MIDDLESEX UNIVERSITY

FINAL YEAR PROJECT

REPORT

Abdur Rehman Nasir

Machine Perception: Investigating Visual Data Analysis through CNN Hand Gesture Recognition

---

Supervisor:
Dr. Clifford DeRaffaele

21 June, 2024

Abstract

This dissertation investigates the enhancement of human-computer interaction through hand gesture recognition in a sandbox Unity application for visual data analysis. The prototype application aims to replace traditional methods of visual data analysis with intuitive hand-gesture controls, visualizing datasets as expansible stars in a 3D space. The study covers dataset management, machine learning techniques, and Unity application development, highlighting iterative improvements and key findings, alongside a brief introduction to what makes a gesture intuitive and why this dissertation retains value in the current research climate.

Table of Contents

1 Project Description

The field of visual data analysis via human-computer interaction (HCI) has remained commercially stagnant since the genesis of Virtual Reality (VR) and Augmented Reality (AR). Advancements in various computer science disciplines since, have re-ignited interest in this field. Progression has been limited to academic and developmental scopes, where considerable improvements in brain-computer interfaces (BCI) and rich 3D visualisations edge us towards a more intuitive control over ever-more powerful machines. This project aims to prototype a refinement to existing human-computer interaction through the use of gestures within a novel Unity application. The prototype proposed will conform to the growing reliance on ease-of-accessibility and convenience to provide intuitive options for gesture-based interaction that may even take precedence over the default keyboard and mouse input method. The resulting application aims to natively and intuitively navigate a native Windows or Linux desktop, and a chosen sandbox application. Despite a lack of steely clarity, the decidedly cautious nature of this proposal aims to allow modifications to the project wherever required as the period for implementation nears. As a whole, this paper intends to develop an easily implementable, intuitive gesture-recognition software to assume hand-gestural control of a dataset, and allow the means for basic data analysis without the long-standing necessity of a conventional keyboard and mouse.

2 Steps of the Research Methodology

The 10 steps to achieve the final outcome of hand-gesture manipulation of a dataset within a standalone application are summarised as follows in sequential order:

Step 1: Problem Description

Step 2: Literature Review

Step 3: Included Fields

Step 4: Dataset Acquirement

Step 5: Standalone Application Development

Step 6: Data Preprocessing

Step 7: Model Training

Step 8: Classifier Algorithm Acquirement*

Step 9: Gesture Mapping

Step 10: Discussion of Performance

**This was instead developed due to the lack of available classifiers for the niche of this endeavour.*

2 Problem Solved

"With the ever-increasing diffusion of computers into the society, it is widely believed that present popular mode of interactions with computers (mouse and keyboard) will become a bottleneck in the effective utilization of information flow between the computers and the human."
- Murthy and Jadon, 2009

There is a niche availability for research within the scope of data analysis undertaken exclusively through hand gestures. While hand gesturing borders a state-of-the-art technology at the forefront of select lines of funding, there remains extraordinarily few opportunities to visually assess datasets within a dedicated environment. The main jurisdictions within which this project seeks to establish itself work in tandem with this purpose, in order to deliver the exclusivity of the environment sought in this project. Their interaction is outlined in *Figure 1* as a project breakdown.

Within *Figure 1*, the training set is handed to the classifier algorithm for a certain number of epochs (iterations), such that a hypothesis is able to be formed for any new data-point that enters the algorithm to be classified. This is then mapped across as a set of actuations within the stand-alone application as a method of interaction. These areas are detailed later in the document.

