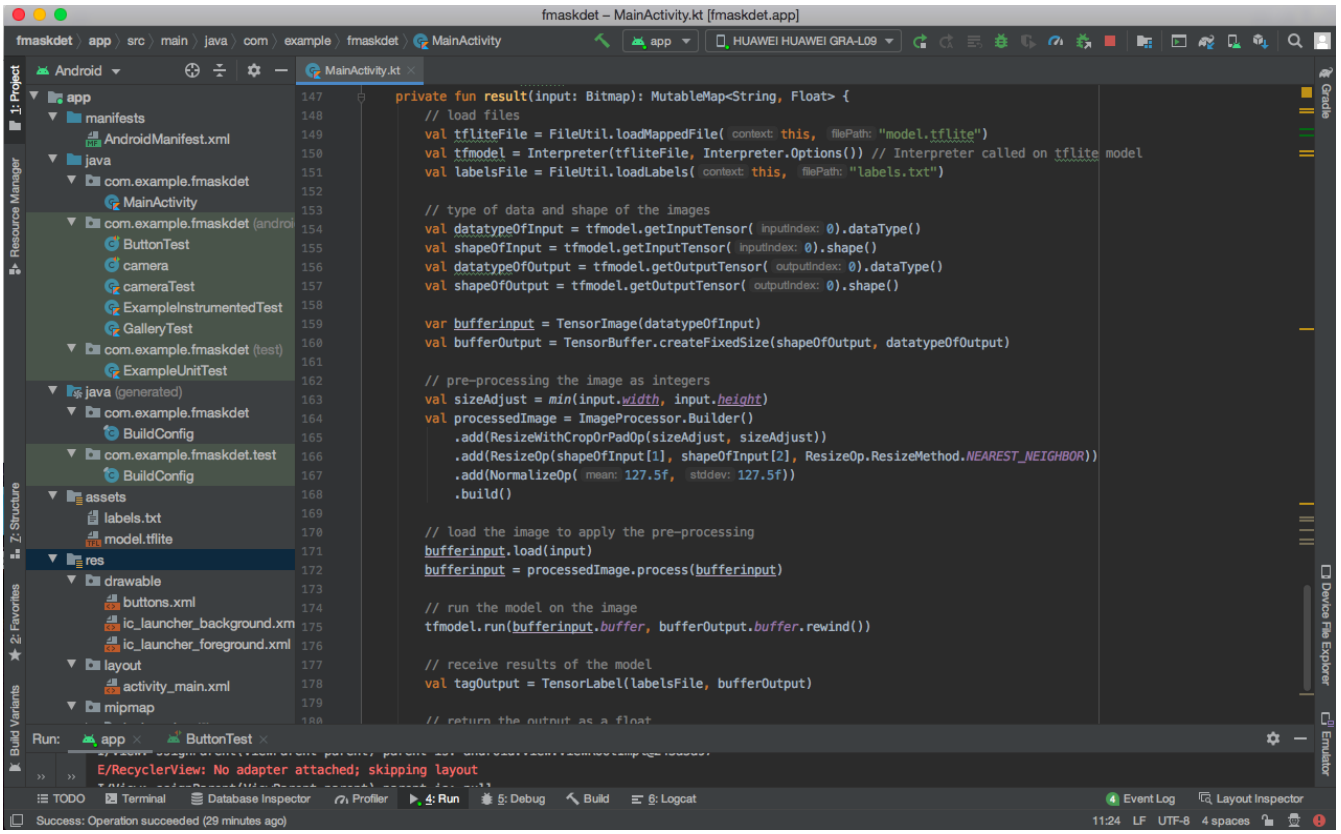
1. facedetector

```
// Creating a face detector
val myfacedetector = FaceDetector.Builder(applicationContext).setTrackingEnabled(false)
    .build()
if (!myfacedetector.isOperational) { // if the face detector is not operational, send a message saying that it failed
    AlertDialog.Builder( context: this)
        .setMessage("The face detector failed!")
        .show()
    return
}

// Detect the faces in the bitmap of the image
val aFrame = Frame.Builder().setBitmap(currentBitmap).build() // build a frame on currentBitmap(image)
val face = myfacedetector.detect(aFrame) // value to detect faces
```

2. definedfaces

```
// display the defined faces detected
for (i in 0 until face.size()) {
    val aFace = face.valueAt(i)
    val leftofface = aFace.position.x
    val rightofface = leftofface + aFace.width
    val topofface = aFace.position.y
    val bottomofface = topofface + aFace.height
    val bitmapAdjusted = Bitmap.createBitmap(currentBitmap, leftofface.toInt(), topofface.toInt(),
        if (rightofface.toInt() < currentBitmap.width) {
            aFace.width.toInt()
        } else { currentBitmap.width - leftofface.toInt() },
        if (bottomofface.toInt() < currentBitmap.height) {
            aFace.height.toInt()
        } else { currentBitmap.height - topofface.toInt()})

    // call Paint function
    val paintIt = Paint()
    paintIt.strokeWidth = 3F
    paintIt.style = Paint.Style.STROKE // create a line box around the face

    // call result class whether there is a mask or not
    val tag = result(bitmapAdjusted)
    var guess = ""
    val mask = tag["Mask"]?: 0F
    val nomask = tag["NoMask"]?: 0F

    if (mask > nomask){
        paintIt.setColor(Color.GREEN)
        guess = "Mask : " + String.format("%.0f", mask*100) + "%"
    } else {
        paintIt.setColor(Color.RED)
        guess = "No Mask : " + String.format("%.0f", nomask*100) + "%"
    }
```

3. loadmodel



Code and analysis of user interface code

The activity_main file was used to design the interface of the application. The ImageView was where the photo taken from the camera or the image selected from the gallery would be viewed. The sizing of where it would fit on the screen was adjusted to allow the image to be clearly viewed by the user. This was done with the android:layout_width and android:layout_height. Id of the ImageView was set to 'viewimage' so it could be called in the MainActivity to set the analysed image to be displayed. Two buttons were added, sizes were adjusted for ease of click, colour was also adjusted by creating a separate buttons file and called in the activity_main with android:background=@drawable/buttons. The text in the button was fit to size where the user can easily read it. Id's of both buttons were created.

In the MainActivity file, listeners were created for both the buttons using their Id. For photoselect where the user would 'Take Photo', the user is able to click on the camera icon which is coded with .showCamera(true). A single image can be selected from the gallery through .single(). In the imageselect button, the user will access the gallery only to select an image, which is why the camera is set to false, .showCamera(false). This is because ImagePicker as default has the camera intent.

1. Activity_main.xml

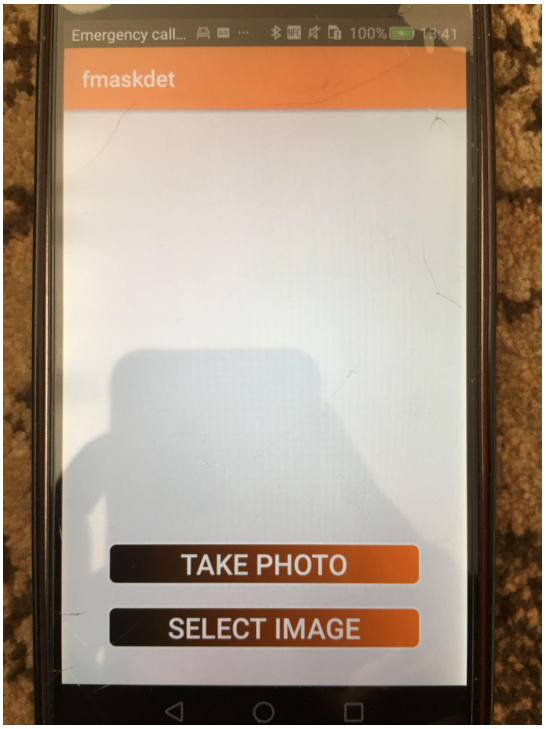
```
<ImageView
    android:id="@+id/viewimage"
    android:layout_width="408dp"
    android:layout_height="562dp"
    android:layout_marginStart="2dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="2dp"
    android:layout_marginBottom="32dp"
    app:layout_constraintBottom_toTopOf="@+id/imageselect"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:src="@tools:sample/avatars" />

<Button
    android:id="@+id/photoselect"
    android:layout_width="279dp"
    android:layout_height="38dp"
    android:layout_marginBottom="88dp"
    android:background="@drawable/buttons"
    android:text="Take Photo"
    android:textColor="@color/ef_white"
    android:textSize="25sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent" />

<Button
    android:id="@+id/imageselect"
    android:layout_width="279dp"
    android:layout_height="38dp"
androidx.constraintlayout.widget.ConstraintLayout › ImageView
```

2. MainActivity.kt

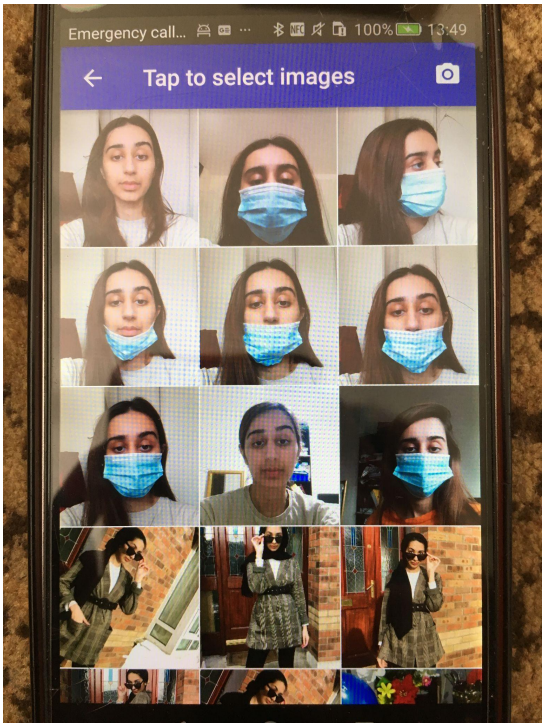
```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    // listener for take photo button
    photoselect.setOnClickListener { it: View!
        ImagePicker.create(this) // jitpack.io
                .showCamera( show: true) // camera icon shown- easier way of adding camera to the button
                .single() // only one image can be chosen
                .start() // start the events
    }
    // listener for select image button
    imageselect.setOnClickListener { it: View!
        ImagePicker.create(this)
                .showCamera( show: false) // Don't need camera
                .single()
                .start()
    }
}
```

