

---

Title of Project

Unity Shape Detection Package

---

Name of Student

Seán Fahey

---

Student Number

K00257361

---

Date

02-04-2024

---


Word Count

10,400

---

I declare that no element of this assignment has been plagiarised:

Student Signature



---

---

## Table of Contents

<b>1</b>	<b>INTRODUCTION, PROBLEM DEFINITION AND BACKGROUND .....</b>	<b>3</b>
1.1	PROBLEM DEFINITION AND BACKGROUND.....	3
<b>2</b>	<b>LITERATURE REVIEW AND RESEARCH .....</b>	<b>7</b>
2.1	PREVIOUS USAGES OF MOBILE GAMES IN SOCIETY .....	7
2.2	EVALUATION OF MOBILE GAMES – THE BENEFITS AND DISADVANTAGES .....	10
2.3	GOOGLE’S <i>QUICK, DRAW!</i> .....	11
2.4	SHAPE & DOODLE RECOGNITION .....	12
<b>3</b>	<b>SYSTEM DESIGN AND CONFIGURATION .....</b>	<b>16</b>
3.1	SPECIFICATION .....	16
3.2	USER STORY MAPPING .....	16
3.3	SEQUENCE DIAGRAM .....	18
3.4	OBJECT DETECTION MODEL LAYERS .....	19
<b>4</b>	<b>IMPLEMENTATION &amp; TESTING.....</b>	<b>21</b>
4.1	INPUT.....	21
4.2	MODEL TRAINING .....	26
4.3	RUN INFERENCE WITH MODEL IN UNITY .....	33
4.4	OUTPUT .....	37
<b>5</b>	<b>USER MANUAL .....</b>	<b>39</b>
5.1	PREREQUISITES .....	39
5.2	SETTING UP THE SCENE .....	40
5.3	DEFINING OUTPUT .....	42
5.4	TRAINING NEW SHAPES .....	42
<b>6</b>	<b>CRITICAL ANALYSIS AND CONCLUSIONS .....</b>	<b>43</b>
<b>7</b>	<b>BIBLIOGRAPHY .....</b>	<b>46</b>
<b>8</b>	<b>REFERENCES.....</b>	<b>48</b>

---

## 1 Introduction, Problem Definition and Background

In the ever-changing, rapidly evolving world of technology and artificial intelligence, programmers need to spend their time wisely; they must be resourceful and avoid “reinventing the wheel”, so to speak. This is why packages and libraries are so important and so widely used; each one encapsulates a different set of functions or objects that allow developers to perform certain tasks automatically, without having to write code that already exists. Generally speaking, video games have a lot in common with each other; they all require a GUI (Graphical User Interface) or, in other words, an output screen for the player to look at. They will typically involve different objects interacting with each other, some form of a point tally, a winning condition, among other features. These have all been written a thousand times and, particularly for big projects, there is no reason to spend valuable time on rewriting their code. Building upon this idea, we can create packages that envelope all the complicated code required to make specific *genres* or types of video games, so programmers can spend their time expanding their games and revolutionising the industry as we know it.

One of the most fascinating yet obscure branch of video games is that which employs *Shape Detection* onto user input (more specifically, it allows a player to draw a pattern, then analyses that pattern to see if it matches a certain shape). I think this style of input is incredibly interesting and fun, which is why I would love to see more of it in modern games. The only issue with this is that shape detection is incredibly complex and, particularly for newer developers, would take an exceptional amount of time to implement. This is why I have taken on the challenge to turn this immensely complicated task into nothing but a couple lines of effortless code.

### 1.1 Problem Definition and Background

#### 1.1.1 Project Aim

The aim of this project is to produce a Unity package that allows developers to match a user's inputted drawing to a member of a predefined list of shapes. To expand on this a little, I would like to create a package that allows game developers to set up a canvas on which their players can draw, then process that drawn input and analyse it to see if it matched any of within a defined list of pre-existing shapes. The program will then execute a certain function or command based on which shape is deemed to match. For example, if the drawn shape is determined to be a *square*, then the *square's* functionality will be executed.

---

### **1.1.2 Research Questions**

In this project, I will explore several topics that will allow me to define and achieve the necessary criteria in order to accomplish the aforementioned Project Aim. The most important information to discuss is, of course, what algorithms and libraries already exist regarding shape/contour detection, image processing, general machine learning, etc.

This project will also dissect previous examples of video games that utilise these algorithms and the concept of recognising user's drawings, with a detailed analysis of how it is achieved, based on whatever information is available.

A certain acceptance criterion must be constructed for the algorithm itself and its ability to detect shapes to a sufficient level of accuracy. This is crucial to determining the actual success of the algorithm and the library itself. Of course, how easy the package is to import and use should also be considered as a gauge for success.

The final primary thought to consider is how exactly the model and the package's programs can be improved, as well as the overall concept itself. For example, I may consider the efficiency of the programs in regard to execution time, tidiness or security, but this project will also keep an eye on the way the different stages of the process are tied together. This is a big point that I'm hoping to improve by means of this sole package.

### **1.1.3 Target Audience**

The target audience for this project is primarily game developers of course, but as I will discuss in future sections, this package may hopefully be expandable to a much wider array of users, many of whom may have no interest in games whatsoever.

### **1.1.4 Wider Context**

Although this package is being designed with a specific focus on video games, the ultimate outcome is to develop it in such a way that it can be used by a much larger, more diverse group of people. After all, game developers are not the only programmers that may want to create an application that analyses a user's hand-drawn input. This problem of having no shortcut or comprehensible path to shape detection is one that all programmers are faced with the moment they decide to tackle the topic. This package could be used for a variety of applications; perhaps, it would provide some comfort to those designing an app to improve hand-eye coordination, or, with the relevant alterations, could help individuals to learn to write or

---

draw shapes. It could also be used for artistic purposes, to teach people in the world of digital art. The potential of this area is quite extensive, given the proper attention.

### **1.1.5 Project Scope**

The deadline for this project's code was set for the 19<sup>th</sup> of March 2024. This is crucial as I had to plan the work in relation to this deadline, scaling it appropriately to ensure that it is completed on time. The first step towards scoping this project relative to this deadline is identifying the set of features that are required to make it work.

The general features of this project are:

- GUI (Graphical User Interface) that encourages a user to draw a shape and passes this information to the algorithm to be processed.
- A pre-processing stage that simplifies the drawing to make the detection of all edges and shapes easier for the algorithm (such as converting to greyscale, smoothing techniques, etc).
- Edge detection/Shape detection algorithm that recognises all the drawn input and identifies common shapes or lines.
- The ability to test these identified lines and shapes against a predefined list to check for the most accurate match, if any.
- A means of relaying the determined shape to the user and executing a certain command or function based on this.

### **1.1.6 Project Plan**

Once the feature set had been defined, I was then able to create a high-level plan for the project, assigning each feature and stage of the project to a certain time frame.

For this project, I utilised an *Agile* methodology, which allowed me to break each feature of the project into much smaller pieces and work on each task for a given time frame. I had chosen this method as it is one with which I have the most experience and success. I was able to break the timeline into two-week sprints and assign each sprint the appropriate amount of

---

work I felt was possible within that time frame. I used *Project Libre* to design the high-level plan for the project which can be seen below.

According to the high-level plan above, the project would be ready for completion by the 15<sup>th</sup> of February, with over two weeks available for refinement and expansion at the end. The plan was to follow this plan by breaking each stage into smaller tasks and completing them, when possible, within each two-week sprint.

### **1.1.7 Important Outcomes**

The outcomes of this project were relatively simple to define; I needed to have a working library that allowed any C# developer to implement a GUI that takes a user's drawing and matches it to a member of a predefined list. The project was to be completed on time; it should be feasible for novice programmers to implement it without researching shape detection; it must be my own work; it should accurately identify the shapes and lines drawn, if drawn to a sufficient standard.

---

## 2 Literature Review and Research

In this chapter, I will be discussing the research that I had undertaken prior to implementing the solution for this project. In order to develop the desired solution to the best possible standard, I had to be sure I took the necessary precautions and explored all the possible methods by which I could implement the different features. I began by researching previous implementations of applications that resemble this package (involve the detection of drawings, etc), to get a general idea of the high-level concepts that are required for such a project. I would then like to discuss the benefits and disadvantages of applications like these, how they can be of use to society and how they could be harmful; the idea here is to conclude that this package is a worthwhile project that will promote beneficial changes in society, rather than cause problems. After conducting research on these topics, I would then begin looking for more specific information regarding implementation of the key features, hopefully exploring a variety of options.

### 2.1 Previous Usages of Mobile Games in Society

In this section I will discuss the previous examples of where similar concepts have been seen. I will look into these areas with the hopes of finding several different potential avenues down which this package could be used.

First, I would like to explore previous implementations of mobile games and their benefits (or lack thereof) on society, what applications exist and how they can help to improve society's shortcomings.

#### 2.1.1 Drawing Apps for Motor Skills Development

One possible application that can be developed using this package is a doodling game that helps the user to develop their motor skills, which would be particularly of interest to children that are first learning to draw and write. Of course, it's entirely possible to develop motor skills using a pencil and paper, but utilising an application such as this has numerous benefits, so perhaps it would be clever to look at this as an *additional* help, rather than a *replacement*. After all, there need not be only one solution.

---

Even if an application does not provide feedback on the accuracy of a drawing, but simply provides the child the ability to draw with creative freedom, you will find that they quickly develop a keen understanding of the concept of drawing and improve their ability to do so, which is a crucial part of a child's development; improving this skill allows them to expand their imagination and create art that exposes their perception of the world and objects within it. Studies have shown that children as young as two are able to use drawing applications in a coherent and intentional manner; for this reason, it is fair to say that developing applications such as *Magic Marker* (created by *Jaytronix*) does not set unrealistic expectations on children's ability to use mobile applications. (Yadav & Chakraborty, 2017)

Children are not the only target audiences for these applications, however; just as we are born without fine drawing and motor skills, so do we begin to lose them as we get older. Because of this, it is important to look at this problem from the other end of the spectrum as well, as many elderly people could also find these applications useful. Some applications, such as *Guess It*, a game developed to help elderly people improve their abilities to navigate through mobile environments and improve their understanding of mobile phones and games, need not put a direct focus on their drawing capabilities and still see the benefits.

The use of simple login features, such as the one implemented into *Guess It*, can have two benefits: the elderly people get a brief introduction to the concept of personal profiles and logins, and they can keep track of their progress by linking previous attempts to their "account". This is something to consider when developing an application such as this using the package I have proposed. (Foo et al., 2017)

There were countless examples of mobile applications designed to *evaluate* the motor skills of children and the elderly, but I found a sincere lack of those designed to actually *improve* these skills, which is why I believe this package could spark a great change and promote more of these applications to be developed in time. As I said before, it's very important for children and people in general to develop these skills, so I think this issue should be given the appropriate attention by us developers.