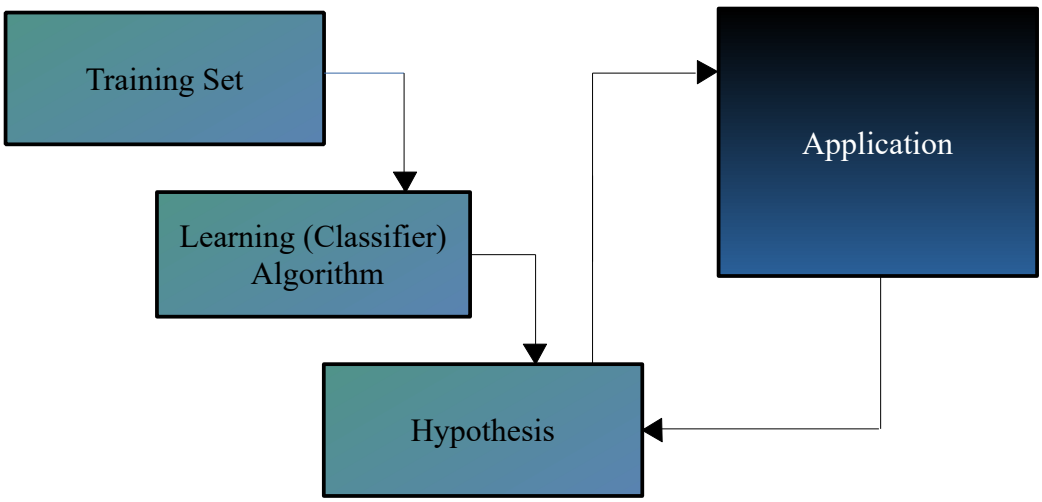


Figure 1: Project Breakdown

The methodology behind the classifier algorithm is outlined in *Section 8*.

Mastering the end-to-end connectivity between the classifier algorithm and the hand gesture mappings within the stand-alone application, such that intuitiveness becomes a symptom of a mathematically seamless pairing is a strong underlying goal within this endeavour. The resulting application purports to provide fertile grounds for the analysis of multi-variate datasets using hand-gesture recognition.

3 Literature Review

3.1 Survey Methodology

I review the most significant research papers in the field published from 2018 onwards (~5 years), with the exception of a few groundbreaking papers from before this period that continue to find use. I focused on reviewing papers from Elsevier, ScienceDirect, Springer and ArXiv. My search criteria varied, starting from a narrower focus on the specific topic of the paper (which concurrently returned zero results), to detailed exploration of much more general keywords. Some notable examples include ("Hand Gesture Recognition Visual Data Analysis"), ("Hand Gesture Recognition Application"), ("HGR HCI"), and more.

3.2 Literature Status

This report aims towards the niche and yet unperturbed section of intuitive HGR (Hand Gesture Recognition) in visual data analysis, aided by the creation of a stand-alone application that presents astral-themed multivariate datasets for such. The specificity of this integration lies in the selection of an appropriate dataset of notable intuitive classes, and classifier algorithm, and training these on a machine learning model, and integrating the hypothesis generation with interactivity in the 3D space. Throughout the literature review, and the development of this dissertation in its entirety I confirm that the concept of this specific idea remains novelty, and yet undiscussed. This includes the lack of a suitable dynamic classifier algorithm, discussed further in *Section 8.10.*

Despite finding footholds in the forefront of HCI (Human-Computer Interaction), research within HGR systems to support further development remains relatively stagnant and plagued with unremarkable accuracy (Mohamed, Mustafa and Jomhari, 2021). However, extraordinary recent advances in the machine/deep learning communities have ignited great potential within the realm of HGR systems within HCI. A multitude of studies within this field leverage the dedicated hardware of the Microsoft Kinect to make algorithmic leaps, often using a combination of CNNs (Convolutional Neural Networks) to greatly enhance the HGR systems to command over 90% accuracy (Devineau *et al.,* 2018, Ma and Peng, 2018). The Kinect was discontinued in 2017, but still finds academic use in relevant studies within the recent years (Cerfoglio *et al.,* 2022), suggesting dedicated commercial funding is likely uninvolved in this sector. With the publication of Tensorflow as an open-source library for machine learning in 2015, more options became available for research to take place.