Screenshots of user interface

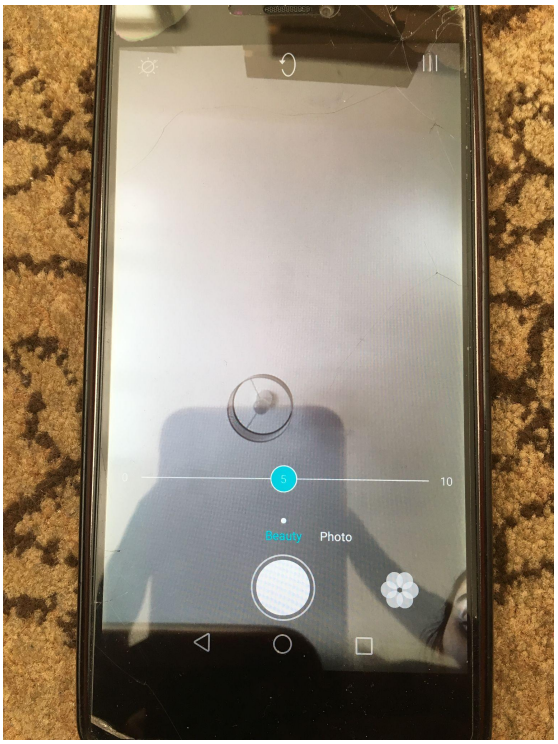
1. Main Page



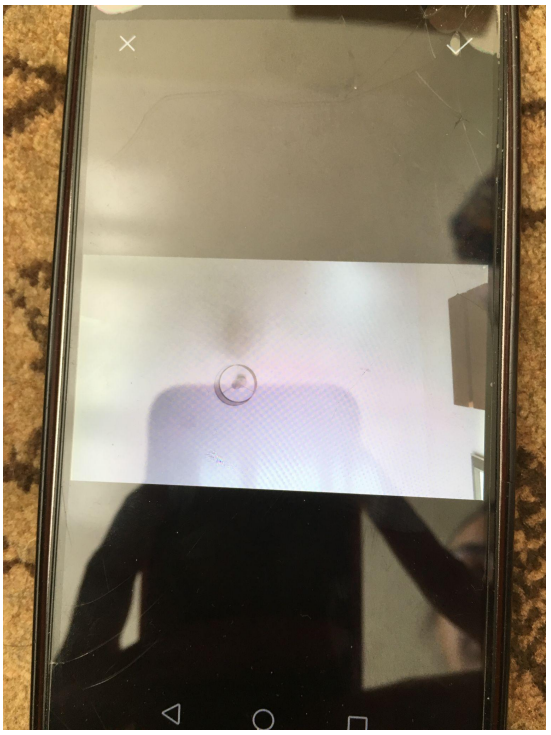
2. Take Photo



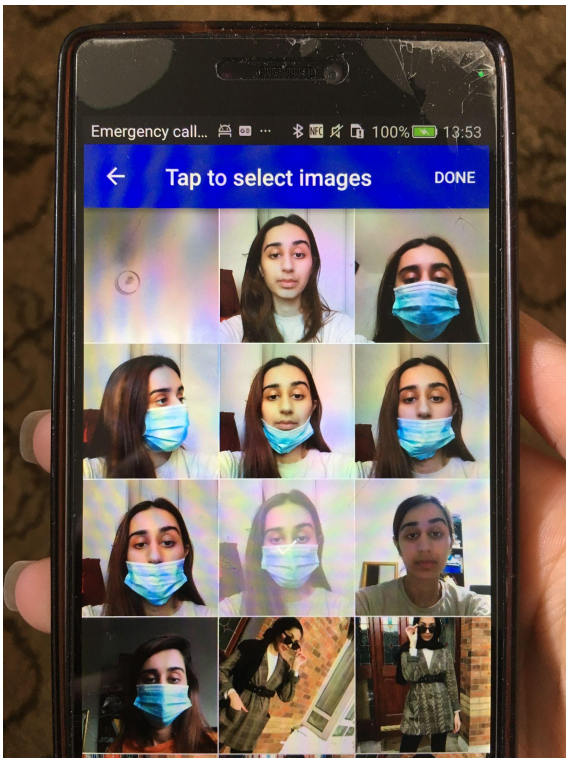
3. Camera open



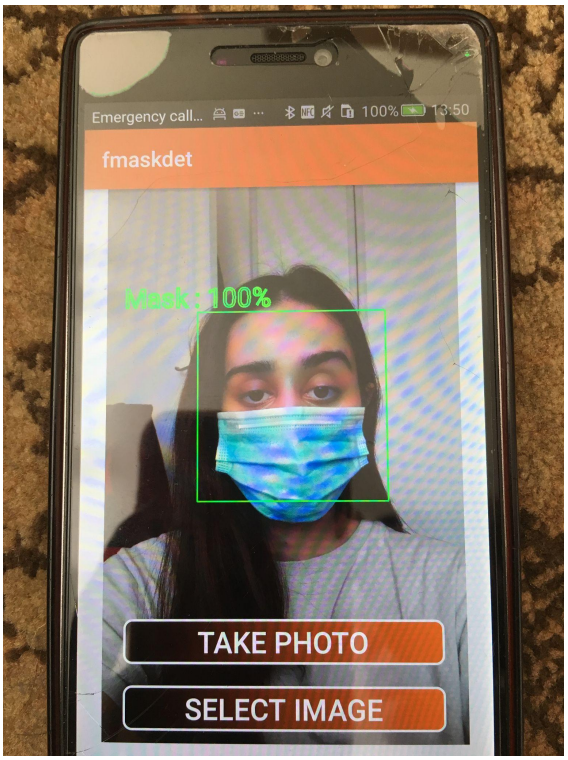
4. Capture



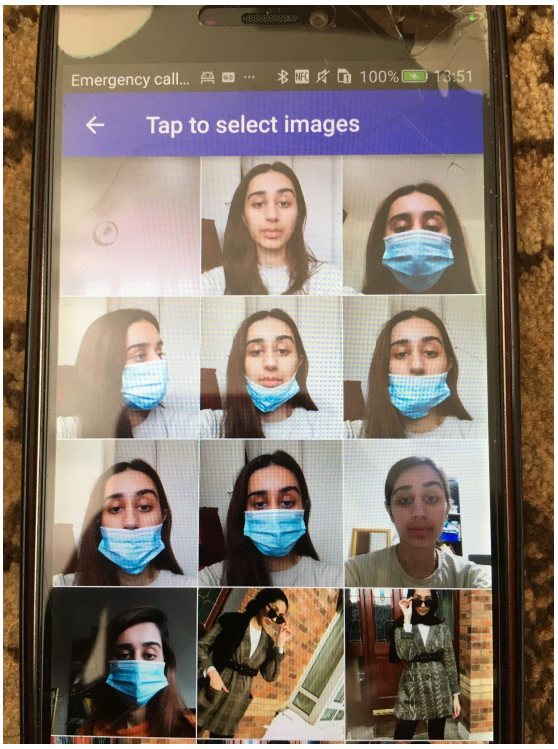
5. Choose Capture



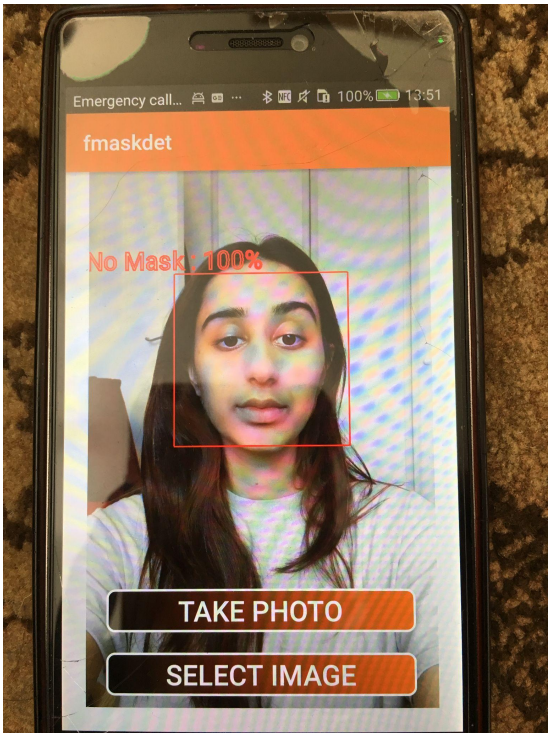
6. Mask- result



7. Select Image



8. No mask- result



5. Problems and Resolution

5.1 Facial Analysis Issues and Solutions

The largest issues we faced during the implementation of the facial recognition model was dealing with images that had faces in poor lighting or faces that were not taking directly straight at the camera but from other angles instead. The changing of the angle at which you view a face moves the points that the model uses in order to recognize facial features within the image and as such, can make it harder to detect if there is a face or if there is a mask being worn on a face. As well as that, images that contained poor lighting often obscured facial features that made facial detection difficult as a result. This problem is compounded by how the facial recognition system detects whether a face is wearing a mask or not. If there are certain features missing within the image, but the overall shape of the face can be recognized, the system classifies the face within the image to be wearing a mask. So this causes some images where the face is in poor lighting or if the image is taken at a different angle to be classified as a mask wearing image. This is not easily solvable as we found out in our research. The best solution we found was to increase the amount of training performed and to increase the size of the training data. While our accuracy scores did improve, this may be a discrete occurrence of slight overfitting as well so this must be taken into account.

5.2 Application Issues and Solutions

The application was created, modified and evaluated using Android Studio as the IDE. We faced many bugs and issues during the development of the app. When attempting to set the name of the application in the virtual machine, the renaming would not occur. We googled this and this is a well documented issue in Android Studio and the fix provided to work around this issue involved clicking the Run button again to re-deploy the application. Another issue we seemed to face was the android application works on most android devices, however when we tested on two different android devices, the camera intent seemed not to work. When clicking the camera icon, the app did not succeed to open the camera to take a picture. However when testing on another android version device, it worked perfectly fine. This problem may have to do with android versions and can easily be fixed with setting the right permissions and adding the updated version of dependencies in the build.gradle file. A problem faced relating to imports was when declaring the import kotlinx files in the MainActivity to be able to use certain functions, there were many problems as those files were not found. After doing research, the solution for this was having to add the id ‘kotlin-android-extensions’ to the plugins in the build.gradle file. Another example of this was when adding the import .esafirm files to the MainActivity to use functions, this import failed. Having searched the problem, the solution to it was that we had to call “ maven { url “https://jitpack.io”}” in the build.gradle(project) file. Another major issue faced was when

integrating the facial recognition model into the application. Initially the application was coded in Java and with research, a classifier would have been needed in order for the facial recognition model to work with the application which may have not been feasible with the time-frame. The solution to this was to switch from Java to Kotlin instead as Kotlin allowed the addition of the model to be coded within the MainActivity while working on the bitmap of the image which was complex but practical.

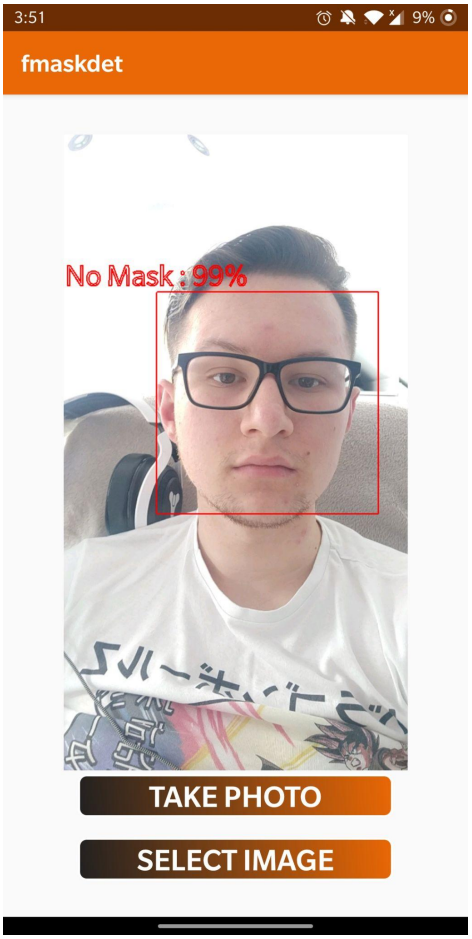
5.3 Backend Issues and Solutions

The largest issues faced by the backend of the system was multi-faceted and must be explained with great detail. Initially, the frontend was intended to operate through constant communication with a cloud computing service that contained all of the backend algorithms and processes necessary to perform facial recognition. The cloud computing service would conduct all of the operations necessary with its available resources and return the results back to the app. As we found out during the project, free and online cloud computing services do not provide its users with the amount of resources necessary to perform intensive machine learning algorithms and facial recognition on images. These machine learning algorithms are very computationally expensive and require a lot of resources that the free tiers of every single online cloud computing service provider do not offer. Each and every online cloud computing service offers different prices and resource limits depending on what payment plan is chosen. After going through each possible service provider and not finding any that were suitable for this task at a reasonable cost, we decided to perform these backend services locally through systems like Flask and Django. However, these web frameworks continuously proved to be incompatible with Android Studio's virtual machines and emulators, with well documented and numerous issues appearing online as well. We found a solution to all of these issues, in order to perform our facial recognition we needed to have the facial recognition system on the application itself. While we are aware that this would slow down our application, we felt this was the simplest and most appropriate option given the position we were in. We were also fortunate enough to have machines at home capable of performing machine learning to an adequate standard. However, if we wanted to use millions of images for the training or increase the amount of times we trained the facial recognition model, this would cause issues due to the slow performance speeds our home machines would offer.