---

The final point that I wanted to research with regards to motor skill development was the use of applications in *rehabilitation*. It appears to be well documented that patients of various physical conditions (such as Parkinson's disease, cerebral palsy, stroke, etc) have seen noticeable improvements in their physical rehabilitation, akin to that achieved through conventional therapy. (Unibas-Markaida & Iraurgi, 2021, p. 13)

### **2.1.2 Drawing Games in Education**

These applications and games have many possible uses outside the realm of physical development; for example, the world of education has seemingly unlimited room for improvement which could be vastly promoted by utilising drawing applications. Possibly the most prominent evidence of this need was the era of the COVID-19 pandemic, which inspired several lockdowns worldwide, placing a huge emphasis on the need for digital implementations of certain educational tools.

Particularly with younger groups, the use of educational games in schools inspires dramatic change in the student's enjoyment and dedication to their studies. By using games and digital applications, students can be motivated to engage more heavily in their education, as these games utilise the valuable techniques that are typically used to keep a player's attention. Thus, by keeping a player's interest in the game, so do they keep their interest in their education. (Mulhem & Almaiah, 2021, p. 10)

Mobile Game-Based Learning has been proven many times to improve the effectiveness of a student's education, so it's important that this is given the appropriate attention, as I've mentioned before. Personally, given the state of the world and the consistent developments of mobile devices and digital applications, I believe it's inevitable that these will become imperative to the educational field. It's been proven over and over again that using these applications will have benefits in education, though it will not come without its challenges. (Nisiotis, 2021, p. 2)

Children that use applications such as these have a strong probability of improving the essential skills targeted by education, particularly younger groups, such as literacy and ability to visual three-dimensional objects; it has also been proven that students that engage in these

---

games and applications are likely to produce significantly better grades (particularly in mathematics and reading) than those who do not. (Moosa et al., 2020, p. 2)

## **2.2 Evaluation of Mobile Games – The Benefits and Disadvantages**

Before exploring specific methods of implementation, I wanted to look a little more closely at the effects of mobile game, both the positives and negatives. I will not linger on the topic too much, but I do believe it is imperative that we give this the proper attention before attempting to develop a new application or package.

### **2.2.1 Mobile Games and Addiction**

One of, if not *the* biggest, concerns with mobile games and video games in general, is the potential for *addiction*. Video gaming is widely considered to be a highly addictive hobby, which is why many parents and guardians are often hesitant about allowing children to engage in them, fearing it could lead to loneliness, depression and similar mental health issues. I wanted to briefly explore this fear and see just how valid the concerns are.

On one hand, it is quite clear from the evidence that there is a correlation between mobile game addiction and mental health issues, which should be of concern to parents and guardians of adolescents. (Wang et al., 2019, p. 4)

On the other hand, we cannot simply conclude that mobile games are therefore harmful, particularly those similar to the applications I have proposed. The research shows that males who are addicted to mobile games return a higher rate of mental health issues rather than females. This could be due to the differences in gender, but it is also worth noting that the vast majority of males engage in a different *genre* of game to that engaged by females; so, I conclude it may be the *type* of video game that truly affects one's mental health, as opposed to mobile games in general. (Wang et al., 2019, p. 2)

I thought it would also be worth my time to analyse this fear that parents experience about their children's use of mobile applications and consider how it could be alleviated. The research returned one particular topic that I thought should be considered, which is the idea of the "*intergenerational transmission theory*", which suggests that psychological issues such as stress and addiction can be transmitted from parent to child, which leaves some adolescents

---

with a higher risk of addiction to mobile games and may trigger this fear in parents. (Mun, 2022, p. 2)

### **2.2.2 Mobile Games in Language Learning**

As I mentioned previously, mobile games can have significant benefits in education, but what I didn't discuss is the employment of these games in the field of language learning. Research has shown that a comparison between mobile games and non-mobile game, with regards to language learning, yields a higher acquisition of *speaking* and *writing* skills in the target language for *mobile* games, but a higher acquisition of skills such as *vocabulary* and *grammar* for *non-mobile* games. (Su et al., 2021, p. 9)

### **2.2.3 The popularity of violence-oriented mobile games**

A large percentage of the fear of video games can be attributed to the popularity of violence-oriented games such as the “*Grand Theft Auto*” series, or war-based games such as the “*Call Of Duty*” franchise. I think it's important to discuss this and to do my due diligence regarding this trend, especially before producing a package such as this, that has such a diverse range of use case scenarios.

After some research, I discovered that a significant portion of video games played by adolescents contain violence and promote bad habits such as gambling and strong language. Many games with ratings as low as “*Pegi 7*” (age 7+) contain elements of violence and horror, which is a worrying statistic to say the least. A vast majority of mobile games are proven to contain elements that promote consumerism, even with ratings as low as “*Pegi 3*”. (Göksu et al., 2020, p. 399)

## **2.3 Google's *Quick, Draw!***

A popular game that utilises *Doodle Recognition*, like the kind proposed for my package, is Google's “*Quick, Draw!*”. The concept of the game is this: the player is told to draw a certain image and, as they draw, the system will attempt to guess what they are drawing; if it determines that the drawing matches the same word that was given to the player, within the time provided, the player wins.

---

A useful aspect of this application is that Google have published the datasets of images used to train the system, in their original format, so that developers may use these images to train their own models. I thought this could be of interest to me as I develop my own artificial intelligence model, so I decided to do some further research into the dataset and decide if it could be of use to me.

The dataset has been used before to develop systems of all types and purposes, given its large size and diverse set of sketches, containing 345 different categories of images, all of size 28 x 28 pixels. It can be used to train a custom Tensorflow model to detect the respective classes, however, the dataset is very grand and quite specific in its formatting, which is why I deemed it unnecessary to use for my package. (Tsai et al., 2019, p. 13)

## **2.4 Shape & Doodle Recognition**

Finally, to ensure this package reaches the appropriate level of quality, I had to conduct a lot of research into the ways in which current games and applications utilise shape recognition; how it is implemented into the programs and what use cases have been developed for it.

### **2.4.1 Previous Examples of Shape Recognition Usages**

Shape recognition is widely used in *computer vision* projects (i.e. projects that involve having a camera *look at* an image or video and extract certain qualities or statistics from it. One such proposed project uses a combination of several open-source engines and libraries to recognise characters within a vehicle's license plate, as well as the type of car. (Herusutopo et al., 2012, p. 80)

The proposed project follows a simple general flow for detecting the car type and license registration – Perform pre-processing steps (such as greyscaling, noise removal, etc), pass into the *artificial neural network*, perform the necessary output and finish. This is a generalisation of the process, of course; the specifics are much more complicated, and I will discuss these shortly.

---

Microsoft's *OneNote* application also uses shape recognition for its "Ink to Shape" feature, wherein the user can begin drawing a certain shape and the software will determine which shape is being drawn, overriding the shape with a perfect geometric version of this detected shape. The backend processes and algorithms used by Microsoft for this feature is evidently not disclosed to the public, but I can imagine it follows a similar pattern to that of the registration plate system.

In my research, I found countless examples of use cases for such a system, machine learning and object detection are themselves used worldwide for an enormous range of purposes, from vehicles to simple smartphone applications, it seems the possibilities are endless and with a package such as the one I have proposed, creating such applications and games using Unity will be much easier and significantly quicker.

#### **2.4.2 Methods of Implementation**

From my research, I have discovered an abundance of ways to implement the different features for such a package, however, the system itself seems practically guaranteed to follow the same general flow: capture the image, pre-process the image (as required), detect the object or *objects* within, submit an output to the user.

The shape detection itself can be implemented in a variety of ways, using one of hundreds of machine learning model libraries or types. One name that will consistently appear during research of such systems is *EmguCV* (an extension of the more popular *OpenCV*); This library simplifies the process of object detection and allows developers to perform all the necessary steps for the detection of objects within images. It has the capabilities to perform the necessary pre-processing steps such as greyscaling, resizing, etc. (Magdin et al., 2022, p. 13)

The aforementioned "car type and registration recognition" project utilised *EmguCV* as well, along with *Tesseract*, an open-source engine for *Optical Character Recognition (OCR)*, a conceptual branch of computer vision, which reads the characters from the plate and returns the result. This concept is also commonly used when creating digital forms of physical papers and documents. (Herusutopo et al., 2012, p. 78)

---

To build the artificial intelligence model itself, there are more options than I would have thought at first, so much so that I knew I would struggle to pick just one.

Many public libraries exist with the sole purpose of simplifying the process of creating and training a custom model, even for object detection specifically. The most common libraries, I found, were *Tensorflow* and *PyTorch*; both of these are open-source machine learning frameworks that were designed specifically to help developers train custom AI models. Tensorflow has its own API specifically created for object detection, to allow programmers to either use a pre-trained model or to train their own custom one. With libraries such as *Keras*, developers can build their own object detection models layer-by-layer to configure the model for their desired purpose.

PyTorch is a machine learning framework developed with the sole purpose of allowing developers to build their own deep learning AI models. It includes a module, called “Torchvision”, which is specifically for object detection. Both appear to be well-received, but it seems as though Tensorflow allows for more customisation and manual configuration, so that is likely the favourite of the two.

When it comes to developing these models and obtaining them on a personal device, there are several ways to go about it; they can be built on a local machine, though it’s recommended this is done inside a virtual Python environment, there is also the option of simply building it on a virtual machine, removing the risk of harming any configurations on your personal device, or finally, to use an online environment such as *Google Colab*, which comes pre-installed with various libraries and configurations that could save a developer a lot of time.

### **2.4.3 Potential Uses going Forward**

Finally, I wanted to briefly explore the possible use cases for a package like this before beginning development. It didn’t take long to find examples of games and applications that use a similar concept, as I have mentioned before, so I was confident that this would be a worthwhile project.

If this package is successful even to a *Make-It-Work* state, the potential for uses is seemingly endless. Video games, educational applications, rehabilitation support, even security and au-

---

thentication has uses for shape recognition, possibly using a user's drawings to authenticate their access to a certain application or account. (Wazir et al., 2020, p. 12)

As the package is developed further, it can be expanded to support a host of different applications and games, making it even easier and more efficient to implement the shape recognition system. With all this in mind, I was excited to finally get the project underway.

### 3 System Design and Configuration

#### 3.1 Specification

As discussed previously, the goal for the package is to allow a developer to quickly implement a system by which a user can draw using touch or mouse input, then capture this drawing, pass it into an object detection model and execute a certain set of commands (defined by the developer) based on the shape detected. Therefore, the system *must* be defined in such a way that the developer need only import the required assets, then define the specific commands for each shape and the system does the rest.

#### 3.2 User Story Mapping

The following figure contains the User Story Map detailing a high-level view of the system, from the perspective of a Make-It-Work, Make-It-Better, Make-It-Best methodology (i.e. what is needed; what would improve the system but is not required; what would be nice to have).