3.3 Landmark Papers

Configured within the hardware native of SNNs (Spiking Neural Nets), and software conformation to CNNs, this paper achieves astounding accuracy in HGR (Amir *et al.,* 2017), while weaving in the cognitive neurosciences in the format of low-power and parallel processing. The novel creation of

their own dataset is also referenced in the interim report, in *Section 4.1.2* within the paper and remains one of the inspiring components of this project. Perhaps the most promising research of outstanding notoriety among such a small niche is that of Amira – a highly interactive system for visual data analysis (Stalling, Westerhoff and Hege, 2005). Despite being released nearly two decades prior, the incredible quality of this paper finds no notable equivalent in the field of HGR + HCI, and continues to be cited into 2024 (16 citations at time of submission), and finds 148 citations since 2020 alone.

The relevance of Amira to this literature review occurs in **Section 4**, **page 13**, where the extension amiraVR takes precedence for a brief few pages. While this doesn't connote visual data analysis directly, VR was one of the options considered during the proposal of this project as a viable method for visual data analysis – only abandoned due to developmental complexity outside the scope of an undergraduate dissertation module. The 3D space within Amira allows for cross-sectional analysis, rotation and transformation to significant effect, and filters of various types that allow the deconstruction of objects rendered within its space. A branch of note that I had not previously traversed, but remains highly relevant to my own work follows in the form of event tracking. Every mouse event within Amira causes visual feedback, which I overlooked when it came to the genesis of my own project, and will likely be an addition in the application I create.

3.3 Intuitive Gesturing

This remains one of the most important components of a coherent HGR system, and the classes within the dataset chosen have been *specifically* chosen and explained in accordance with the concepts of intuitive HCI. On a broader philosophical scope, the intuition of gestural interactivity both with machine and other humans correlates far more strongly to the physical motion (the *how*), not the result (the *what*) nor the goal (the *why*) (Thioux, Gazzola and Keysers, 2008). This suggests a high degree of importance in the correct hand-held visualisation that the user is directly interacting with the application, its nodes and data as if they actually existed. Correspondingly, high visuospatial availability can boost the thinking abilities of an individual, when rendered under the context of hand gestures (Hyusein and Goksün*.,* 2023). A relevant reflection is found in VR applications, whereby traditional interactive devices such as a mouse and keyboard greatly reduce the immersion of users when compared to the more modern methods of interaction, such as dedicated VR controllers, hand gestures etc. (Zhang, Zhu and Zhu, 2019).

In summation, the gestures that produce greater intuitiveness differ within the context of the environments they claim availability within (Ophelia and Keerthijith, 2018). When considered under the viewpoint of the physical motion (the *how*), binary devices in the vein of motion sensors, FSA (Finite State Automata) etc. induces greater intuitiveness with static gesturing adherent to the simplicity of the state change, as opposed to the abject near-necessity of dynamism with manipulations in 3D space (whether real or virtual) (Gao *et al.*, 2021, Chang *et al.*, 2023).

4 Dependencies

In order to host an intuitive system of gesture recognition, certain hardware and software properties need to be meshed comprehensively into a suitable environment for demonstration and development.

While the dependencies required seemed straightforward early in pursuit, the constantly shifting developmental requirements at the stage of implementation led to exploration of a variety of libraries. The final environment is listed in this section, with an insight into the steps before arriving at this conclusion, outlined in *Section 10.*

4.1 Virtual Machine (VirtualBox 7.0.12)

In order to keep reproducibility high, and results consistent, I introduce all following software dependencies within a dedicated virtual machine hosted on a Linux operating system – detailed further in *Section 3.2*.

To accommodate the excessive size of the dataset – detailed in *Section 4.2* – I allocated extensive machine resources to the virtual machine. At this point, I realised I could've opted for a 'dual boot'-style of environment, but decided against it in preference of the 'Snapshot' feature of a virtual machine, which ensures reliability and backup potential as opposed to relying on the repeatability of booting a hard drive on double operating systems.