6. Results and Testing

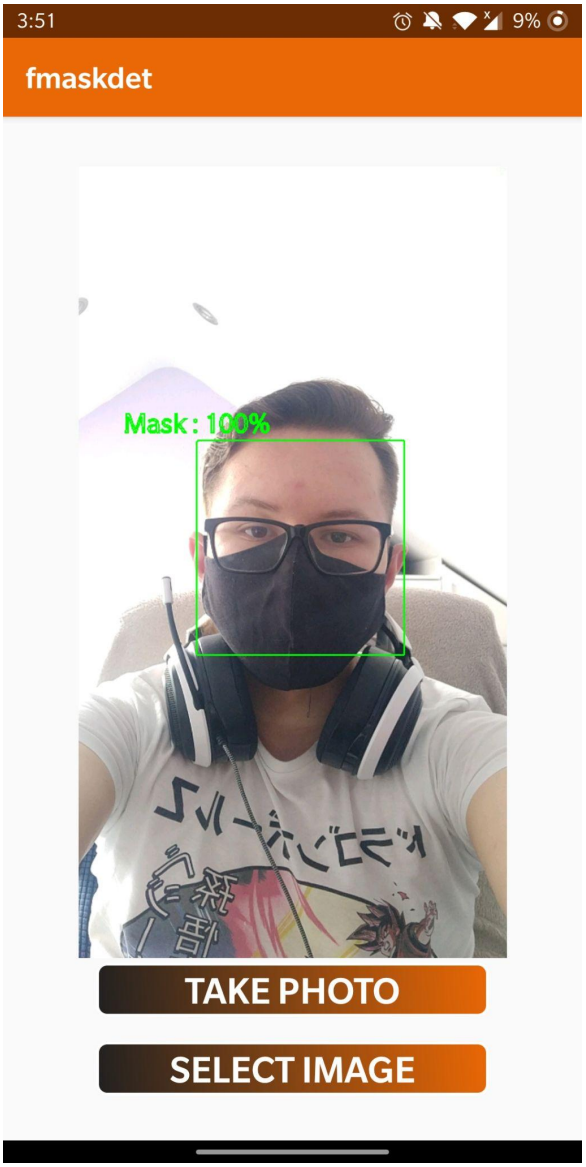
6.1 Facial Recognition Testing

Test #1



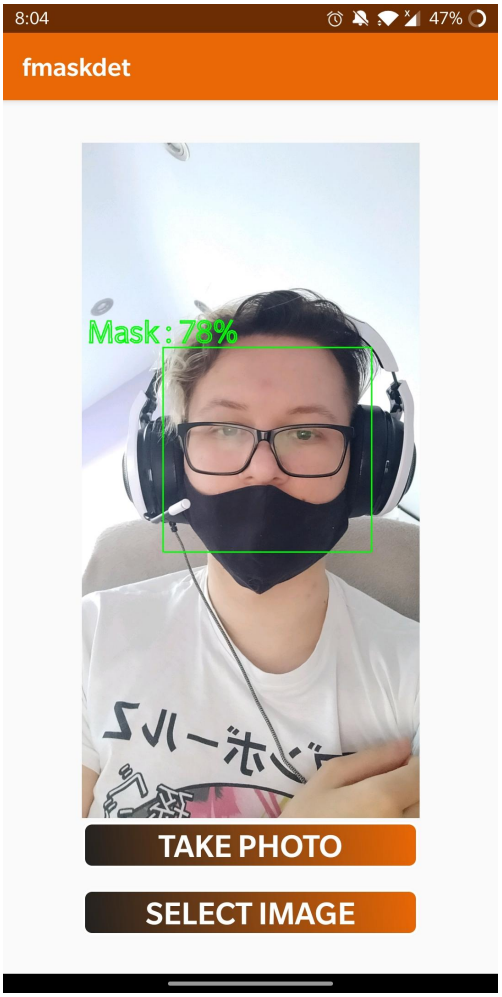
This is considered to be the optimal conditions for a picture to be taken by our application. A well-lit image with the camera positioned directly in front of the face allows for a clear view of the face. The facial recognition system does not have to handle any difficulties in relation to detecting the face and identifying any facial covering. The system was able to clearly put a box around one of the project member's face and perform the tasks of facial recognition and face mask detection. As we can see, the score returned was a 99% confidence level of no mask detected within the image. This is evident since there is no mask to be found in the image and as a result, our facial recognition does work at detecting a face with no mask in the most optimal environment conditions. The glasses may or may not have affected the confidence score so more testing will be required for that.

Test #2



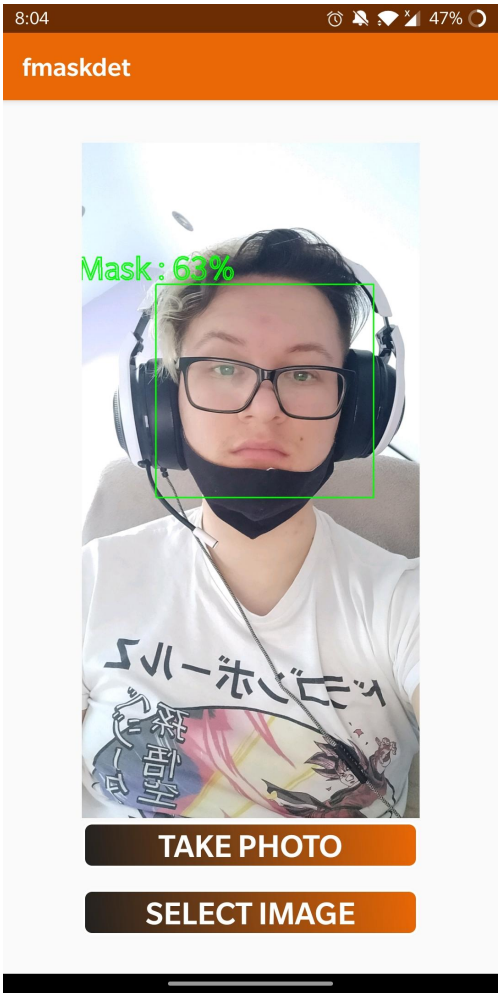
Here we have another image that contains the most optimal conditions to take a clear photo for the facial recognition system. Yet again we have a well-illuminated picture with the camera positioned straight towards the front of the face that allows for a clear shot of the project member's face. The significant difference this time is the face mask being worn correctly on the subject, resulting in the mouth, nose and jaw being completely covered. The facial recognition system clearly detects that the mouth, nose and jaw are being obstructed by the mask being worn over the subject's face and returns a 100% confidence level that the user is wearing a mask.

Test #3



We decided it would be important to test how well the system could detect the mask even if the mask was being worn incorrectly. In the image above, we chose to have optimal conditions for lighting and both position of camera and face angle for this test. As we can see in the image above, the facial recognition system was able to successfully find the mask located on the face of the subject. Despite the facial mask not covering the nose and potential obscuring of the face by the glasses, the system detected the face and the fact that the face was wearing a mask. It is important to take note of the score here that the system acquired. This 78% was not as high as the previous tests scores, indicating that the system is less confident about this image having a mask on the user's face compared to the previous images. Further tests should examine how much confidence the facial recognition system has towards detecting incorrectly worn masks.

Test #4

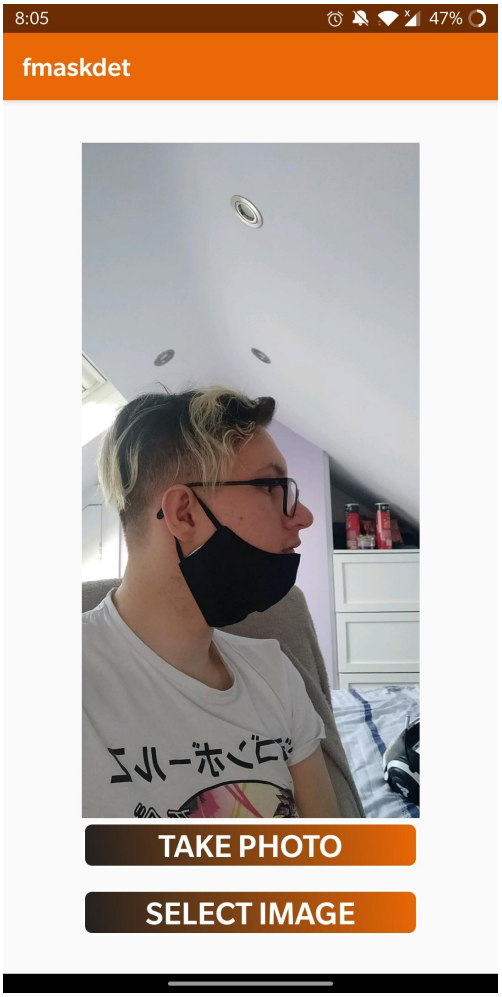


Here we have a mask being worn on the project member's face but the mask is not covering the mouth or nose, with other environmental conditions remaining identical. Here we can see that the confidence that the facial recognition system has towards the subject wearing a mask has lowered again compared to the previous test. However, the system still detected the face of the user and recognized that the user was still wearing the mask despite the poor position of the mask on the face. We can learn from this test that the facial detection system can produce high quality results given the most optimal conditions for when the picture is taken. Poor attempts at wearing the mask will affect the accuracy of the results though so this should be kept in mind.

Test #5

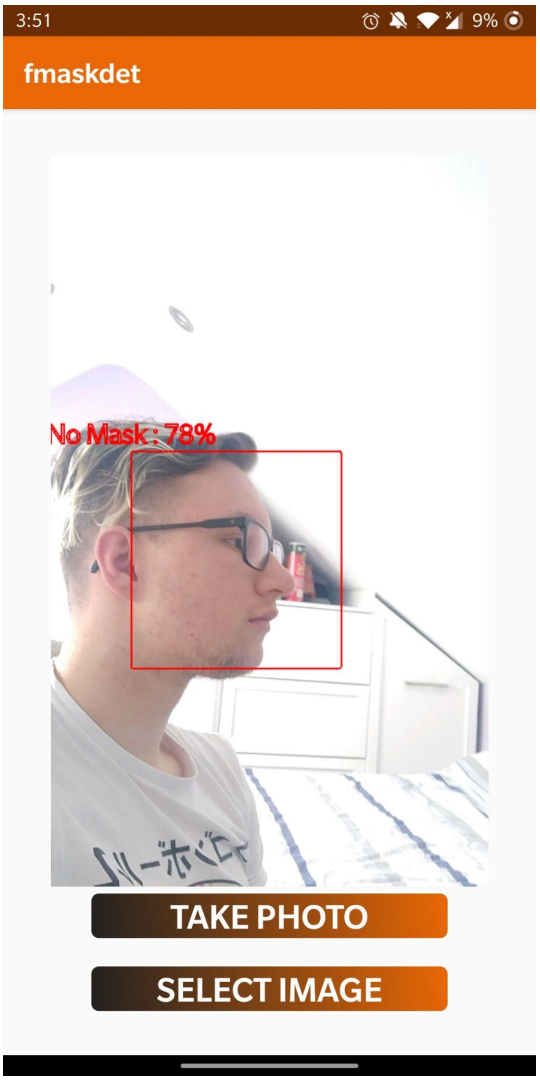
This image was taken in order to provide a large shift in environmental conditions with the goal of discovering what situations might the facial recognition system fail to detect a face. The strategy involved taking into account the conditions that lowered the facial recognition system's confidence with the previous tests kept in mind. We learned that wearing a mask incorrectly and changing the angle had certain effects on the accuracy of the results. The changes brought to the environment in this image include taking the picture from the right hand side of a person's face, lowering the angle of camera from the user's face and wearing a mask incorrectly by not having the nose covered. As we can see, the application failed to detect a face in this image due to the lack of a red or green box present within the image. This is a key step towards learning what conditions may cause the system to fail and we must test further in order to figure out why the system failed.