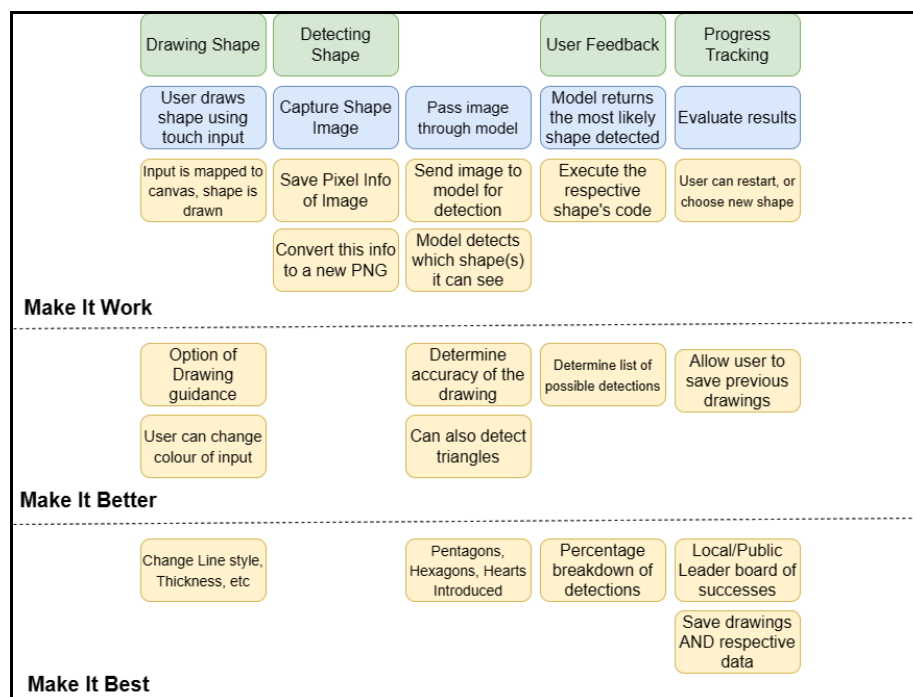


Figure 1 - Project User Story Map



---

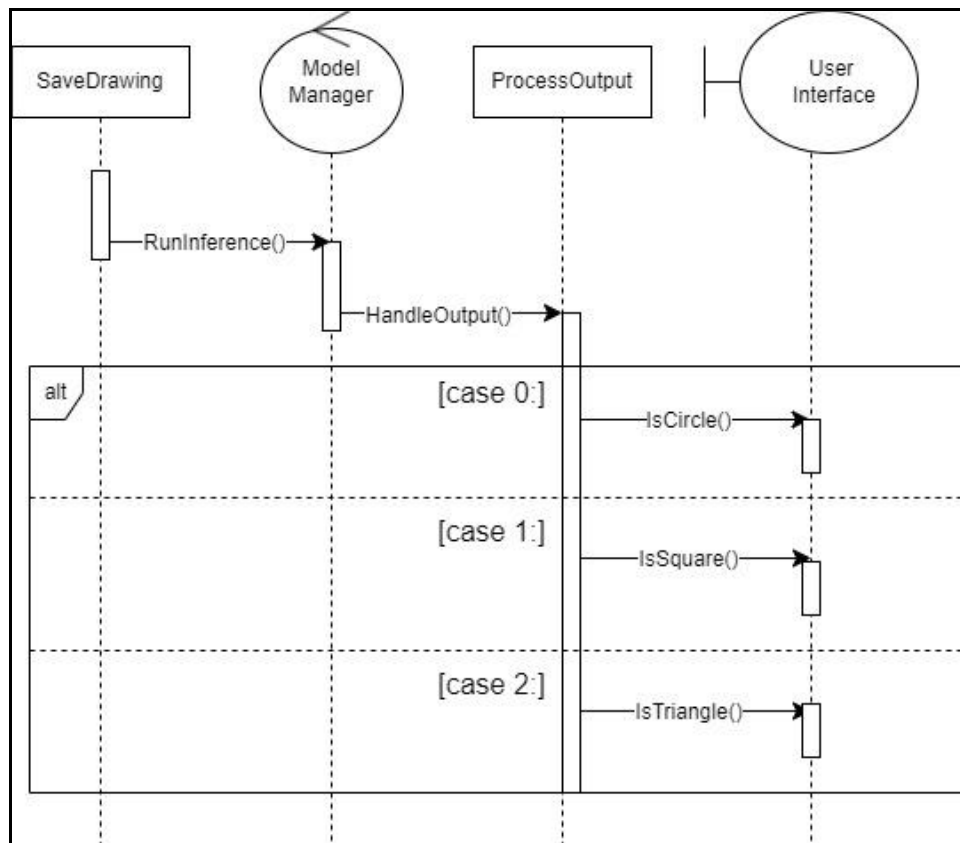
The Make-It-Work section above (what is required) details a functional package that allows the user to draw using touch input, as specified, by mapping the location of the touch from the screen to the canvas object, then begin drawing. This is a high-level view of the design; more specifics will be provided in the following *Implementation* chapter. The next step in the system is to detect the shape which the user had drawn (simply either a Square or Circle, originally); the first course of action here is to capture the image drawn by the user, by reading and saving the pixel data of the canvas, then passing this data to the object detection model, which would locate and determine the shape drawn. Finally, the package would call the respective code, defined by the developer, depending on the shape that was detected. **Note: The original design for the system converted the drawing's pixel data to an image (.png image file), as shown in the map above, this image would then be converted back to the format required by the model. I will discuss this design, its benefits and disadvantages, later.**

The Make-It-Better design implemented some interesting features that would improve this system further. For example, the drawing feature could include the option to change the colour of the line being drawn, which just gives the user some more personalisation, or perhaps they could be given the option to have guidance with their drawing, which specifically tailors to the applications designed to improve the user's drawing capabilities. Furthermore, the model could determine the accuracy of the drawing i.e. how confident it was that the shape determine was the desired shape. Another way to improve the package is to train the model so that it can detect an additional shape, specifically, triangles.

The Make-It-Best version of the package could introduce detection of a variety of new shapes, such as those specified in the User Story Map above; it could also allow the user to save their drawings and their respective data (accuracy, etc) to a database, or leaderboard, perhaps. These are *nice-to-haves*, of course, and are by no means necessary for the project. However, by saving the user's drawings to a public database, the images could potentially be used to further train the model to detect shapes more accurately, or simple allow the user to reflect upon their previous attempts.

### 3.3 Sequence Diagram

As this package utilises a system wherein a lot of work behind the scenes is done in one swift flow, it is perhaps best described as a sequence of steps. Thus, I believe a sequence diagram can most accurately convey the procedure by which the system operates. Consider this diagram shown below.



**Figure 2 - Project Sequence Diagram**

The sequence diagram above shows the general flow of the system, from the point the drawing is complete, to the point that the output is displayed to the user. The lifeline object “SaveDrawing” represents the class that captures the user’s drawing and passes it to the model for detection.

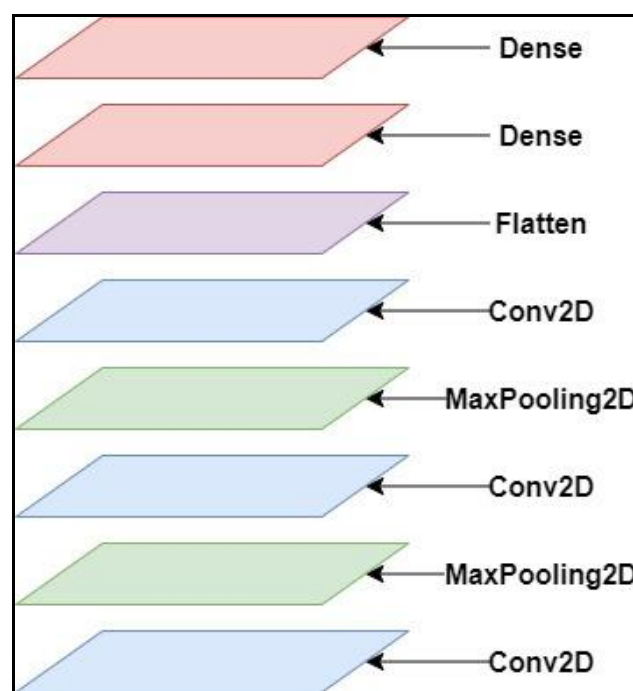
Once the user decides to submit the drawing, the data is passed to the control lifeline “Model Manager”, which runs inference using the image and determines the shape drawn. The result returned (‘0’ for Circle, ‘1’ for Square, ‘2’ for Triangle, more on this later), will then be read

---

in by the “ProcessOutput” class and the function, respective to the shape detected, will be called. The developer using this package will define what they wish to do upon certain shapes’ detections, all of which will certainly display a form of result to the user.

### 3.4 Object Detection Model Layers

A key part to creating a custom AI model is configuring it with the correct type and number of layers. There are many layers that could be considered for such a model as this, some are essential, others simply brought the model to a higher level of accuracy. I will briefly discuss these layers and the purpose of each, using a simplistic diagram to visualise the model’s structure. The layers used in this model were all decided through research of Tensorflow’s *models* library and analysis of each layer’s function; the structure of the model is relatively simple and follows a common approach.



**Figure 3 - Model Layers Diagram**

**Conv2D:** The two-dimensional convolution layer is fundamental in machine learning with image processing; in simple terms, these “convolutions” involve taking small pieces of a picture, called “kernels” or “filters” and sliding it across the targeted image, which helps to ex-

---

tract common features of these images like edges or patterns. There are a total of three two-dimensional convolution layers in the model's design, as shown in the figure above, which was the result of a trial-and-error process consisting of deploying the model, testing it, and reconfiguring it to get the best possible accuracy results.

**MaxPooling2D:** The max pooling layer essentially reduces the “noisier” parts of the image, extracting only the pixels with the brightest value in certain regions; this results in an image with much less complex data and simplifies the following convolution.

**Flatten:** The *flatten* layer is fairly self-explanatory; it reduces the multi-dimensional data of the image into a single dimension, which is essential for transitioning from the preceding convolution layer to the following dense layer.

**Dense:** Finally, the dense layer is the most complex conceptually, in my opinion; the process involves connecting each neuron of the previous *flatten* layer to each neuron in the current dense layer. The purpose of this is to analyse and adjust the *weight* of each connection, ultimately determining the relationship between these neurons.

---

## 4 Implementation & Testing

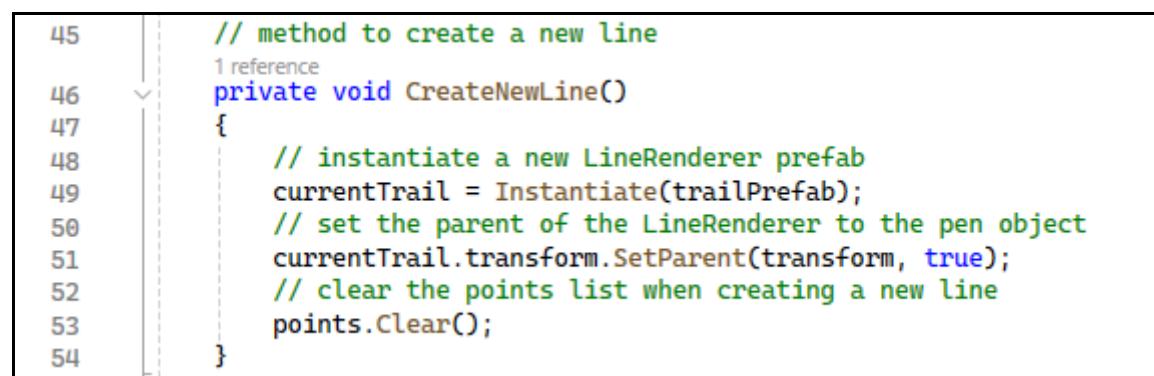
This chapter will detail the journey of the implementation of the project, from beginning to end, following the general sequence and design specified in the previous chapter. As stated before, the project utilised an agile framework, which meant that tasks from different features could be implemented simultaneously; however, priority was generally given to the earliest features in the system sequence (i.e. in the form, from highest to lowest priority: *Input, Image Capture, Model Processing, Output*). For this reason, I will describe the implementation by feature, in this same sequence, detailing the issues encountered and resulting solutions developed.