4.2 Operating System (Windows 10)

Initially, I opted out of a Windows environment in favour of Ubuntu 20.04, mainly under the perception that the open-source and academic basis of this endeavour would require many integrated dependencies interwoven solely through the command line. This turned out to be the case for both Tensorflow and OpenCV, which by default prefer Linux-based development.

Another strength of Linux lay in its ease of access to multiple applications consecutively, without having to deal with cluttered GUIs and therefore dependence on resource allocation. Command line-based navigation is favoured in Linux. For example, the AWS command line interface package (*Section 3.6*) required only 4 commands within the terminal to download over 260GB of dataset into my directory.

However, as the project progressed, I discovered that there was limited compatibility between my choices of software environment and the circumstance of a virtual machine hosting Linux. This is explained in detail in *Section 4.2,* which led to my current and final choice of Windows 10.

4.3 Programming Language (Python 3.9)

While Python is a language I have little to no experience with, it's apparent that it is held in extremely high esteem within machine learning communities (Rayhan & Gross, 2023), and hosts a significant number of relevant machine learning libraries, granting it incredible credence in solving specific problems in these areas. On specific offer in the Linux library is the ability to create a virtual environment to host all these specific dependencies, using a '*venv'* command.

The culmination of these strengths ushered me towards the view that spending time on learning this would not only benefit the purposes of this project, but also any further academic endeavour beyond the undergraduate dissertation.

4.4 Machine Learning (TensorFlow 2.10)

Both the TensorFlow package itself, and any dependencies required to integrate with my virtual environment proved easy to install. An introduction was given in the form of an interactive web page, linked to a cloud-hosted GPU processing service, where a reasonable extent of tutorials and demonstrations were presented. This service is termed Google Colab, and formed a larger part of my tests with TensorFlow.

Additionally, a large number of hand-gesture recognition projects were discovered with TensorFlow as the underlying machine learning library during the literature review.

4.5 Video Processing (OpenCV 4.7.0)

Intended for real-time computer vision, this library promises to work interpersonally with the machine learning component of this endeavour. The Tensorflow model is intended to be trained on the dataset such that customised hand recognition for the classes chosen is possible. A Logitech C920S webcam was purchased for the purposes of this project.

4.6 AWS-CLI (Amazon Web Services Command Line Interface)

This package became a necessity for the retrieval of the IEEE-Dataport dataset since its size was offloaded to the AWS S3 servers for storage. The simplest method to retrieve this dataset was made available to me by the choice of Ubuntu as an operating system. In all, one line of installation code, only two lines of configuration code and one line of execution code was all I required for the download of a >260GB dataset.

4.7 Git (Version Control System)

Git was used for the majority of the written work in this project, providing a consistent and trackable workflow across its duration. Screenshots are given in *Appendix C* at the end of the document.

Git was not used for the classifier algorithm post-transfer to Windows (*Section 4.2*), or the Unity project (*Section 7*).

5 Dataset

This project requires an intuitive, dynamic set of hand gestures utilising one-handed gesturing within the scope of visual data analysis. Within this conceptualisation, a prospective user should be able to *comfortably* manipulate data-points within a multivariate dataset created in a 3D space, with the ideal aesthetic of each data point resembling a star in a cluster (collection).

5.1 Available Choices

Among the swathes of open-source and free datasets, availability in this specific niche remains markedly limited. Within this scope, I reviewed some of the most popular datasets that find general usage within the genre, and narrowed down to some final quality picks from which to purchase the data which most closely fit this endeavour.

5.1.1 HaGRID

Containing over 550,000 samples of hand gestures over only 18 classes, this is easily one of the most comprehensive datasets within the topic. In addition, the main focus of the dataset lies within the usage of these gestures to manage devices under the umbrella of HGR (Hand Gesture Recognition) systems (Kapitanov, Makhlyarchuk and Kvanchiani, 2022). Despite being almost 4x larger than the final choice reviewed in **Section 4.2**, and consistently updated (with the previous being a few months before the creation of this document – Sep 2023), this proposes static hand gesture recognition instead of the required dynamism. I review the differences more scrutinously in the Literature Review**.**

While this remained an incredibly strong contender for the final dataset till the very final stages of planning, the final choice won over by a small margin.