Test #6



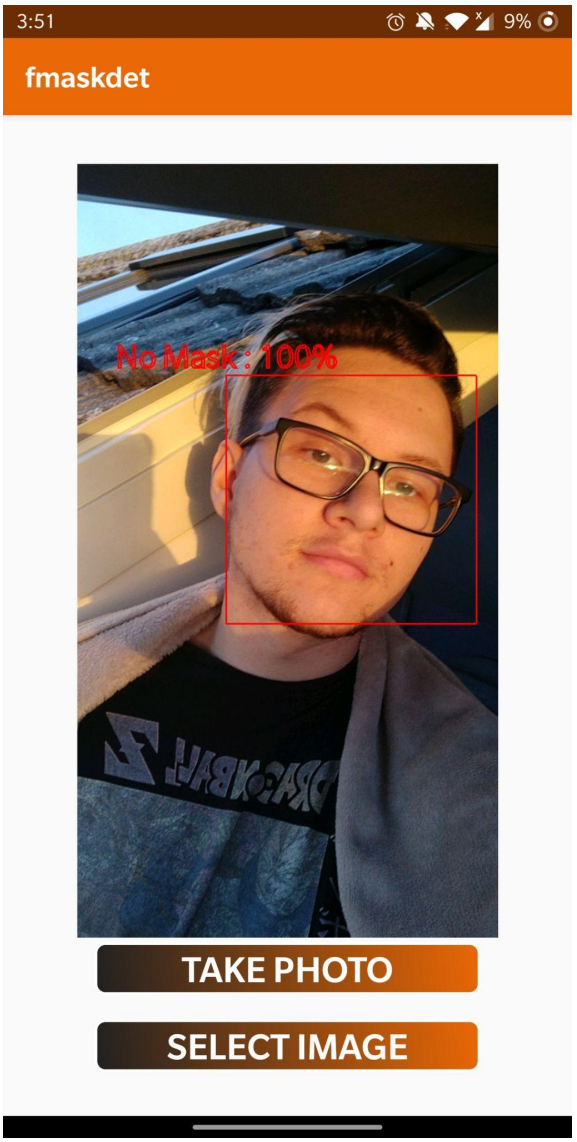
Here we decided to cover less of the face from the mask but still maintain identical environmental conditions to the previous test. This time the mouth and the nose was not covered by the mask, but the angle and lighting remained the same in order to figure out what conditions caused the facial recognition system to fail. Interestingly, the system malfunctioned again and was not able to detect any face even though one was present in this image. This could be as a result of the mask covering key facial features and obscuring points of the face that the system needs in order to detect a face within the image. The angle of the picture may also be having an effect on the system's ability to detect a face, since only half of the face is being shown in the image for both this test and the previous one as well. More testing needs to be performed in order to have an idea of why the system is erring in these scenarios.

Test #7



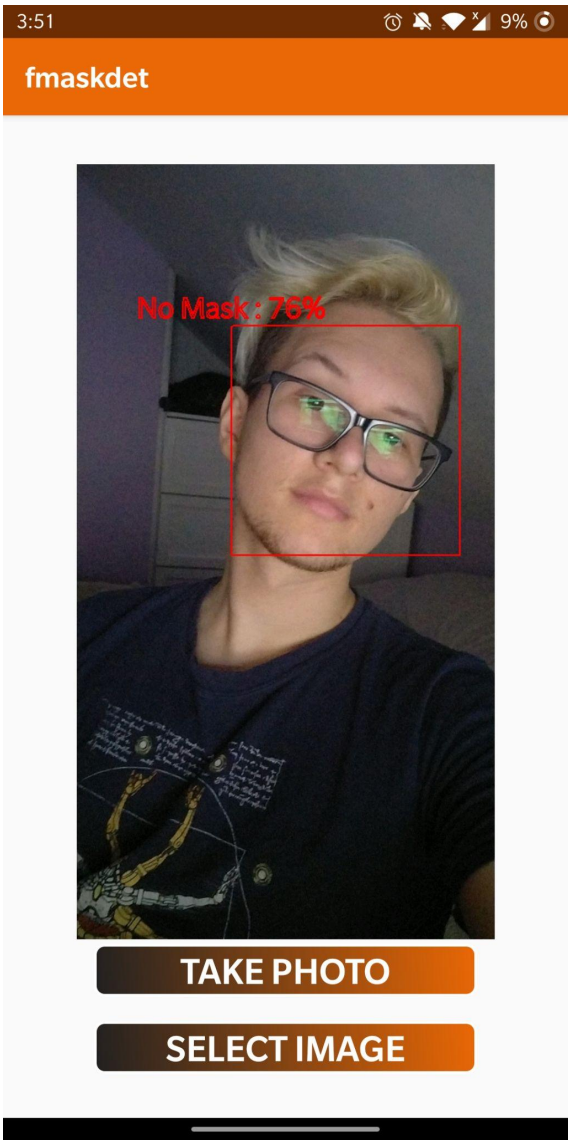
For this test, in order to figure out why the system failed to detect a face we decided to completely remove the mask but tried to feasibly maintain the same environmental conditions if possible. With the angle and lighting in this image being nearly identical to Test #5 and Test #6, the application successfully detected a face this time when a mask was not worn on the project member's face. With a confidence level of 78%, the system was able to detect the user's face with only the side profile of the person. We learned that the application can detect a user's face if only half of the subject's face is given but there are difficulties in that process. If the user is wearing some type of facial covering, this will obscure key information that allows for the detection of a face and as a result could cause the system to feel.

Test #8



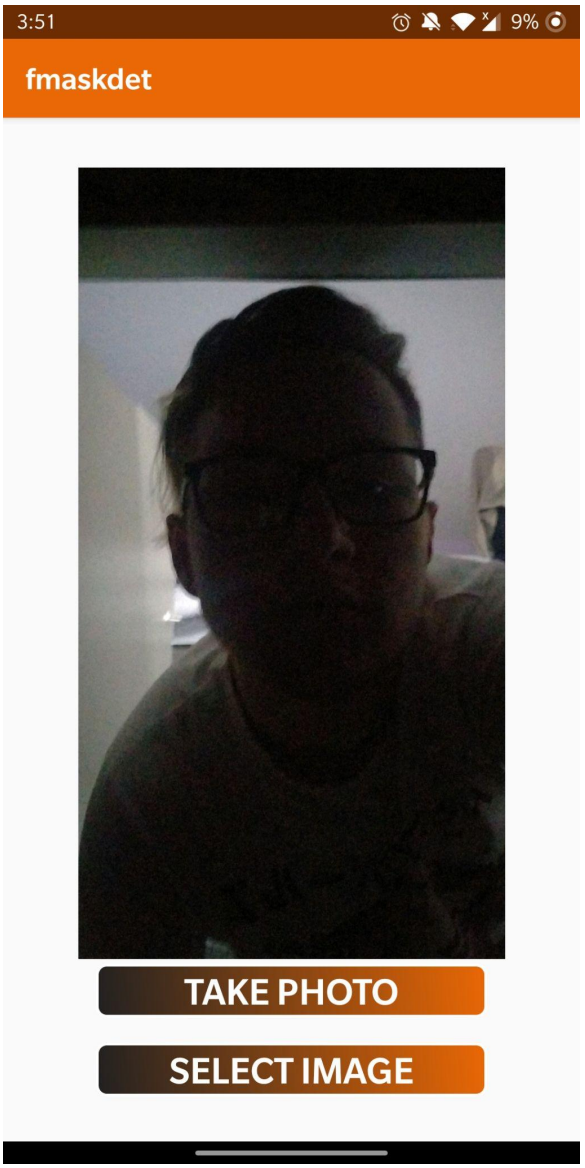
For a robust testing process, it is important to test many different conditions and environments for the system you are testing. We decided to change the lighting from a bright, well-lit room to a sunsetting environment while having the camera slightly angled away from the center of the face. Surprisingly, this gave us the highest confidence score out of all the other tests previously conducted. If we had to give a reason for this, it may be due to how the lighting of the face and angle of the picture created a clear image of the face, with no points or facial information obscured by any shadows whatsoever. Lighting should be investigated further, in order to see how differences in the lighting test the facial recognition system.

Test #9



In order to figure out how lighting affects the facial recognition system, we decided to maintain the same environmental conditions in terms of the angle of the face and position of the camera. The project member changed the lighting inside the room he was in to make the entire room darker but maintained a dim light from a lamp behind the camera but in front of him. As we can see above, the application was still able to detect a face that did not have a mask on in this image even with the darker lighting conditions. We must take note of the confidence score though. The facial recognition system outputted a confidence score of 76%, showing that the system was less confident about the label produced here compared to the previous tests with better lighting conditions. We need to conduct more tests in order to figure out how the lighting can cause the system to give an incorrect result or fail. This will allow us to learn more about how the lighting system affects the system.

Test #10

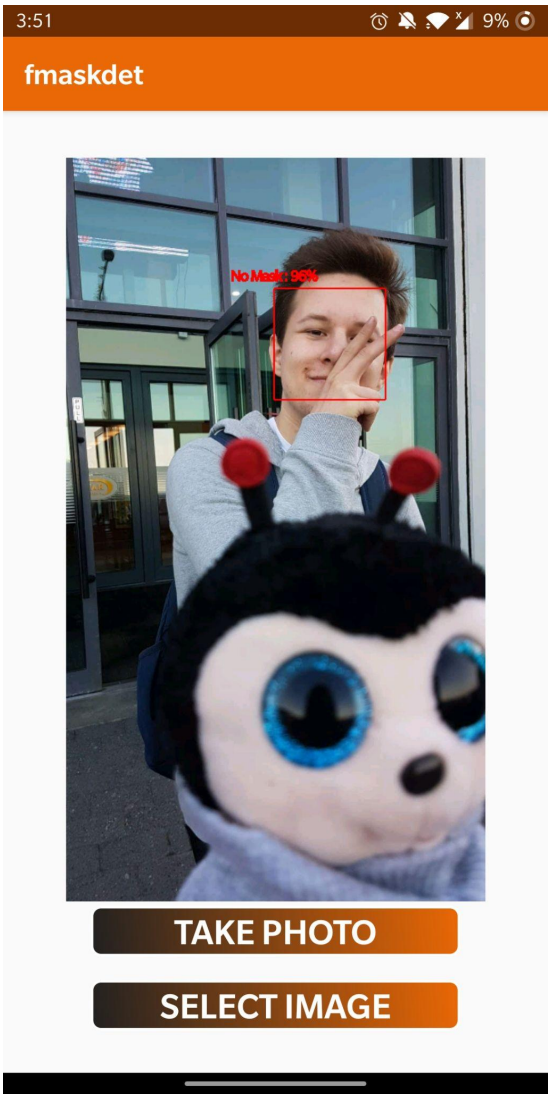


This test was conducted with the goal in mind of seeing what lighting conditions could cause the application's face recognition system to fail. We decided to perform this test under the one of the darkest lighting conditions possible while trying to keep the subject's face visible within the image. The system completed failed at detecting the face that can be found within the above image. It is clear to see that the system failed to find any key facial features or any important information concerning the face of the project member within this image. As a result, we can safely conclude that the lighting of an image does affect the system's ability to perform and well-lit lighting conditions are recommended when using this application.