### 4.1 Input

The input of the package is handled using a combination of various built-in Unity modules and some custom code primarily utilising Unity's Line Renderer component.

#### 4.1.1 Draw Line

The following code draws the user's lines onto the canvas (*Paper*) object, which entirely takes place within the "*DrawLine.cs*" file. When the user initially touches the screen (or presses the left mouse button down), a function simply entitled "CreateNewLine" is called. This function is used to instantiate a new Line Renderer object, which will essentially be the newest line drawn onto the canvas.



```
45 // method to create a new line
46 1 reference
47 private void CreateNewLine()
48 {
49     // instantiate a new LineRenderer prefab
50     currentTrail = Instantiate(trailPrefab);
51     // set the parent of the LineRenderer to the pen object
52     currentTrail.transform.SetParent(transform, true);
53     // clear the points list when creating a new line
54     points.Clear();
55 }
```

Figure 4 - CreateNewLine Function

Line 49 above does just this; it instantiates a new object of the LineRenderer prefab that comes with the package, saving it to the local variable “currentTrail”. This variable then has its parent set to the “Pen” object, to which this script is attached (this makes it very easy to clear the canvas, when needed). Finally, in Line 53 above, you can see the List “points” is cleared – the purpose of this List is to keep track of all the points (vertices) in the local LineRenderer object.

Once per frame, while the user continues to hold down the touch input, the function “AddPoint” is called, which adds a new point to this LineRenderer object through a Raycast-Hit collision. The position of the touch (or mouse position) is used to cast a ray from the screen through the game scene.

```
59 // ray variable from the input position on the screen
60 Ray ray = Cam.ScreenPointToRay(Input.mousePosition);
```

**Figure 5 - Cast Ray from Screen Point**

Notice the “Input.mousePosition” is used regardless of the input type, be it mouse or touch input; I decided to use this specific variable as I discovered through my research that Unity detects this as either input type, so it saved some code. I then used a simple pair of nested “if” statements to detect a collision between this Ray and the Paper object.

```
61 // new raycast hit variable to detect if ray
62 // collides with paper object
63 RaycastHit hit;
64 if (Physics.Raycast(ray, out hit))
65 {
66     // check if the object hit is the paper object (tagged "Paper")
67     if (hit.collider.CompareTag("Paper"))
68     {
```

**Figure 6 - Handle Collision with Paper**

Line 64 casts this ray and outputs a detected collision to this RaycastHit variable; from there, all that is required is to get the collider that met the ray and compare its tag to that of the Paper object. If there is a match, then a new point shall be added to the Line Renderer object, by way of an “UpdateLinePoints” function, that simply adds a point to the “points” List and re-

---

sets the `LineRenderer`'s points to the points in this List. This process is repeated as long as the user holds down their finger or mouse button. If they raise their finger and press it down again, the process restarts from the “`CreateNewLine`” function call.

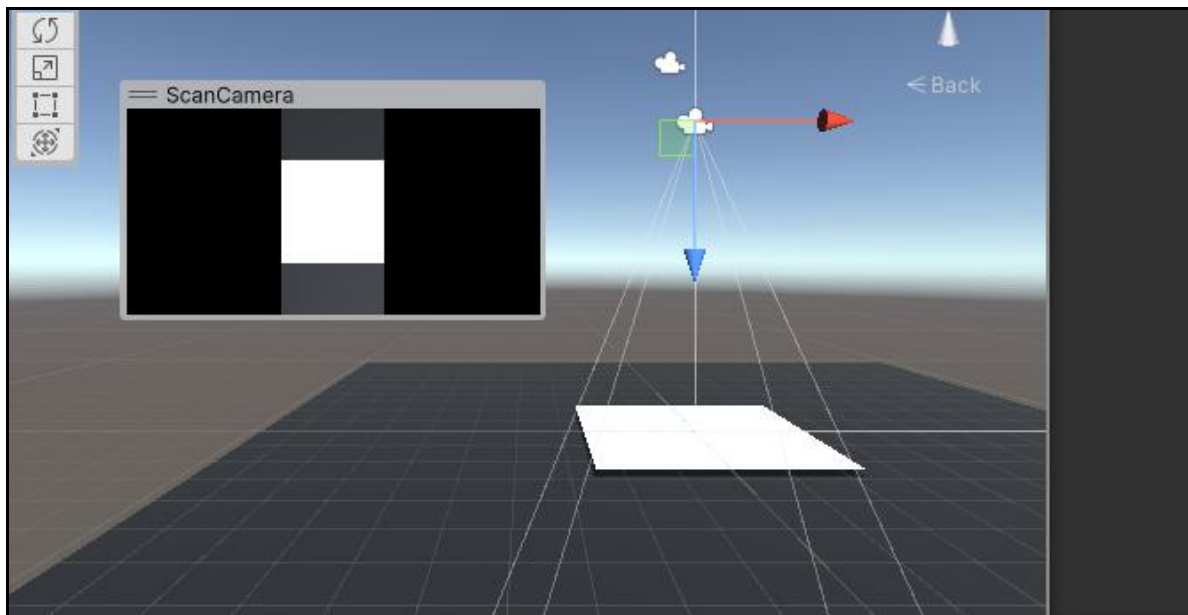
#### 4.1.2 Capture Drawing Image

In order to process the drawing and to detect any shapes within it, we first have to capture it as an image. The original design for the image capture was this:

- Read in the drawing – pixel by pixel, saving it to an instance of Unity's `Texture2D` data type.
- Convert that texture data to an image, in PNG format.
- Pass that PNG image to a preprocessing file – resizing it, turning to greyscale, etc.
- Read in this new PNG and convert it back to `Texture2D` format – to be used in the model.

The same general flow was used in the end; however, I encountered some issues during the *preprocessing* stage, which I will discuss shortly. Firstly, let's walk through the capturing of the image in code, as it is vital to the system. This all takes place inside the “*SaveDrawing.cs*” file, which is attached to the *Confirm Button* object in the package.

This process entirely revolves around the *Scan Camera* object, as this is what will constantly “look at” the canvas and capture the image in its entirety. The *Scan Camera* is a child of the *Paper* object, in order to ensure it remains its relative position, so that the captured images will be consistent.



**Figure 7 - Scan Camera in Scene**

Firstly, a *Render Texture* is created with the desired width and height for the captured image; this texture is then set as the target texture for the Scan Camera, as shown in lines 31 and 33 in the figure below, respectively.

```

30      // create a new RenderTexture with the desired width and height
31      RenderTexture rt = new RenderTexture(textureWidth, textureHeight, 24);
32      // set the target texture of the scan camera to this new RenderTexture
33      scanCamera.targetTexture = rt;

```

**Figure 8 - Setting the RenderTexture**

Finally, we simply render whatever the Scan Camera sees and read it into a new *Texture2D* object of the same size.

```

34      // Texture2D object to which the drawing input will be rendered
35      Texture2D texture = new Texture2D(textureWidth, textureHeight, TextureFormat.RGBA32);
36      // render the texture seen by the scan camera
37      scanCamera.Render();
38      // set the RenderTexture object to be active
39      RenderTexture.active = rt;
40      // read in the pixels to the texture object
41      texture.ReadPixels(new Rect(0, 0, textureWidth, textureHeight), 0, 0);
42      // apply the texture changes
43      texture.Apply();

```

**Figure 9 - Render Image to Texture**



---

Line 37 above calls the *Render* function which, as the name implies, renders the image seen from the Scan Camera's point of view. The Render Texture is then set to be the active one in line 39, before reading in the pixels seen into the newly-created Texture2D object (lines 41-43).

And that's that, the image is captured whenever the confirm button is pressed; as I mentioned before, the plan was to convert this to PNG format and save it to a local path, to allow pre-processing of the image. My intention was to use EmguCV (a *.NET* wrapper for OpenCV), which would allow me to perform the required pre-processing steps to the image.

My prior research returned several confirmations that this would be possible inside Unity, but not without its challenges. Regardless, I decided to take the risk and set out to import this into my project. I will not linger on this issue for too long, as I discovered a workaround for the time being, but the attempt developed like so:

- I attempted to import EmguCV into my project through Visual Studio's Nuget Package Manager, which was successful.
- I then tried to link it inside Unity through its package manager, per EmguCV's online instructions. This was successful at first, but upon reload it caused further errors (not being able to find the EmguCV library), once the code was written.
- I reset the Nuget Packages and decided to try a local install, to at least ensure that would work. This resulted in similar errors, whereby Unity could not recognise the EmguCV library.

I repeated these installs several times, each with different configurations, as well as troubleshooting the errors and researching possible solutions. I also considered using OpenCV itself – although, given its configuration, I decided it would be overly time-consuming to attempt to utilise it through Unity. There is a public package entitled “OpenCV For Unity”, that would resolve any of this confusion, however my package is intended to be free and easy to use, so outsourcing this step to a costly external package is counterintuitive.