5.1.2 DvsGesture

This remained another strong contender for the final dataset, introduced as an original dataset paired with a state-of-the-art SNN (Spiking Neural Nets) algorithm with an astounding 96%+ accuracy (Amir *et al.,* 2017). Over 330 citations lend great credence to the quality of the dataset.

However, the dataset failed to meet the criteria of this dissertation within its format as static hand gestures as opposed to introducing the dynamism required for intuitive interaction **(**Gao *et al.*, 2021).

5.1.3 IPN Hand

With over 4000 gesture samples and 800,000 RGB frames from 50 distinct subjects, and provisioned as a ‘tough’ benchmark for a state-of-the-art ResNext-101 algorithm (Benitez-Garcia *et al.,* 2020), this dataset was yet again another contender for the final choice. I contemplated the strengths of this dataset at many points, especially considering the fact that dynamism was guaranteed alongside some sparse static gestures, which might’ve allowed for desktop integration in an earlier iteration of the project **(Section 6.1)**.

However, I felt that in this particular instance the gestures themselves did not resemble the intuitiveness I sought, or were irrelevant for the purpose of this endeavour. For example, class B0A is entirely unintuitive, with multiple stages of the movement as opposed to one smooth motion, while G03 to G06 *(Figure 2)* find minimal relevance for data analysis. Out of a meagre 14 gestures, this already purports a large fraction to be outside the purposes I seek them for.

6	G03	Throw up	200	62 (25)	
7	G04	Throw down	201	65 (28)	
8	G05	Throw left	200	66 (27)	
9	G06	Throw right	200	64 (28)	

Figure 2: IPN Dataset Classes G03 - G06

5.1.4 HANDS

The gestures themselves are extremely well sought, using wide angles and full HD resolution and 300 frames per gesture. The article introducing them is well-presented, and are intuitive enough to find usage in most general applications, including within the scope of visual data analysis. Some gestures highlighted in *Figure 3* are both unique and intuitive in specific application.

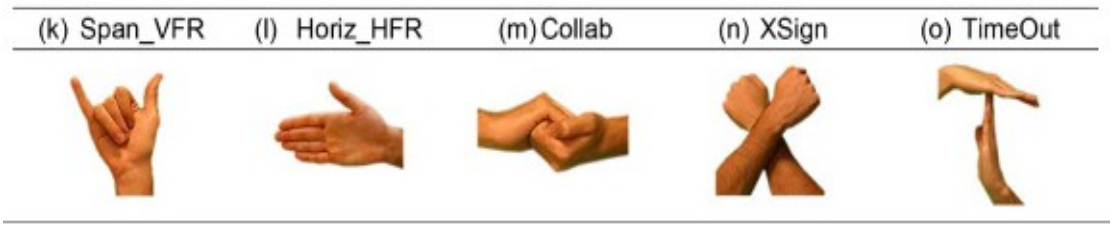


Figure 3: HANDS Dataset Classes (k - o)

However, the same issue of static hand gesturing limits the likelihood of usage. These find more application in industrial contexts as opposed to the dynamism sought for this project (Nuzzi *et al.,* 2021).

5.2 Chosen

Considering the requirements of this endeavour, I will split this section into further subsections such that the process of consideration is revealed with notable clarity.

5.2.2 Dataset for Dynamic HGR Systems

This unnamed 27-class dataset provides ample choice for intuitive dynamic hand gesturing. Mostly sourced from the popular NVIDIA dataset (Molchanov *et al.*, 2016), which was in turn sectioned and compiled from existing commercial systems or datasets, this was chosen for its extraordinary comprehensiveness, and generous offering in terms of uniquely intuitive motions within gesturing.