Test #11

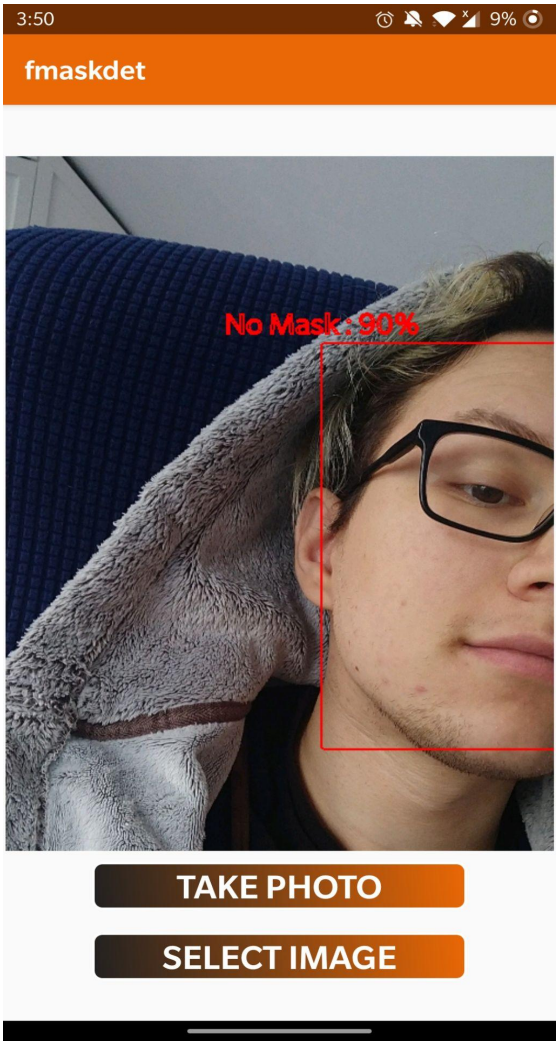
While conducting our tests, we noticed most of our images had the subject a short distance away from the camera. We wanted to see if distance from the camera could affect the accuracy of the results. As well as that, we wanted to see if shadows on the person's face could affect the system's ability to detect a mask or not. As we can see in the image above, the facial detection system has no difficulty detecting a face at a medium distance in this scenario despite shadows covering the face. The shadows here may not have as much of an impact on the points the system needs to detect facial features. It seems none of the pigeons caused any difficulties for the facial recognition systems, so animal faces are not likely to be detected.

Test #12



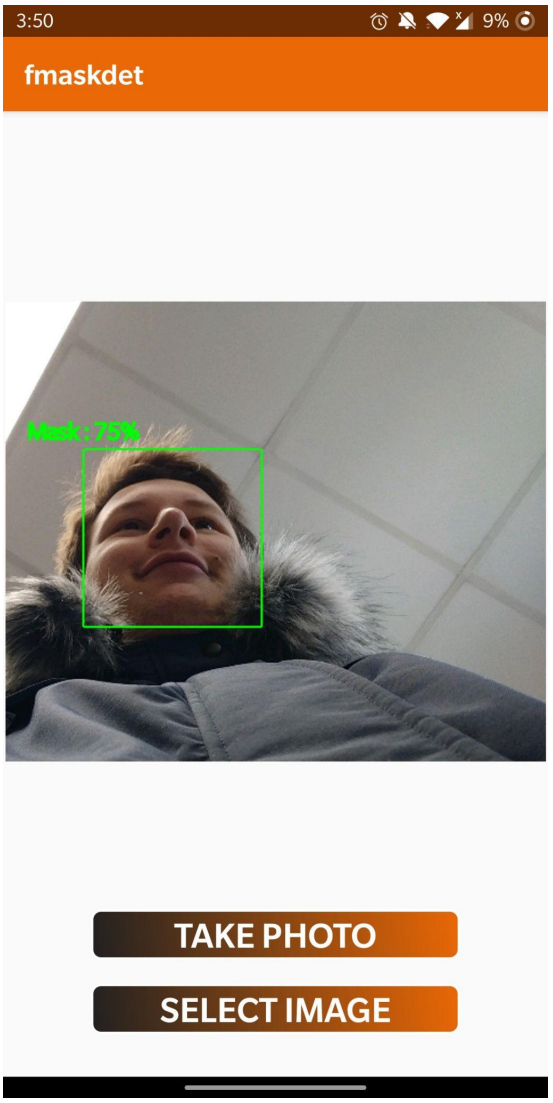
Here we have another image that has the face at a medium distance away from the camera, but now we have the project member's hand covering some of his facial features. This test was designed to see if the image partially covering the subject's face with a hand at a medium distance would cause any issues for the facial recognition system. Other key environmental conditions that are significant to mention include the well-lit nature of the image and the very large ladybug teddy bear face which can be located a very short distance away from the camera. As we can see in the picture above, the facial recognition system was able to successfully find the human face in the image given these conditions and successfully labeled the face as having no mask. With a confidence level of 90%, that is a relatively high score considering all the variables within the picture.

Test #13



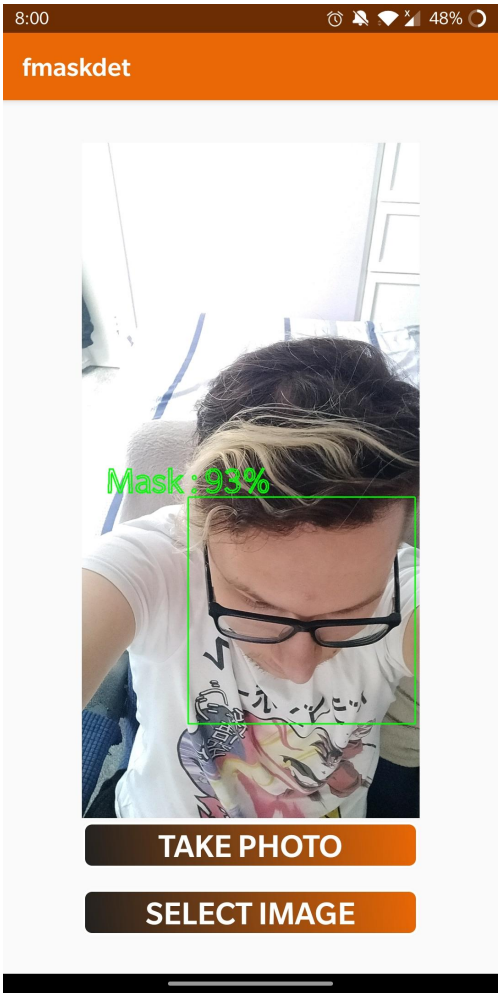
The strategy for this test was to see if an image with only half of the face directly in front of the camera could cause problems to occur for the facial recognition system. We decided to have the room well-lit since we wanted to investigate how distance and angles could affect the system's ability to perform properly. As we can see in the image above, despite only half of the face being visible in the image, the facial detection system still picked up on the face within the image. With a confidence level of 90%, the system labelled this picture with the no mask classification and put the red box correctly around the location. This shows that even at some strange angles or some missing facial features, the facial detection system can still perform its abilities accurately.

Test #14



We decided to test more strange angles and this image involves the camera being directly below the project member's face. With the camera and face at the position and angle shown in the image, certain points of the project member's face have been obscured and as a result, the app classified the image incorrectly. The user is not wearing a mask in this image and the application has labelled this face as wearing a mask. The shadows casted by the lighting and the facial features may have played a small role here, but we believe that the incorrect label was due to the angle of the face and position of the camera.We need to do more testing to figure out how this happened.

Test #15



For the conditions of this test, we changed certain environmental factors like the position of the camera as well as the angle of the face. Tilting the face downwards and placing the camera slightly above the head clearly impacted the facial recognition system's ability to perform again as we can see above. The app has successfully detected a face and yet, there is a surprising result from this test. The system has classified this image with the mask label and stated that a mask has been worn by the subject despite that not being the case for this test. We can certainly argue that the same incorrect labels have been placed as a result of the very obtuse angles of the face and positions of the cameras that both test's conditions had. This must be improved upon in future work to remove issues like this.

6.2 User Case Testing

The following are the observation notes made by the project members as well as the answers to the questionnaire that the 4 anonymous participants gave. The observers did not speak or interact with the participant during the time they had used the application. Answers from the participants who filled out the questionnaire in the link below:

https://docs.google.com/forms/d/1OwF_FQ3_Or74rEVNssSEwVY_UQI7cWSCeTI6D0v194s/edit#responses

Participant #1

The participant slowly sat down and carefully opened the app on the mobile device. After entering the application, they tapped the take photo button. There was a pause as they searched for the camera button in the top right corner. After a couple attempts of tapping that button, they opened the camera and took a picture of themselves. After viewing the result of the face mask detection, they took another photo using the in app camera. After witnessing the second result, they attempted to go back and see the previous result by constantly tapping the back button. Afterwards, they tapped the select image button and uploaded a photo from their gallery. After seeing the third result, they tried to go back again by tapping the back button. Afterwards, when they could not go back to previous results they closed the phone.

Key observations

- The participant might have pre-existing assumptions about features that are not present in the app.
- The participant changed their mind and wanted to go back to past results numerous times
- The button sizes might not suit some people compared to than others
- The application can perform quickly if it is the sole app open at that moment in time

Questionnaire Answers

Q1: Was taking an image of your face with the assistance of the app easy or hard?
A1: It was very difficult

Q2: (to the previous question, regardless of answer): Why was it easy or hard?
A2: I do not take selfies often so it was hard to get a good picture

Q3: Was navigating around the menus and pages of the application easy or hard?
A3: I think it was okay

Q4: (to the previous question, regardless of answer): What made it easy or hard for you?
A4: The menus are nice but if I press something it's hard to go back

Q5: Did you like the results that were displayed by the app?
A5: Yes

Q6: (to the previous question, regardless of answer): Why did you like or not like it?
A6: I got the results that I wanted

Q7: Was the app's performance fast or slow for you?
A7: I thought it was fast

Q8: If you could add an additional feature, what would it be?
A8: Let me go back if I press something

Evaluation #1

- Some people may assume there are functions within the application that are not present, like the ability to view multiple results. This is a feature that could be added in a future iteration.
- Button sizes may suit or may not suit people depending on their finger size. Having the option to change in-app button sizes would increase the accessibility and functionality of the application.
- Quicker performance than expected for the facial recognition system. This may be dependent on image sizes, image quality and resources available for the device.
- Simple menu structure adds extra navigatibility and increases the quality of user experience.

Participant #2

The participant opened the app and pressed the take photo button. They successfully took a photo and viewed the result. They repeated this two more times before tapping the select image button. Here they uploaded other photos of themselves and viewed each result as well, some with masks on and with masks off, with the application working successfully each time. Finally, they uploaded a landscape image of some people at a lake. After a bit of waiting, the application returned the result the participant desired and closed the app.