---

After over a week's diligent work, I decided the cost-to-necessity ratio was simply too high to spend any more time on this issue – especially regarding my project plan. I reconsidered how necessary the entire *pre-processing* step was, particularly when one remembers that I already configured the Render Texture's size from before, so this would almost be redundant work – to capture it in one size just to convert it to another straight away. With this in mind, I augmented the design of the *image capturing* step and settled to set the texture's width to that which is expected by the model (32 x 32 pixels).

```
8 // local variables
9 private const int textureWidth = 32;
10 private const int textureHeight = 32;
```

**Figure 10 - Defining Desired Image Sizes**

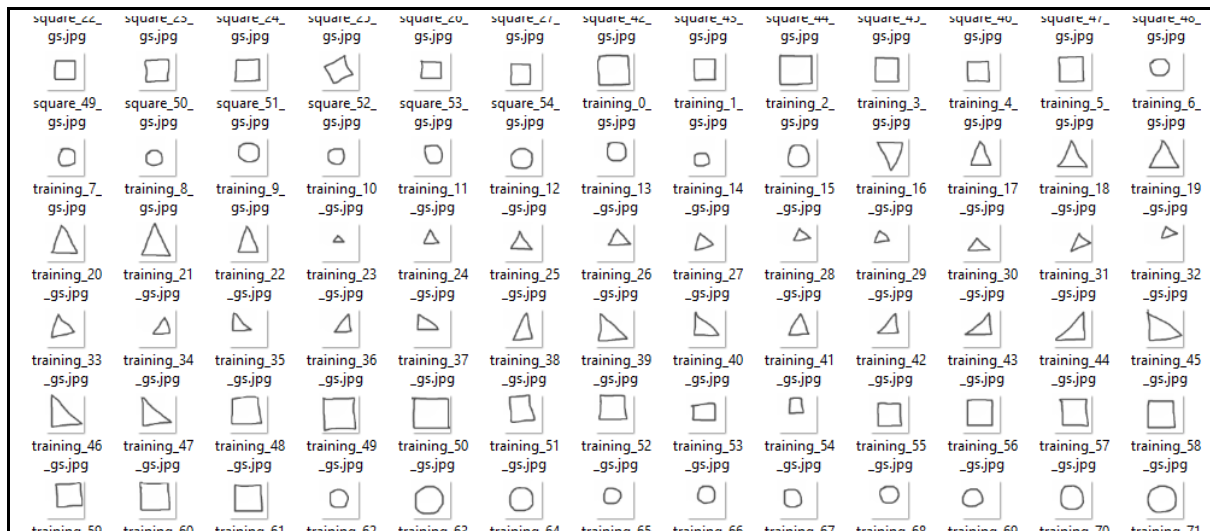
So, in conclusion, the image will be captured in the desired size, saved into a Texture2D instance, and then sent immediately to the model for detection.

## 4.2 Model Training

The most crucial part of this project was to create an artificial intelligence model that can detect which of a predefined set of shapes is present inside an image.

### 4.2.1 Gathering Image Data

As per the Make-It-Better stage of the project, I managed to achieve an AI model that was trained to recognise *three* shapes: circles, squares and triangles. Luckily for me, gathering the training images was a simple process, because I already had *everything* that I needed to create a plethora of images, all in the expected format for the model. I took advantage of my drawing input and image-capturing system and, with a slight augmentation to the code (saving the drawing to a new PNG file upon each click of the *confirm* button), I developed a large mass of images (each containing one of the three shapes, in all sizes and orientations) that were ready to help train and test the model.



**Figure 11 - Set of Training Images for Model**

#### 4.2.2 Initial Model Creation Attempt (Tflite Model Maker)

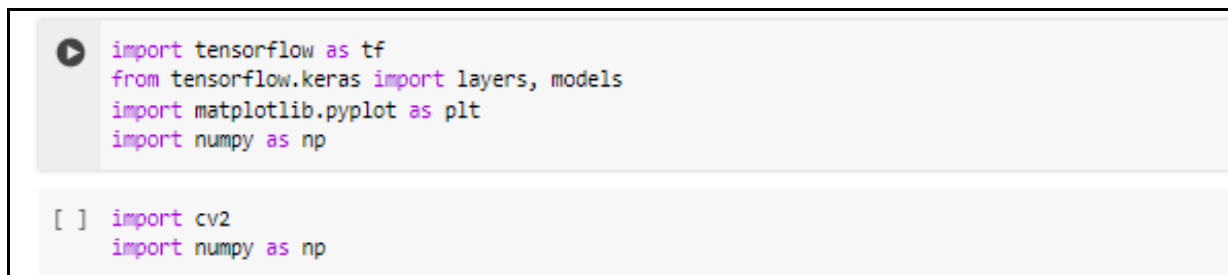
During my research, I discovered an open-source module, created to exist alongside TensorFlow, called “Tflite Model Maker” which was specifically designed to help make the process of building and training certain AI models significantly easier. I thought it too good to be true, but I was convinced I would be a fool not to try it, at the very least. So, I read all about the library during the research phase of my project in the hopes of discovering that it would be possible to build a model based on my custom images. After a significant amount of reading, it seemed as though it was not only *possible*, but actually quite easy; all that was required of me was to convert the image dataset into *csv* format, which was certainly manageable.

Then it came time to implement this; the first step was to import the Tflite Model Maker library onto my local machine using *pip*. The library had a large number of dependencies, with various specific version configurations that had to be perfect. Upon first installation, the command returned a set of errors, which was confusing to me, but I researched them and amended the issue with the necessary commands (specifying certain versions of dependent libraries, etc). Every time one issue was resolved, a new one appeared; so, after trying the installation in countless ways (virtual machines, virtual environments, different local environments, online environments such as *Google Colab*, etc), I was still unsuccessful. In the end, I decided that taking this shortcut was causing more of a delay than it would to build the custom model myself (using one of TensorFlow’s less-evasive libraries).

---

### 4.2.3 Final Model Creation

As mentioned previously, the final model was constructed using Tensorflow's *Keras* library inside a Google Colab environment. There are numerous benefits of using an online environment such as this, most obvious of these is the fact that it comes with a lot of libraries pre-installed, such as Tensorflow itself. Once I uploaded my training and testing images to my Google Drive and mounted it in my Colab notebook file, I was ready to begin building the model.

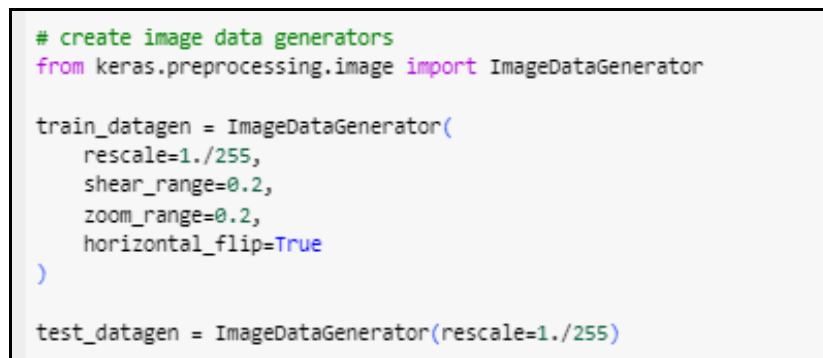


```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np

import cv2
import numpy as np
```

**Figure 12 - Importing Necessary Libraries**

First, I imported all the necessary libraries, as shown in the Python code above. The next step was to create an image data generator, which is used to manipulate the images, giving the model a much more diverse and expansive range of data to work with.



```
# create image data generators
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

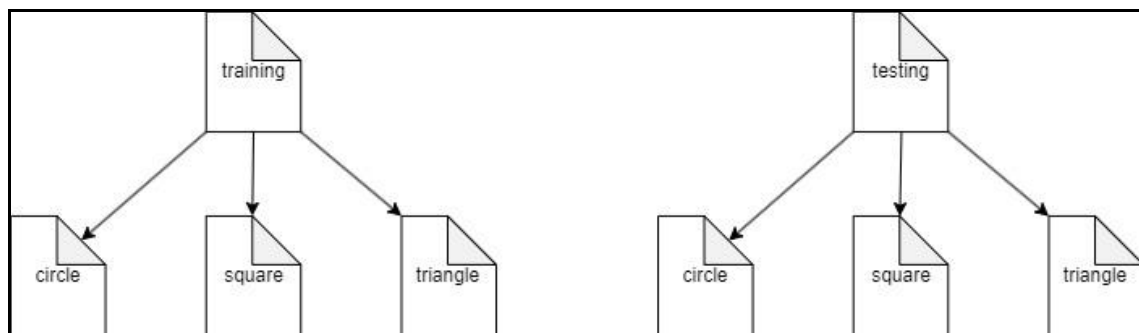
test_datagen = ImageDataGenerator(rescale=1./255)
```

**Figure 13 - Defining Data Generators**

The “rescale” parameter essentially means the generator will alter the size of the images to diversify the data; the “shear\_range” value is a float determining the intensity of the shear, the “zoom\_range” value specifies that we want a lower zoom limit of “1 – 0.2”, which is 0.8, and an upper-limit of “1 + 0.2”, or 1.2. Finally, setting the “horizontal\_flip” value to true means

the input images will randomly be flipped upside-down upon training. Although I tried to diversify the images as much as possible upon creation, these image data generators will only further improve the accuracy of this model, so I think it's clever to make good use of their options.

**NB: I am using *directories* to define the various classes of images (i.e. which shape they contain), which is why they are divided into sub-directories with their respective class names as the sub-directories' titles. Consider the simple diagram below:**



**Figure 14 - Structure of Training and Testing Folders**

Each element represents a folder – inside both folders *training* and *testing* are sub-folders entitle respective to the shapes in the images they contain.

Before I applied the image data generators, I saved the paths to both *root* folders (training & testing) from my Google Drive into local string variables, as shown in the figure below. I wanted to show this because it is *imperative* that these are the paths we use going forward.

```
# path to root training folder
trainingImg_path = "/content/drive/MyDrive/training/"
# path to root test folder
testIm_path = "/content/drive/MyDrive/testing/"
```

**Figure 15 - Defining Paths to Images**

To apply the generators using this directory structure, we utilise the “*ImageDataGenerator.flow\_from\_directory*” function call with the parameters shown in the figure below.

```
# apply generators
train_generator = train_datagen.flow_from_directory(
    trainingImg_path,
    target_size=new_size,
    batch_size=32,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    testIm_path,
    target_size=new_size,
    batch_size=32,
    class_mode='categorical'
)
```

**Figure 16 - Applying Data Generators**

The parameters shown above are defined as follows: the first parameter is simply the path to the root image folders mentioned previously; the second parameter “target\_size” is the size, in pixels, to which all the images will be resized (in my case, I specified a tuple “new\_size”, which I previously defined as a pair of integers “32, 32”); the third, “batch\_size”, is the number of images that will be in each batch of data; finally, the fourth parameter “class\_mode” is defined as *categorical*, which means the label arrays returned will be 2-dimensional one-hot encoded labels.

At this point, we officially have the image data ready to train and test the model; so, now it’s time to actually build the model layer-by-layer. Firstly, we define the new model of type *Sequential* using Keras’ *models* library.

```
[ ] from keras.models import Sequential
    from keras.optimizers import Adam
    num_classes = 3

    # build cnn model
    model = models.Sequential()
```

**Figure 17 - Define New Keras Sequential Model**

I defined the number of classes (shape types) into a new variable as you can see in the third line of the figure above, as well as utilising a call to Keras’ “*models.Sequential*” function to create a new blank model.

---

Now, we add the layers. The figure below shows the code to add each layer onto the model.

```
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# add dense layers on top
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))
```

**Figure 18 - Adding Layers to Model**

The model follows the exact form defined in Chapter 3, but I will give an explanation for the specific parameters here, as they make a big difference in the outcome of the model.

- **Conv2D:** The first argument defines the number of filters or kernels per convolution; the second defines the size of each kernel, the “activation” argument specifies the activation function to use, in this case “relu”; the “input\_shape” parameter defines the output shape of the convolution through the number of filters, new rows and new columns.
- **MaxPooling2D:** The only argument passed into this layer defines the “pool\_size” i.e. the size of each “pool” over which to take the maximum value.
- **Flatten:** The call to “*Flatten*” above takes no arguments.
- **Dense:** The two calls to “*Dense*” define the units (or dimensionality) of the output space and the activation function (as seen in the Conv2D calls), respectively.

The model is now defined, which means that we can now compile it using a call to “*model.compile*”.

```
# compile model
model.compile(optimizer=Adam(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

**Figure 19 - Compiling Keras Model**

The parameters specify the optimiser instance (I used the Adam optimiser), the loss function and the metrics that will be evaluated by the model during the training and testing, in this case, “accuracy”.

We are now ready to train the model! One call to the “*model.fit\_generator*” function is all it takes.

```
[ ] # train model using fit generator
history = model.fit_generator(
    train_generator,
    steps_per_epoch=train_generator.samples // 32, # total samples / batch size
    epochs=15,
    verbose=1
)
```

**Figure 20 - Training the Model**

Here we provide the data generator results and specify some configurations; we define the “steps\_per\_epoch” to be equal to the total number of samples divided by the size of each batch, we specify the number of epochs (in the figure above, I specify 15 epochs, this was the result of trial and error during testing – which I will discuss shortly), finally we specify the verbosity mode; in this case, I used “1” which defined it as a “progress bar”.

The local model is officially trained, now, before we export it and bring it into Unity, we ensure that it’s giving the results we want, and so we test it.



```
[ ] # evaluate the model
    evaluation = model.evaluate(
        test_generator,
        steps=test_generator.samples // 32
    )
    print(f"Test Loss: {evaluation[0]}, Test Accuracy: {evaluation[1]}")

1/1 [=====] - 0s 419ms/step - loss: 0.1371 - accuracy: 0.9688
Test Loss: 0.13712404668331146, Test Accuracy: 0.96875
```

**Figure 21 - Evaluating the Model Performance**

The evaluation above returned an accuracy of approximately 96.9%, which is quite good for an AI model, so I was happy to see that worked out. I will describe how this was imported into Unity shortly; but first, I'll expand on the concept of testing the model very quickly.

#### **4.2.4 Model Testing and Reconfiguring**

So, from the previous section we can evaluate the accuracy of the model using test images that we've created, which is fantastic and provides a quick insight into how the model is doing, so I was pleased to have done this. Of course, this wasn't the only testing done, however, as this model was far from perfect in the beginning. When I originally exported the model to the desired format and brought it into Unity, I noticed some inconsistencies. I had to accept that the model was never going to be perfect, because no AI model is, I knew that sometimes errors happen, but that's okay, as long as I improved the model wherever it fell short.

Particularly in the early versions of the model, it seemed to struggle significantly more when detecting squares; it hardly ever returned "square" as the answer, it would typically return "triangle" instead. I was glad to have caught this early on in the process, because it meant I could simply develop a number of additional images of squares and retrain the model with this expanded set. After doing this, I saw a significant leap in success with the model's ability to detect squares, so I was thoroughly relieved.

### **4.3 Run Inference with Model in Unity**

At this point, we have the model in Google Colab ready to go, as well as a Unity project with the necessary input system implemented; all that's needed now is to bring the model into the project and handle the responses. First, we need to export the model to the required format.

---

### 4.3.1 Import Model into Unity

In order to use the model in the Unity project, we need it to be in a single file format that is supported by the software; the format in question is *.onnx*, which is an open-source format that was built to encapsulate AI machine learning models, defining a set of common functions and operations that is required by such models. To convert my Keras model into this format, I used a public library called “*tf2onnx*”, which was created for this exact purpose.

First, I saved the recently-trained model using Tensorflow’s own function:

```
tf.keras.models.save_model(model, "/content/drive/MyDrive/finalModel")
```

**Figure 22 - Save Trained Model**

This saved the entire model to my Google Drive in Tensorflow’s *SavedModel* format, which is what I used this *tf2onnx* library for, to convert the model from this form to the required *onnx* format, using the code below:

```
# convert to ONNX
import tf2onnx

# load SavedModel formatted tf model
loaded_model = tf.keras.models.load_model("/content/drive/MyDrive/finalModel")

# save to a new onnx model - ignore second variable returned
# using a throwaway variable
onnx_model, _ = tf2onnx.convert.from_keras(loaded_model)
```

**Figure 23 - Exporting Model to ONNX**

I loaded the *SavedModel* model from my Drive to a local instance, then employed *tf2onnx*’s “*convert,from\_keras*” function to perform the conversion. Notice, in the last line of the figure above, where this operation takes place, I used a throwaway variable (‘\_’) alongside the “onnx\_model” variable to which this new *onnx* model will be saved. The reason for this is because the conversion function returns *two* values; the *onnx* model and an *external tensor storage dict*, which I did not need for this package, hence the throwaway variable.

So, the model is officially converted to the correct format, but it is still in a local variable, so we now need to save this to the Drive, in order to download it and import it to Unity.

```
# save new onnx model
onnx_model_path = "/content/drive/MyDrive/final_onnx/tfModel.onnx"
with open(onnx_model_path, "wb") as f:
    f.write(onnx_model.SerializeToString())

print(f"Model converted to ONNX and saved to {onnx_model_path}")
```

Model converted to ONNX and saved to /content/drive/MyDrive/final\_onnx/tfModel.onnx

**Figure 24 - Saving the ONNX to Drive**

By performing a simple write function as shown in the above figure, I saved the model into a new *onnx* file entitled “*tfModel.onnx*”. Once this operation was performed, I was able to locate the *onnx* file in my Google Drive and download it to my local machine. The worst of the work was now done, the last step is to bring this into Unity and use it.

#### 4.3.2 Run Inference using Barracuda Package

Unity has a package that simplifies the process of employing a new AI model inside a project, called “Barracuda”; once I imported the model into Unity, which was done like any asset inside Unity (through a simple drag-and-drop into the *Assets* folder, or otherwise), I created an empty *GameObject* entitled “*ModelManager*” and attached a new C# script with the same name, and it was ready to go!

The code to use the model through this Barracuda package was relatively simple, compared to the rest of the project; the general process was this: I loaded the model asset into the script as an “*NNModel*” (Barracuda’s model format), then created a new *IWorker* instance from this model, which will essentially be the object that executes the model’s shape detection, and finally, I program the worker to execute the detection whenever the *confirm* button is pressed.

The process of getting the assets’ instances is simple; we load the *NNModel* into a local “*Model*” object instance, then create the worker from this model using Barracuda’s ‘*WorkerFactory*’ class.

```
18 // load the AI model asset
19 m_runtimeModel = ModelLoader.Load(modelAsset);
20 if (m_runtimeModel != null)
21 {
22     // if the model is not null - set the worker object
23     worker = WorkerFactory.CreateWorker(WorkerFactory.Type.ComputePrecompiled, m_runtimeModel);
24 }
```

**Figure 25 - Load the Model Asset**

---

The new *IWorker* is now saved to this local “worker” variable. The worker is now ready to run inference with this model; this all happens within the “*RunInference*” method, which is called by the “*SaveDrawing.cs*” script when the Confirm button is pressed.

To execute this, I created a new instance of Barracuda’s “*Tensor*” class (using the *drawing* Texture2D data read in from the function in line 40 below).

```
39      // Run inference method
40      1 reference
41      public void RunInference(Texture2D drawing)
42      {
43          // send image into model for evaluation
44          Tensor inputTensor = new Tensor(drawing, 3);
45          worker.Execute(inputTensor);
```

**Figure 26 - Run Inference with Texture**

The input Tensor is created in line 43 above, followed immediately by a call to the “execute” function, which performs the detection on the drawing.

#### 4.3.3 Decipher Model Result

The model has now run the shape detection and determined the most likely shape visible in the image; we just need to find out which shape it detected. To do this, I created another Tensor instance, this time it will be the output tensor, which was gathered using the “*PeekOutput*” function as shown below.

```
46      // save the output texture and perform required action
47      Tensor outputTensor = worker.PeekOutput();
48      outputProcessor.HandleOutput(outputTensor.ArgMax()[0]);
```

**Figure 27 - Handle Result of Model**

This tensor now stores all the information required to complete the final stage of the project: output. As you can see in the figure above, I employed the services of the “*ArgMax*” function which returns the highest valued index in the output tensor (i.e. the index of the shape de-

tected). The index discovered is passed to this “*HandleOutput*” function found within the “ProcessOutput” C# script in the project, which I will now explain.

## 4.4 Output

As you may recall from Chapter 3’s sequence diagram, the indexes of the shapes are as follows: 0 for Circle, 1 for Square, 2 for Triangle; knowing this, handling the output becomes much easier, all we need to do is execute the correct function for whichever shape is detected.

All of the output handling is done inside the “ProcessOutput” script in the package; This script is a class that declares one method for each shape (*IsCircle*, *IsSquare*, *IsTriangle*), one of which is called depending on the index returned by the output tensor from the previous section. This happens inside the public “*HandleOutput*” function.

```
21 // call the respective method from the detected shape index
22 public void HandleOutput(int index)
23 {
24     // set the outputReturn variable to the current detected shape index
25     outputReturn = index;
26     // depending on the returned output - call the correct method
27     switch(outputReturn)
28     {
29         // detected shape was a circle
30         case 0:
31             IsCircle();
32             break;
```

**Figure 28 - Handling Output from Model**

The function has the index of the shape passed into it upon call (line 22 above), it saves this new index to the local “outputReturn” integer variable, and finally uses a simple switch statement to execute the respective function based on that index.

All the developer must do to when using this package is to define what functionality to execute when this shape is detected, i.e. for the *LearnToDraw* app, we could check if the user has drawn the correct shape:

---