Key observations

- Many uploads of different pictures can accumulate to a lot time spent navigating the menus
- Facial recognition system has varying speeds between one photo to the next, possibly to size of the photo
- Slowdown can occur if a very large image is used
- Experience with mobile devices helps with using the app

Questionnaire Answers

Q1: Was taking an image of your face with the assistance of the app easy or hard?
A1: It was easy

Q2: (to the previous question, regardless of answer): Why was it easy or hard?
A2: I have lots of experience taking selfies

Q3: Was navigating around the menus and pages of the application easy or hard?
A3: I think it's easy to navigate

Q4: (to the previous question, regardless of answer): What made it easy or hard for you?
A4: The design of the menus are pretty simple

Q5: Did you like the results that were displayed by the app?
A5: Yes

Q6: (to the previous question, regardless of answer): Why did you like or not like it?
A6: It was easy to understand how I got the results

Q7: Was the app's performance fast or slow for you?
A7: I thought it was kind of fast

Q8: If you could add an additional feature, what would it be?
A8: I want to upload many images instead of just one at a time

Evaluation #2

- The ability to upload many images at once can save a lot of time and effort needed from the user.
- Can be repetitive uploading multiple images one at a time
- Large images can cause the application to slow down, dampening the user experience.

Participant #3

The participant opened the app and hit the take photo button. After taking the photo from a low angle, he viewed the result which had no box whatsoever. The participant took another picture but this time the picture was straight in front of their face and the app successfully worked, showing the person positive results. The participant proceeded to take another picture but this the camera was far above their face. The application failed to detect their face and this was seen in the results when the app displayed them. The participant then tapped the select photo button and uploaded an image of a forest. The application took some time before returning a result where no faces were detected. Afterwards, the participant closed the application.

Key observations

- Taking pictures at angles that are not straight at your face causes issues for the facial recognition system
- Participants may not understand why the application may not work sometimes.
- User interface may be bland to some people
- The facial recognition system can take some time to return a result where a face has not been recognized

Questionnaire Answers

Q1: Was taking an image of your face with the assistance of the app easy or hard?
A1: The face detection was kind of finicky

Q2: (to the previous question, regardless of answer): Why was it easy or hard?
A2: When I tried taking pictures from different angles, it would not work.

Q3: Was navigating around the menus and pages of the application easy or hard?
A3: It was fine

Q4: (to the previous question, regardless of answer): What made it easy or hard for you?
A4: The menus were simple, a bit boring

Q5: Did you like the results that were displayed by the app?
A5: No

Q6: (to the previous question, regardless of answer): Why did you like or not like it?
A6: I was taking pictures of my face but it was not working

Q7: Was the app's performance fast or slow for you?
A7: I thought it was slow, the loading took too long

Q8: If you could add an additional feature, what would it be?
A8: I would focus on making the app work first

Evaluation #3

- Some users may find the user interface bland and too simple.
- Users who regularly take pictures at different angles may encounter many issues with the facial recognition system.
- Constantly running into issues during their use of the application will negatively impact the user experience.

Participant #4

The participant opened the application and selected the take photo button. After taking a photo and viewing the result, they saw that the picture came back with no result. They retook the photo by tapping the take photo button again, but this time they turned on the flash on the camera. After a quick flash, the user took a picture of themselves and after viewing the results, the user saw a box around their face. They tapped the select photo button and uploaded an image from their gallery. After viewing the picture and the results they acquired, the participant closed the phone.

Key observations

- Lighting affects the quality of the images and the results received
- Colour choices can affect the user experience of the app to a great extent.
- Inconsistency of facial recognition system can affect the user experience of the app
- Clothing like scarves or burkas can affect the facial recognition system's ability to detect a face, not just masks.

Questionnaire Answers

Q1: Was taking an image of your face with the assistance of the app easy or hard?
A1: The face detection was okay

Q2: (to the previous question, regardless of answer): Why was it easy or hard?
A2: The room did not have good lighting

Q3: Was navigating around the menus and pages of the application easy or hard?
A3: It was easy

Q4: (to the previous question, regardless of answer): What made it easy or hard for you?
A4: The menu's layout was straightforward and easy to use for me. Nice colours as well.

Q5: Did you like the results that were displayed by the app?
A5: Yes

Q6: (to the previous question, regardless of answer): Why did you like or not like it?
A6: Sometimes it worked and sometimes it did not

Q7: Was the app's performance fast or slow for you?
A7: Pretty fast

Q8: If you could add an additional feature, what would it be?
A8: Let me keep the flash on the camera when taking the photo. A dark mode would be nice

Evaluation #4

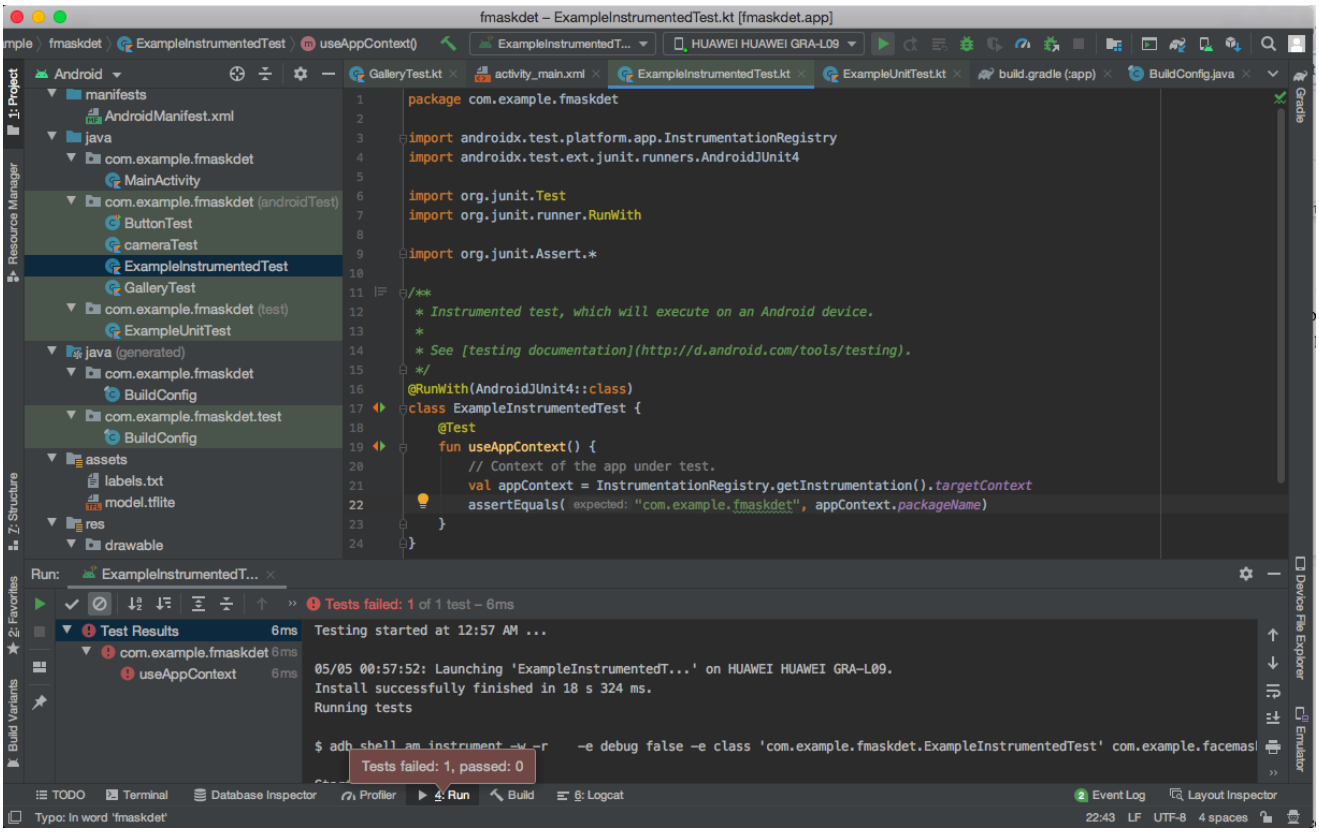
- Lighting of the room can affect the quality of the application's performance which affects the quality of the user experience.
- Dark mode for the user interface would increase the accessibility of the app.
- Possibly could implement a feature to automatically turn on the flashlight when the in-app camera is on.

6.3 Integration / Unit Testing

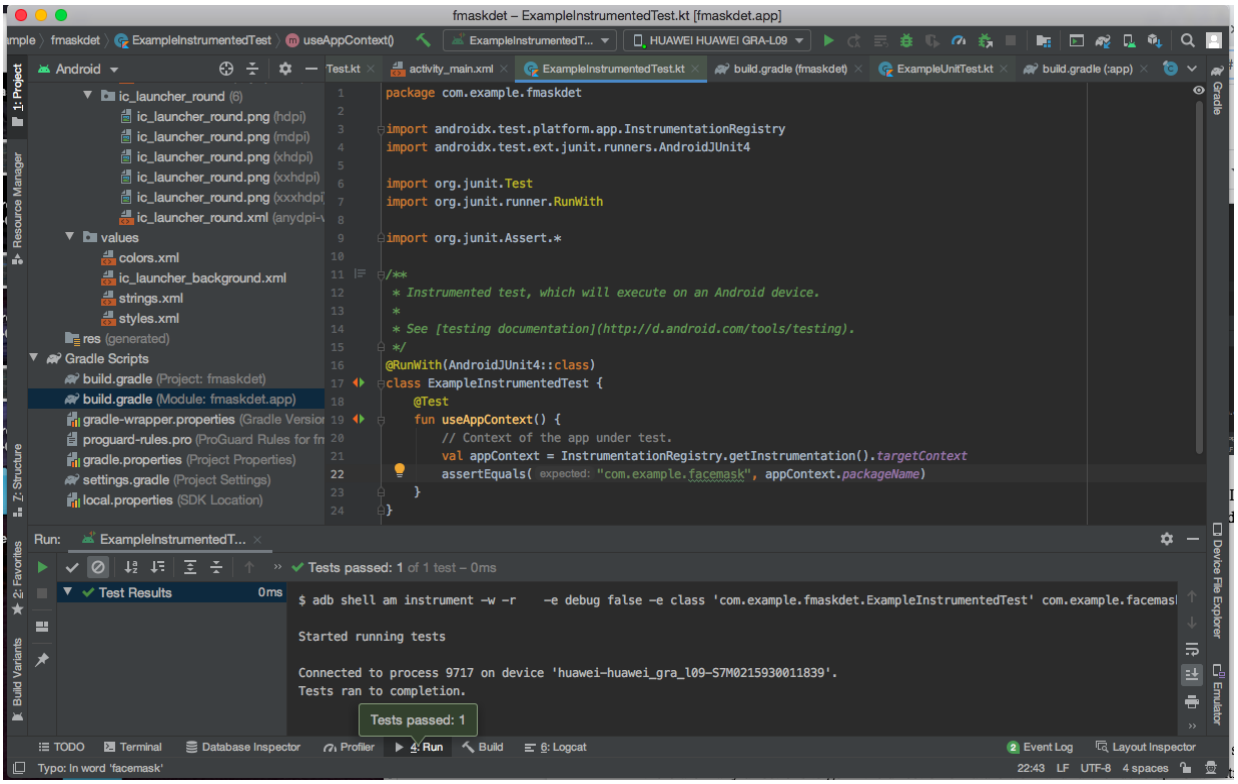
To test the application on android studio - Kotlin version, there were many different types of testing strategies used to test different parts of the application. Areas such as the UI design (visual representations) of the application were tested, the right package of the overall application and constant log captures. Since there were no actual calculations tested in this app and more so the interaction between the user and the UI of the app, the use of Espresso framework was thoroughly used.

ExampleInstrumentedTest

The ExampleInstrumentTest is an automated test kotlin file generated when the application is built. Starting off by getting used to how test files work and how they pass or fail is very important so the first test was using assertEquals() whether the applications package name is equal to the expected output or not.