```
49 // what to do when a circle is detected
50 1 reference
51 private void IsCircle()
52 {
53     // add your code here
54     // only if the drawn shape matches the shape selection can the player succeed
55     gameManager.HandleSuccess(ShapeSelection.Instance.selectedShape == ShapeType.CIRCLE);
}
```

**Figure 29 - Defining the Output Code**

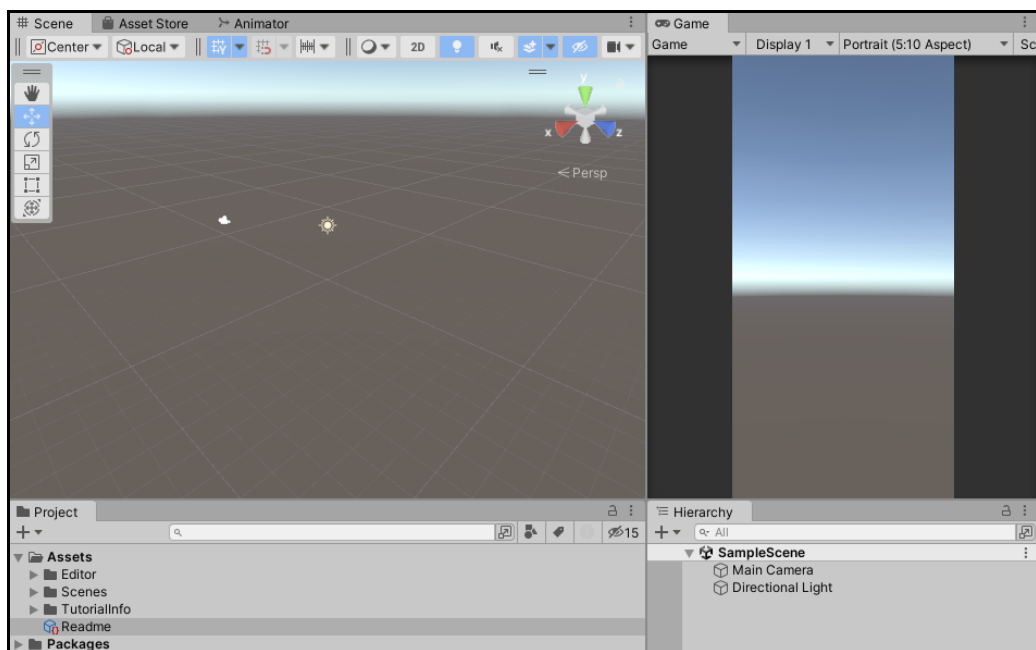
The example above simply checks if the selected shape instance matches the type of shape detected by the model (i.e. the user drew the correct shape). This is just one example of how the developer can use this output technique, there are countless ways to use this package, the possibilities are endless.

## 5 User manual

The *README.txt* file contains the information and steps necessary to use the package, but I will provide a more specific and visual example in this chapter.

### 5.1 Prerequisites

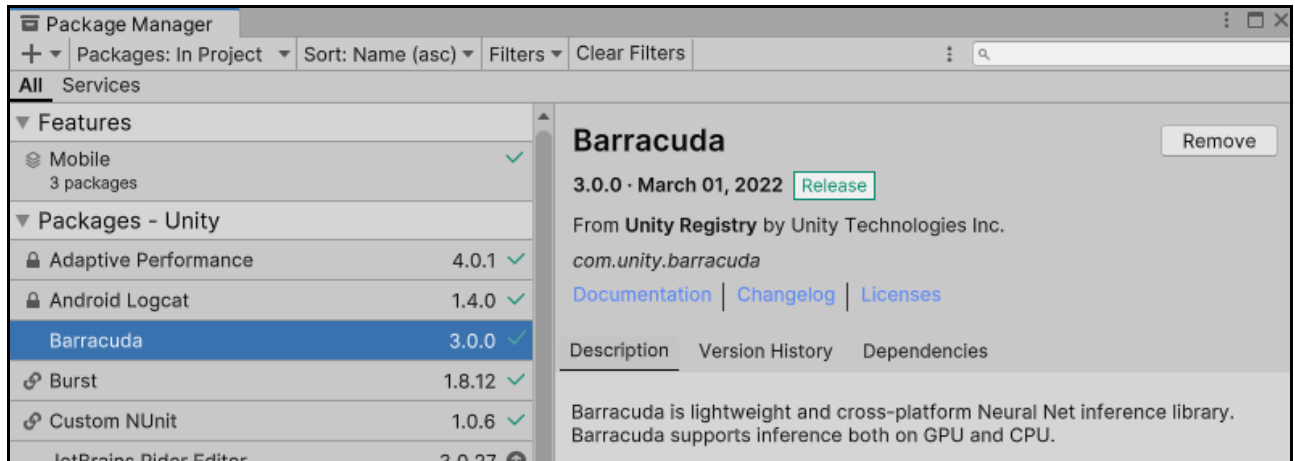
- Once the package is downloaded, there are few steps required to use it; firstly, you should have a new 3D Unity project created, with a scene opened and ready. The sample scene shown in the figure below is a 3D Mobile scene created using Unity 2022.3.20f1.



**Figure 30 - Creating a New Scene**

- The package utilises the *Barracuda* package which is accessible through the *Package Manager* in Unity (Window > Package Manager > (+) > Add package by name... > Name: com.unity.barracuda). **Note: Older versions of Unity do not have the “Add package by name...” feature, so you may need to download it to your disk from**

the official website or add the package by inserting an entry for “com.unity.barracuda” into your project’s *‘Packages/manifest.json’* file.



**Figure 31 - Unity Barracuda Package**

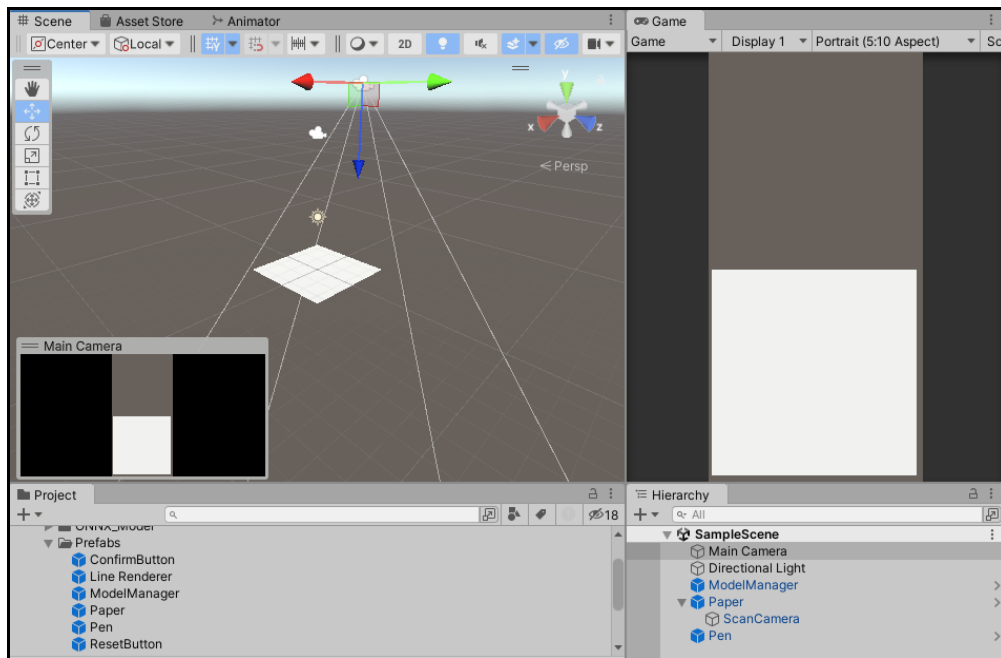
Once these two steps have been completed, the package is ready to import!

## 5.2 Setting Up the Scene

You can now import the package like any other by simply selecting “*Assets > Import Package > Custom Package...*” and selecting the doodle detection unity package file.

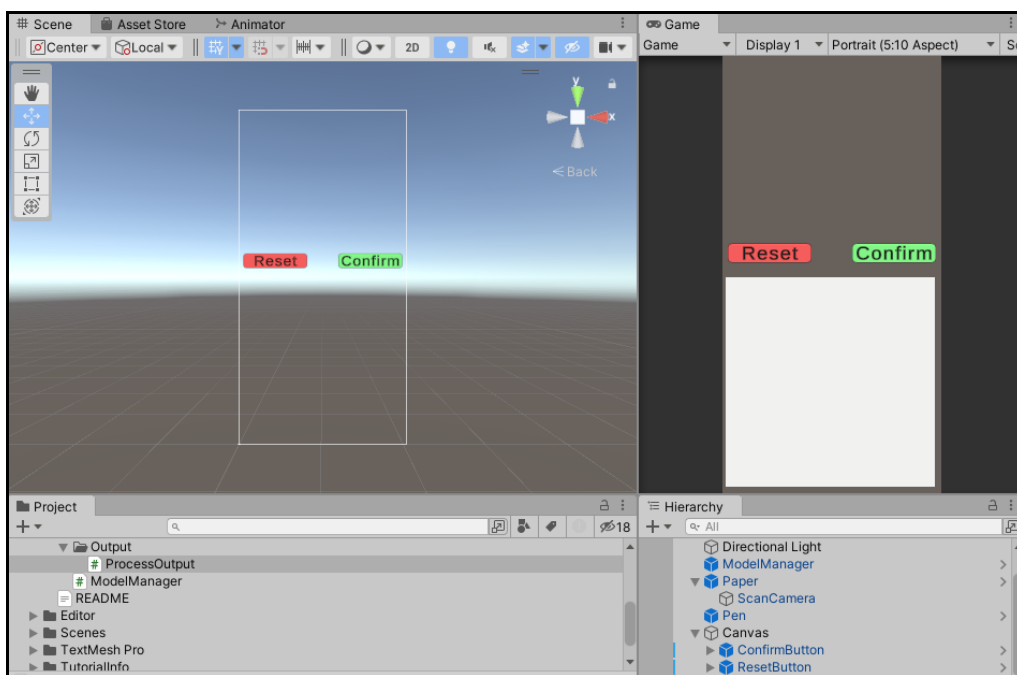
To begin using the package, you need to ensure the correct assets are inside the scene. Under the “Drawing Scene/Prefabs” folder, you’ll find a number of prefabs required for the system to function. Drag the “ModelManager”, “Paper” and “Pen” prefabs into the scene, ensuring the Paper object’s entire plane is visible by your scene’s Main Camera, as shown in the figure below (You will likely prefer to rotate the Main Camera so that it looks downwards onto the Paper, but it is your decision).





**Figure 32 - Adding Assets to Scene**

Finally, you'll need to add the *reset* and *confirm* buttons, so you can pass the drawing into the model automatically. To do this, simply add a blank Canvas object (or use an existing one, if you already have one in the scene) and drag these buttons onto the canvas so that they are also visible on-screen.



**Figure 33 - Place Assets On-Screen**

---

The scene is officially set up! Just ensure that all of the references inside the *Inspector* are correctly assigned, as some require Scene-specific references (**ConfirmButton** – Scan Camera & Model Manager; **ResetButton** – Pen).

### 5.3 Defining Output

Now that the scene is set up, you are free to create! In order to actually implement your own functionality, you just need to open the *ProcessOutput.cs* file inside the *Scripts* folder and edit the correct functions to execute your code! (*IsCircle*, *IsSquare*, *IsTriangle* – these methods will execute only when its relative shape is detected).

### 5.4 Training New Shapes

In order to train the AI model with new shapes, you will need to follow the steps outlined in the *Implementation* chapter. As a brief summary: gather your training data (images), separate these images into sub-folders with their respective classes as the folders' titles, upload these to a Google Drive account, mount this account to a *Google Colab* environment, follow the steps to build and export the model to the required ONNX format.

---

## 6 Critical analysis and Conclusions

The overall goal for this project was to create a package that made it very simple for a Unity developer to implement a Shape Detection system, with which they can have the user draw a shape onto the screen and return an output based on the shape detected; I feel as though I have done that. The project certainly did not come without its faults and errors, which delayed the implementation by days, if not weeks, but as an overall conclusion, I would confidently report that I am satisfied with the result.

Machine learning is not a subject that I had prior experience with, which meant I had to do a large amount of research into the areas that were specific to that topic; this meant I was undoubtedly setting high expectations for myself with regards to work ethic and adaptive capabilities. Having prior experience with Unity, I was able to adapt when certain issues were presented themselves and produce solutions to these issues relatively fast; this allowed me to keep on top of the project at a relatively steady pace.

With the complications experienced regarding the environment in which I would build and export the machine learning model, I was certainly set back a couple of days' work, which meant I had to take certain shortcuts to account for this lost time. In the end, using a Google Colab environment seemed to be the simplest method by which to train the model, so that was probably the biggest lesson that I learned from the assignment and something that I will definitely remember for my future programming work.

In the earlier builds of the model, I noticed a severe malfunction with the detection of the “square” class of shapes. It appeared during testing that most of the time a square was drawn, the system detected a triangle instead; of course, this was not acceptable and had to be rectified immediately before continuing to develop the project. I considered removing the square functionality altogether if the system continued to malfunction, but luckily after expanding

---

the square training images dataset, the model was able to detect them more accurately; a crisis delightfully averted.

After finishing the project, I concluded that my decision to employ an Agile methodology was entirely beneficial; having experience with such a structure before, I felt confident that I would be able to design the project in a way that I could build it brick-by-brick as it were, producing the project in increasingly high-quality releases. By defining the project as a set of features, and those features as sets of tasks, I was able to implement each task from highest to lowest priority, resulting in a product that I consider to be of decent quality.

In terms of future work for the project, I have a plethora of ideas. For starters, the model is currently limited to the three predefined shapes, but what I would love to implement in the near future is a way to allow the developer to automatically retrain and redeploy the model with their own set of training images – thus, increasing the number of shapes that can possibly be detected. There are numerous ways in which this could be done; after all, the bulk of the code is already available by way of the previous implementation chapter, but if I were to implement a way for the developer to simply execute a short series of commands and return a newly-trained model, the project would jump to a *Make-It-Best* version in an instant, so that’s certainly a strong goal for me.

Of course, no artificial intelligence model is perfect; every model will have some errors and fail at times, which means that every model can constantly be improved. I would like to do some more research into the layers implemented in the model and perhaps reconfigure it to produce better, more accurate results. I would likely do this by adding more layers, such as a Global Average Pooling (GAP) layer before the *flattening* stage.