The test case above fails because although the package name is ‘com.example.fmaskdet’ as seen in line 1 and we would expect that to be equal to as the .packageName, the applicationId in the build.gradle file is equal to ‘com.example.facemask’ so we would have to change the expected output and put in the right .packageName for the test to pass as shown below.

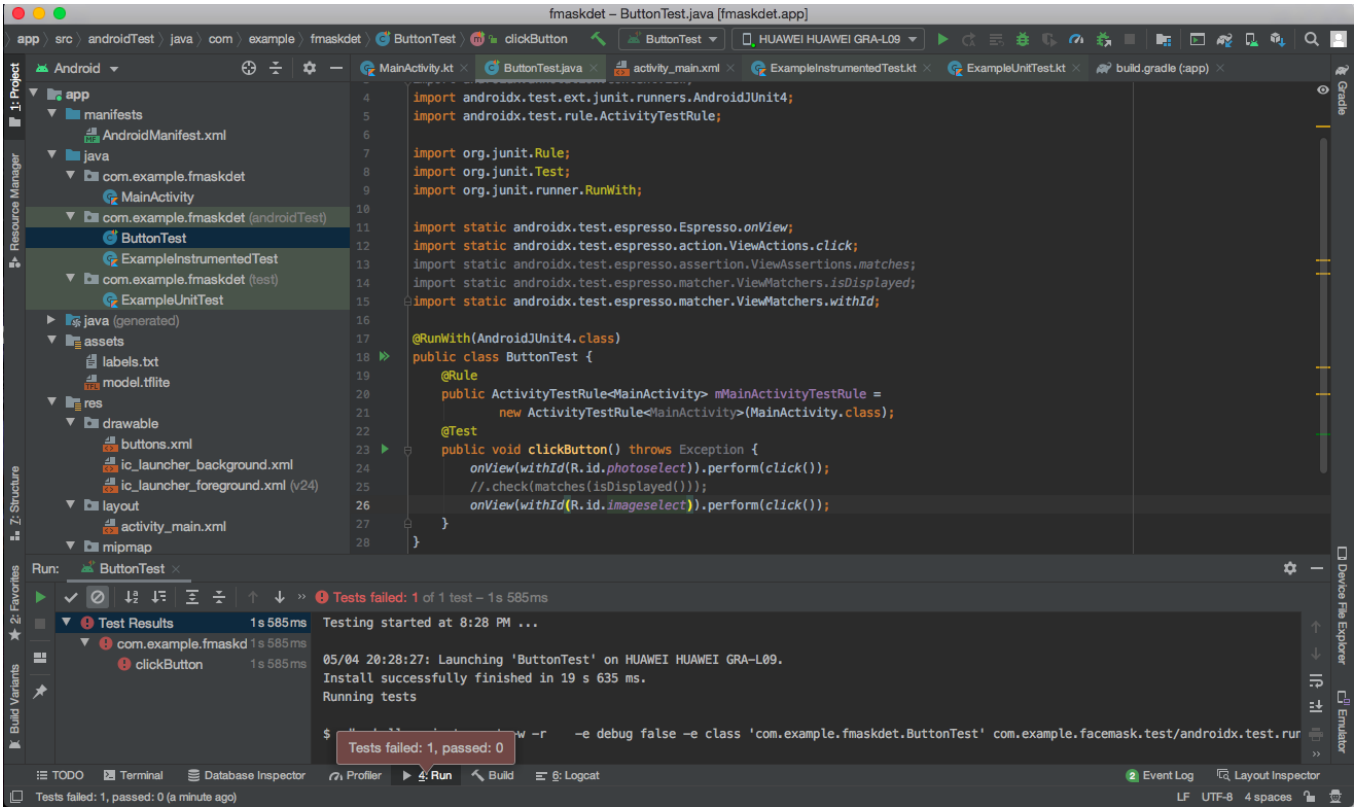


Espresso AndroidTest

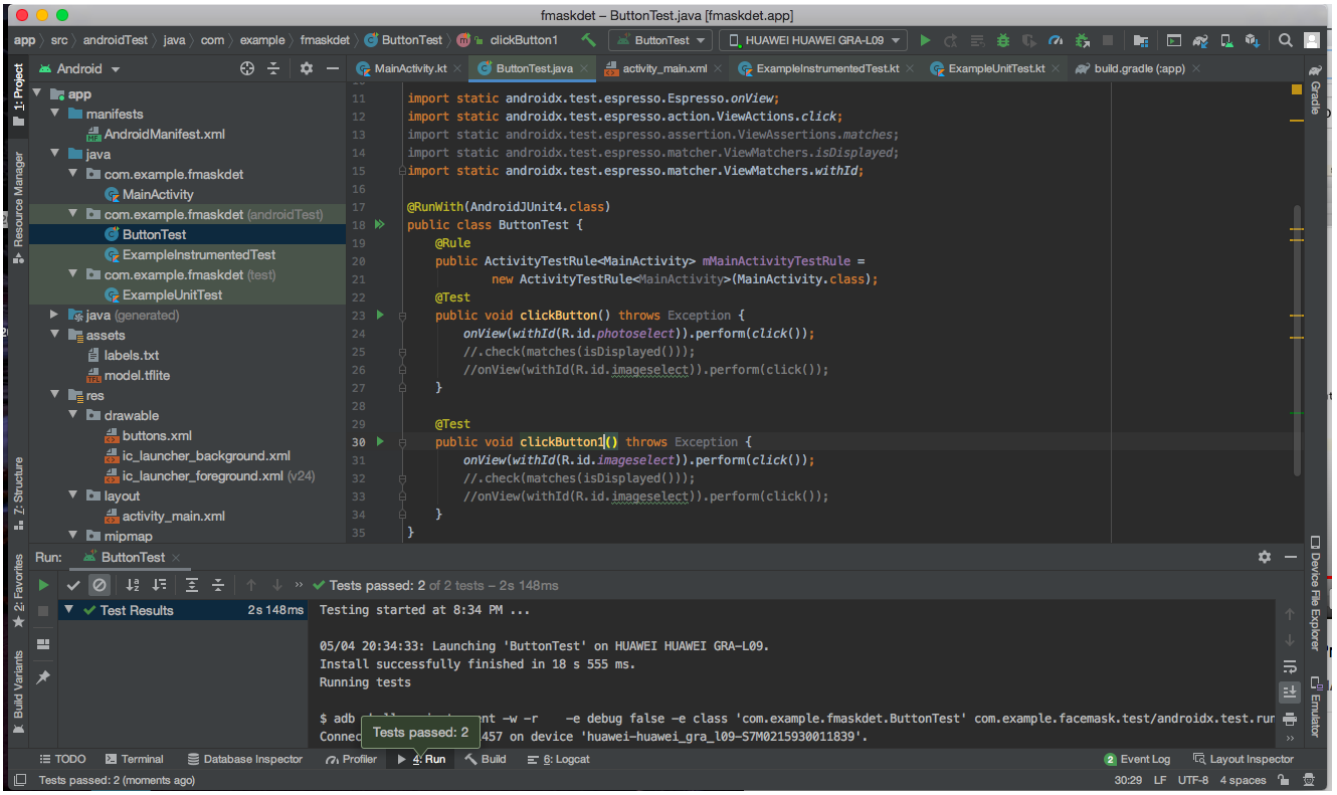
Espresso is a framework that allows developers to write UI tests that are concise, reliable and using an API. Espresso also takes care of any synchronization with any UI events. This means we don't really have to worry about any implementation details. The concept of Espresso is similar to real life scenarios in terms of the anatomy of a UI test. To use Espresso, the Android Test Runner and Espresso dependencies were configured in the build.gradle file.

ButtonTest.java

This Unit Test is a java extension file created using Espresso to test the User Interface of the application. An ActivityTestRule was added to tell the AndroidJUnit 4 runner to launch the MainActivity before any test and to turn it down once finished. The clickButton() method was created and the onView() is also a method on the Espresso class. The Id of the floating action button was placed within the onView() and the click() is a user interaction that will be stimulated by Espresso. In this case the button is only clicked and that is what we want to check. Both Ids of the two buttons were added in the clickButton() method but this seemed to have failed.

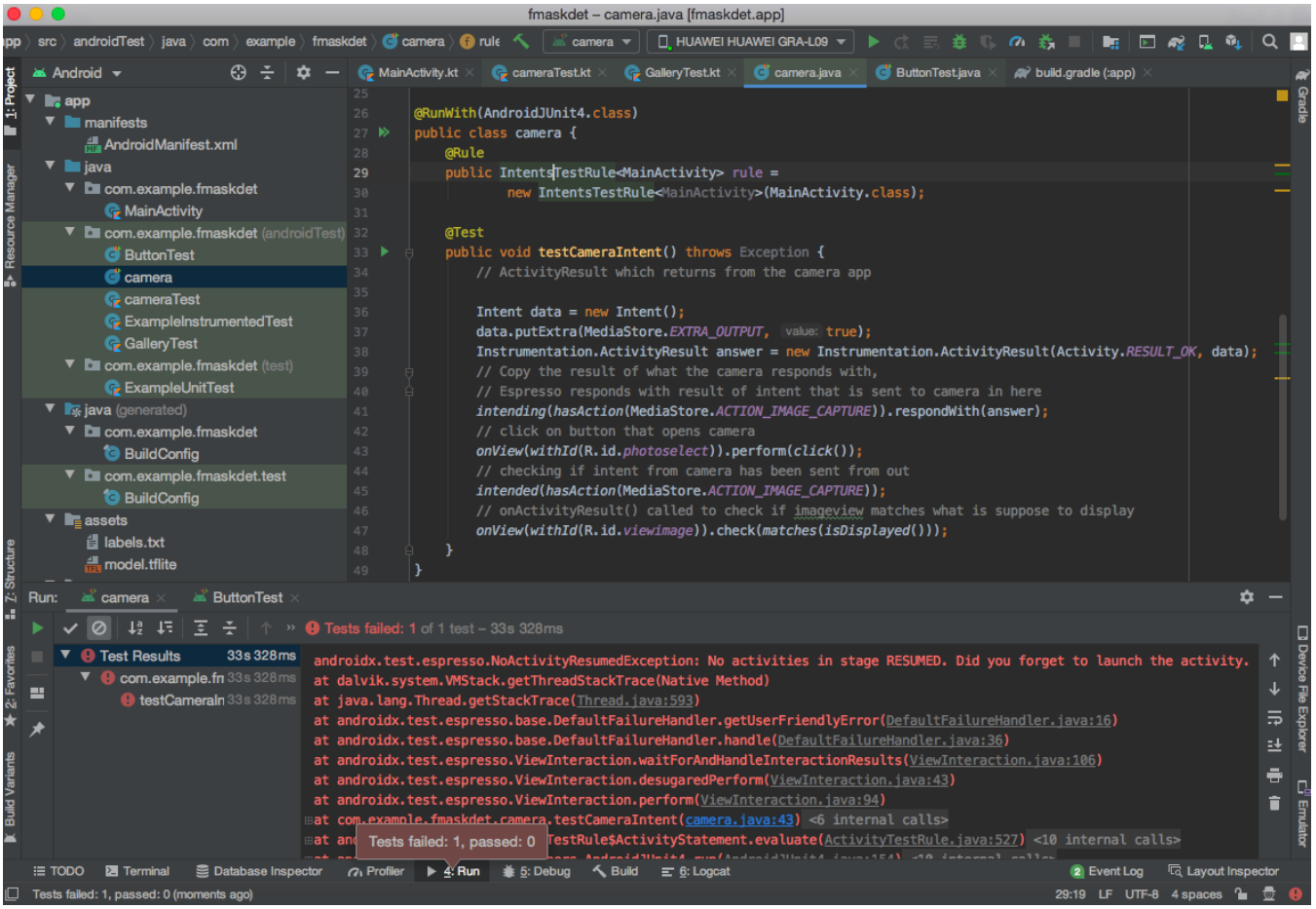


After a few changes around, it was found that both buttons needed separate methods to check for their .perform(click()). Two methods called clickButton() and clickButton1() were created with the separate Id of each button in the separate method and these passed the test.