In conclusion, there are certainly multiple avenues down which this package could go, each one improving upon it by increasing its accuracy or diversity, but overall, I am genuinely

---

pleased with the result of this project and I believe I have learned a lot of valuable lessons during the process. I have learned the effectiveness and benefits of utilising an online python environment to produce and train custom machine learning models; I have also learned a lot of information regarding the topic of machine learning as a whole, such as the different layers and stages of object detection, the types of pre-processing steps that can be implemented to improve the models accuracy, and how to implement such a model into Unity. The package has taken a lot of time and effort to produce, but I thoroughly believe that it would be of great use to those who wish to create applications and games that utilise *AI shape detection*, which was ultimately the goal of the project.

---

## 7 Bibliography

- Foo, W Y, W N Lim, and C S Lee. 2017. "Drawing Guessing Game for the Elderly." In *TENCON 2017 - 2017 IEEE Region 10 Conference*, 2236–41. <https://doi.org/10.1109/TENCON.2017.8228233>.
- Göksu, İdris, Alper Aslan, and Yiğit Emrah Turgut. 2020. "Evaluation of Mobile Games in the Context of Content: What Do Children Face When Playing Mobile Games?" *E-Learning and Digital Media* 17 (5). <https://doi.org/10.1177/2042753020936785>.
- Herusutopo, Antonius, Rizky Zuhurdin, Willy Wijaya, and Yuka Musiko. 2012. "RECOGNITION DESIGN OF LICENSE PLATE AND CAR TYPE USING TESSERACT OCR AND EmguCV." *CommIT (Communication and Information Technology) Journal* 6 (2). <https://doi.org/10.21512/commit.v6i2.573>.
- Magdin, Martin, Juraj Benc, Štefan Koprda, Zoltán Balogh, and Daniel Tuček. 2022. "Comparison of Multilayer Neural Network Models in Terms of Success of Classifications Based on EmguCV, ML.NET and Tensorflow.Net." *Applied Sciences (Switzerland)* 12 (8). <https://doi.org/10.3390/app12083730>.
- Moosa, Alaa Mohammed, Noor Al-Maadeed, Moutaz Saleh, Somaya Ali Al-Maadeed, and Jihad Mohamed Aljaam. 2020. "Designing a Mobile Serious Game for Raising Awareness of Diabetic Children." *IEEE Access* 8. <https://doi.org/10.1109/ACCESS.2020.3043840>.
- Mulhem, Ahmed Al, and Mohammed Amin Almaiah. 2021. "A Conceptual Model to Investigate the Role of Mobile Game Applications in Education during the COVID-19 Pandemic." *Electronics (Switzerland)* 10 (17). <https://doi.org/10.3390/electronics10172106>.
- Mun, Il Bong. 2022. "A Longitudinal Study on the Effects of Parental Anxiety on Mobile Game Addiction in Adolescents: The Mediating Role of Adolescent Anxiety and Lone-

- 
- liness.” *International Journal of Mental Health and Addiction*.  
<https://doi.org/10.1007/s11469-022-00890-2>.
- Nisiotis, Louis. 2021. “Utilising Mobile Game Based Learning Methods Effectively to Support Education.” *Educational Technology Research and Development* 69 (1).  
<https://doi.org/10.1007/s11423-020-09887-x>.
- Su, Fan, Di Zou, Haoran Xie, and Fu Lee Wang. 2021. “A Comparative Review of Mobile and Non-Mobile Games for Language Learning.” *SAGE Open* 11 (4).  
<https://doi.org/10.1177/21582440211067247>.
- Tsai, Tsung Han, Po Ting Chi, and Kuo Hsing Cheng. 2019. “A Sketch Classifier Technique with Deep Learning Models Realized in an Embedded System.” In *Proceedings - 2019 22nd International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2019*. <https://doi.org/10.1109/DDECS.2019.8724656>.
- Unibas-Markaida, Iratxe, and Ioseba Iraurgi. 2021. “Commercial Videogames in Stroke Rehabilitation: Systematic Review and Meta-Analysis,” June.  
<https://doi.org/10.1080/10749357.2021.1943798>.
- Wang, Jin Liang, Jia Rong Sheng, and Hai Zhen Wang. 2019. “The Association between Mobile Game Addiction and Depression, Social Anxiety, and Loneliness.” *Frontiers in Public Health* 7 (SEP). <https://doi.org/10.3389/fpubh.2019.00247>.
- Wazir, Waqas, Hasan Ali Khattak, Ahmad Almogren, Mudassar Ali Khan, and Ikram Ud Din. 2020. “Doodle-Based Authentication Technique Using Augmented Reality.” *IEEE Access* 8. <https://doi.org/10.1109/ACCESS.2019.2963543>.
- Yadav, Savita, and Pinaki Chakraborty. 2017. “Children Aged Two to Four Are Able to Scribble and Draw Using a Smartphone App.” *Acta Paediatrica* 106 (6): 991–94.  
<https://doi.org/10.1111/apa.13818>.

---

## 8 References

- Foo, W Y, W N Lim, and C S Lee. 2017. "Drawing Guessing Game for the Elderly." In *TENCON 2017 - 2017 IEEE Region 10 Conference*, 2236–41. <https://doi.org/10.1109/TENCON.2017.8228233>.
- Göksu, İdris, Alper Aslan, and Yiğit Emrah Turgut. 2020. "Evaluation of Mobile Games in the Context of Content: What Do Children Face When Playing Mobile Games?" *E-Learning and Digital Media* 17 (5). <https://doi.org/10.1177/2042753020936785>.
- Herusutopo, Antonius, Rizky Zuhudin, Willy Wijaya, and Yuka Musiko. 2012. "RECOGNITION DESIGN OF LICENSE PLATE AND CAR TYPE USING TESSERACT OCR AND EmguCV." *CommIT (Communication and Information Technology) Journal* 6 (2). <https://doi.org/10.21512/commit.v6i2.573>.
- Magdin, Martin, Juraj Benc, Štefan Koprda, Zoltán Balogh, and Daniel Tuček. 2022. "Comparison of Multilayer Neural Network Models in Terms of Success of Classifications Based on EmguCV, ML.NET and Tensorflow.Net." *Applied Sciences (Switzerland)* 12 (8). <https://doi.org/10.3390/app12083730>.
- Moosa, Alaa Mohammed, Noor Al-Maadeed, Moutaz Saleh, Somaya Ali Al-Maadeed, and Jihad Mohamed Aljaam. 2020. "Designing a Mobile Serious Game for Raising Awareness of Diabetic Children." *IEEE Access* 8. <https://doi.org/10.1109/ACCESS.2020.3043840>.
- Mulhem, Ahmed Al, and Mohammed Amin Almaiah. 2021. "A Conceptual Model to Investigate the Role of Mobile Game Applications in Education during the COVID-19 Pandemic." *Electronics (Switzerland)* 10 (17). <https://doi.org/10.3390/electronics10172106>.
- Mun, Il Bong. 2022. "A Longitudinal Study on the Effects of Parental Anxiety on Mobile Game Addiction in Adolescents: The Mediating Role of Adolescent Anxiety and Lone-



- 
- liness.” *International Journal of Mental Health and Addiction*.  
<https://doi.org/10.1007/s11469-022-00890-2>.
- Nisiotis, Louis. 2021. “Utilising Mobile Game Based Learning Methods Effectively to Support Education.” *Educational Technology Research and Development* 69 (1).  
<https://doi.org/10.1007/s11423-020-09887-x>.
- Su, Fan, Di Zou, Haoran Xie, and Fu Lee Wang. 2021. “A Comparative Review of Mobile and Non-Mobile Games for Language Learning.” *SAGE Open* 11 (4).  
<https://doi.org/10.1177/21582440211067247>.
- Tsai, Tsung Han, Po Ting Chi, and Kuo Hsing Cheng. 2019. “A Sketch Classifier Technique with Deep Learning Models Realized in an Embedded System.” In *Proceedings - 2019 22nd International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2019*. <https://doi.org/10.1109/DDECS.2019.8724656>.
- Unibas-Markaida, Iratxe, and Ioseba Iraurgi. 2021. “Commercial Videogames in Stroke Rehabilitation: Systematic Review and Meta-Analysis,” June.  
<https://doi.org/10.1080/10749357.2021.1943798>.
- Wang, Jin Liang, Jia Rong Sheng, and Hai Zhen Wang. 2019. “The Association between Mobile Game Addiction and Depression, Social Anxiety, and Loneliness.” *Frontiers in Public Health* 7 (SEP). <https://doi.org/10.3389/fpubh.2019.00247>.
- Wazir, Waqas, Hasan Ali Khattak, Ahmad Almogren, Mudassar Ali Khan, and Ikram Ud Din. 2020. “Doodle-Based Authentication Technique Using Augmented Reality.” *IEEE Access* 8. <https://doi.org/10.1109/ACCESS.2019.2963543>.
- Yadav, Savita, and Pinaki Chakraborty. 2017. “Children Aged Two to Four Are Able to Scribble and Draw Using a Smartphone App.” *Acta Paediatrica* 106 (6): 991–94.  
<https://doi.org/10.1111/apa.13818>.