Camera.java

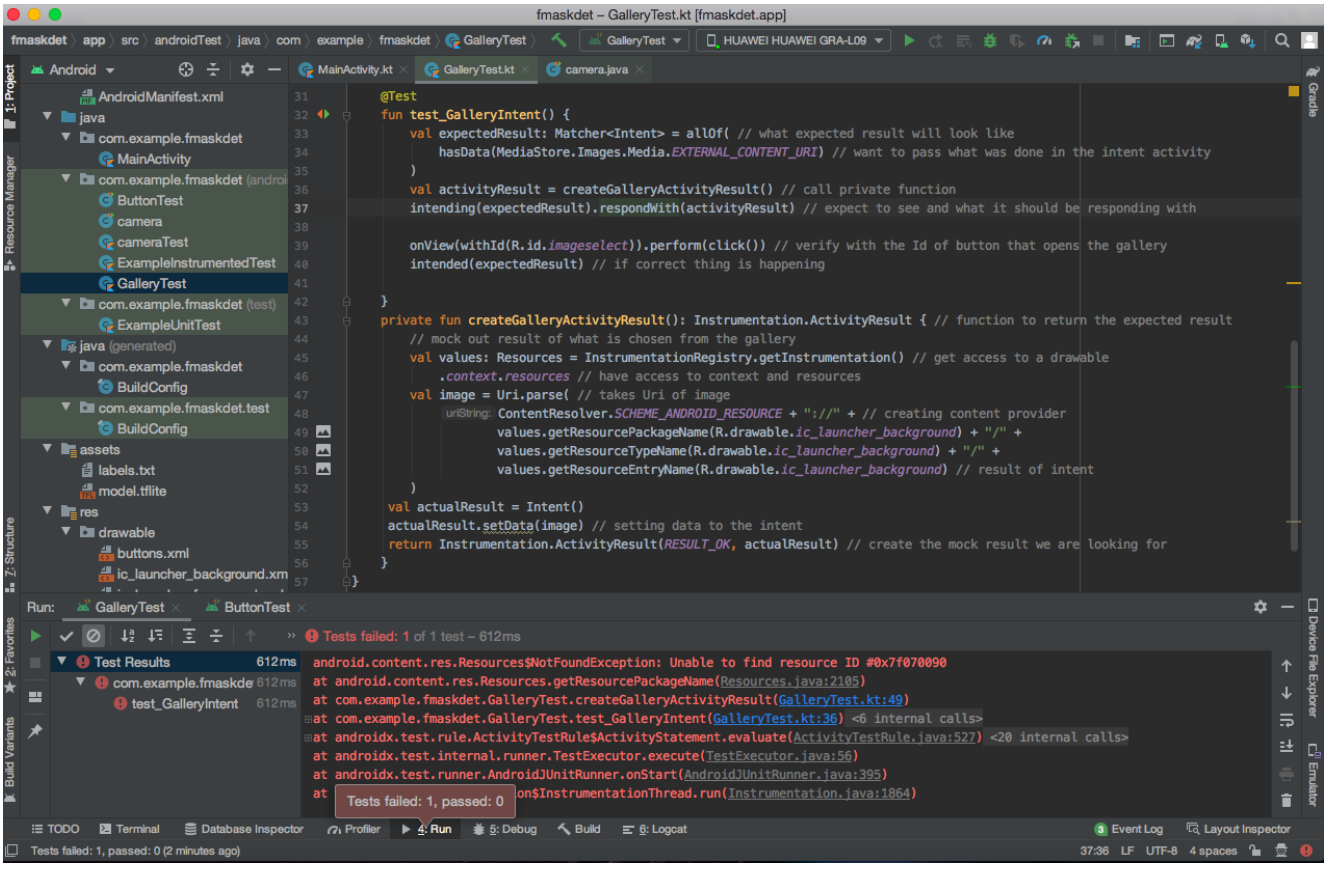
In this test file, the intent of the camera was tested whether the camera displays what is supposed to be displayed in the view(imageView). The first step was to set up an Intent rule to declare this is an intent test file. After creating the public method, the ActivityResult is called which returns from the camera app. Next, the result of the camera is copied and Espresso responds with the result of the intent which is labelled as the variable 'answer'. The button then has to be clicked which opens the camera and then 'intended()' checks if the intent from the camera has been sent from out. Lastly the .viewimage Id is checked whether it matches what is suppose to be displayed in the ImageView.



This test seems to fail, the first failed feedback tells whether 'Did you forget to launch the activity' to which if the Rule in the public class is changed from IntentTestRule<> to ActivityTestRule<> it fixes the problem because the Activity is then being launched. With this, there are still failed feedbacks after fixing that one problem. This test fails because of the intent call of the camera and is not quite correct.

GalleryTest.kt

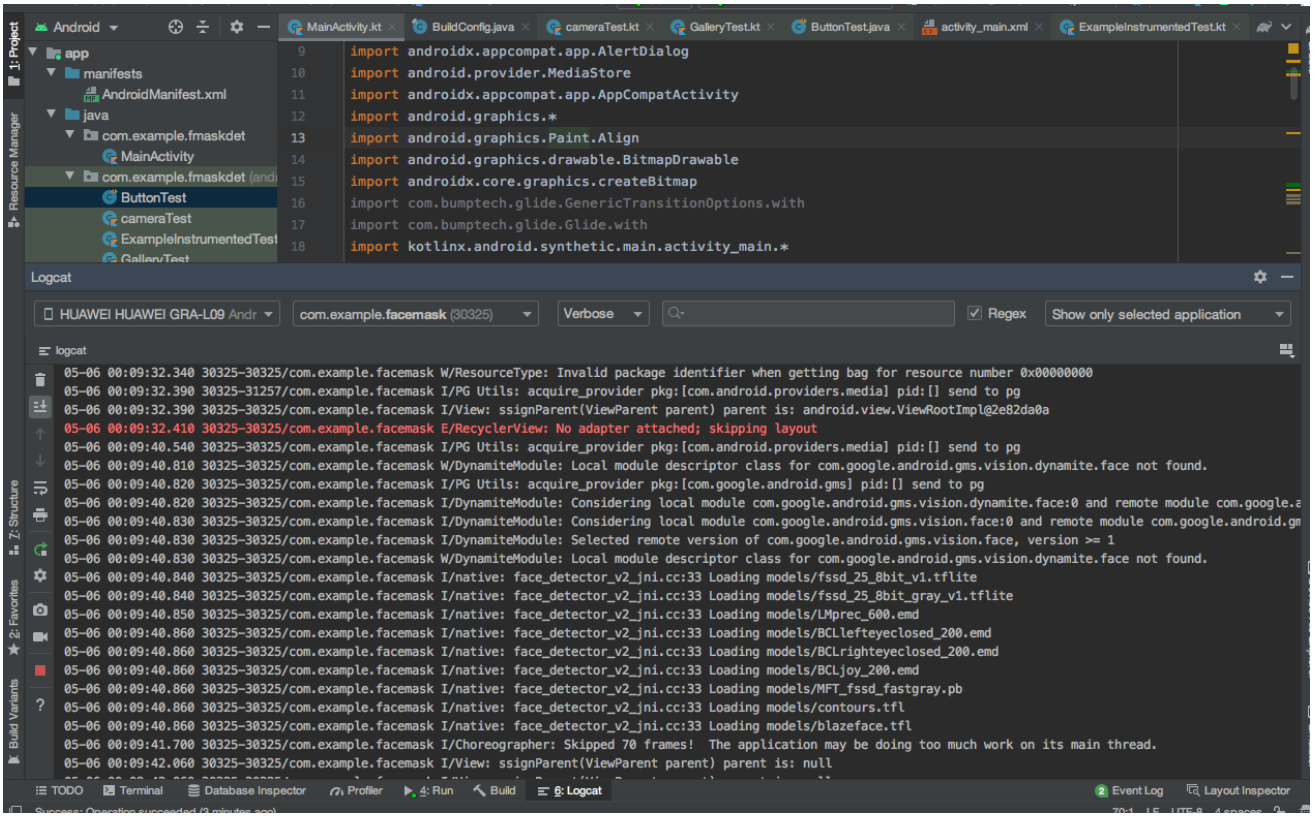
The gallery test file was written in kotlin version rather than java. This class also used the rule IntentsTestRule because it is the intent that is being tested. A method called test_GalleryIntent() was created to label a value for what the expected result will look like. It passes whatever is done from the MainActivity. A function was then created called createGalleryActivityResult() to return the expected result. It mocks out the result of what is chosen from the gallery, gets access to a drawable, context and resources. The Uri of the image is taken rather than the bitmap and then the content provider is created. The data is set to the intent and a mock result is created for what is looked for. The function is then called in the method and intending() is called to see the expected result and if that responds with the actual result. The button click() is then called which is used to launch the gallery intent and then intended() is called to see if the correct thing is happening.



This test seemed to fail because in actuality, we have created a bitmap of the image chosen from the gallery in the MainActivity and did not want to receive the Uri of the image. The feedback given shows resource ID is unable to be found.

Android Studio Logs

Android studio Logcat was used to continuously track the progress of the application and all the interactions that were taking place. While the application is running, it displays each interaction with the user interface, what is clicked, what is accessed and what is displayed. This helps to display any errors that could be encountered such as a programming error. It allows for debugging the android application more efficiently. Also, if there is any addition to dependencies in the build gradle file, there is a constant notification to sync the project immediately which helps keep the project up to date.



This above image is an example of the Logcat for when the app is running and a user interacts with the application. The line in red is just relating to RecyclerView which is part of Espresso. It does not have any effect on the running of the application. This Logcat the interaction between the application for when a user clicks the button ‘Select image’ and once having the image selected and then uploaded, the tflite model performing the analysis to return the analysed image.

7. Future Work

7.1 Future Possibilities

This application has high levels of scalability and a high capacity for added functionality, so there are many directions to go in the direction of future work and further development. Extra functionality that could be added involves allowing the user to upload multiple images at the same time, upload a dataset or upload their own facial recognition model as well. The architecture of the application itself could also be modified, with the addition of online cloud computing servers or locally operational servers to run the facial recognition algorithms instead of the app performing this system. As documented above, this app was developed and tested on Android Studio and runs on Android machines exclusively. The possibility of creating an iOS version and a Windows version of this application exists as well.

Moving beyond the scope of future work possible for the application itself, the skills acquired and lessons learned from working on this application could be used in future projects to come. For example, future facial recognition systems could be created that allow the user to recognize faces even if a mask or facial covering was being worn. This could be useful in security systems intended for the police in scenarios of tracking criminals or finding missing persons. Another potential project could involve creating a system that utilizes 3D facial recognition technology to detect faces or create 3D models of those faces. This highly advanced process offers the possibility of higher degrees of face detection accuracy compared to 2D facial recognition due to the 3D shape and structure of facial features. This extra dimension avoids issues that 2D facial recognition algorithms face such as changes in the lighting, different head angles and alternative facial expressions.

Overall, we feel that there are many potential directions in which future work can go in relation to this project. We learned a lot of new information and new skills were acquired during the creation, modification and evaluation of the application and these will all be useful in the future to come.