This dataset features:

- 1701 videos
- 204,120 frames
- 27 dynamic classes
- 21 different subjects
- 3 videos per class
- Quality variability (320x420, 640x480, 1920x1080)

No other dataset I reviewed offered as much relevance to this endeavour. Within it, I found the ideal conditions for a strong and notably intuitive hand gesture recognition system. While all 27 classes were robust, when considered against both the objective and the astral theme, only 11 were chosen for the purposes of this dissertation.

The breakdown of these 11 gestures within the scope of the application within which they are intended to be added, and the datasets which they intend to analyse and superficially manipulate, are detailed in *Section 7.7.*

6 Machine Learning

As mentioned in *Section 3.4*, machine learning is a vital component of this endeavour. The chosen environment isn't as beginner-friendly as was suggested before the deep-dive preceding this progress report. However, the field itself is proving in my limited view of it to be almost infinitely scalable, with complexities available proportionate to any skill level.

6.1 Classifier Algorithm

Alongside a variety of more minor but relevant content, I surmised that an introduction into AI algorithms would not only benefit my understanding of the underpinnings of a classifier algorithm and the subsequent HGR (Hand Gesture Recognition) system, but also reveal to me some greater concepts about Tensorflow, OpenCV and the integrations behind it all. As such, I chose to study Stanford CS229: Machine Learning as part of the undertakings of this module. This helped extraordinarily with the novel classifier algorithm development in *Section 7.*

6.1.1 Classification vs Regression

In *Figure 1*, the project breakdown can be visualised with some clarity. Within this, at an intermediate stage between the training set and hypothesis, lies the classifier algorithm. Classification is a task that requires the use of machine learning algorithms to sort data into classes, or clusters (which finds some relevance in my application theme).

Simply, regression predicts analogue, or continuous outputs, while the binary outputs are handled by a classification categorisation. While both generally work in probabilities, the former is bounded

by a range in which a “hard margin” sits centrally, dividing existing data-points, and any new data-points into a correct mapping. The latter instead relies on probabilities to enter classification boundaries, where at a very basic level, if the likelihood of a new data-point belonging to a certain category is highest, it will be classified as such thenceforth. As such, this endeavour is a *classification problem.* The output is binary as opposed to continuous.

While the optimal classification method remains a mystery at this point, the topic as a whole is discussed at some length in the Literature Review, alongside the visual data analysis component.

6.1.2 Hypothesis and Cost Function

The simplest problem in machine learning is linear regression, but in any case, the job of any learning algorithm is to:

- Input a training set.
- Output a hypothesis
  - Input a new data-point.
  - Output a learned prediction.

(Ng, 2018)

The hypothesis can be modelled mathematically as follows:

$$h(x) = \theta_0 + \theta_1 x$$

Immediately, we find similarity with the general formula of a straight line graph y = mx + b – thereby reinforcing the implication of a “hard margin”. As this evolves to a parabola or further to meet the specification of complex regression problems, this basic hypothesis can evolve to polynomial or even differential functions.

Upgrading this to a summation series allows for a more condensed visual that can be stated as a general formula for a hypothesis.

$$h(x) = \sum_{j=0}^{n} \theta_j x_j$$

In contrast to before, this summation series accepts terms up till n, theoretically allowing for infinite parameters (akin to a Support Vector Machine), where the initial case of j = 0 precedes an operation that stays in line with the original hypothesis function.

$$where\, x_0 = 1$$
$$\theta_0 x_0 = \theta_0$$

Then, choosing θ such that h(x) is close to y is the next step in determining the highest likelihood of correct hypothesis among any single (i'th) data-point, represented by x and y, where x is the parameter and y is the "output", or target variable.

$$(h(x) - y)^2 \qquad\qquad (x^i, y^i)$$

Such that the closest answer is closest to zero. The combination of these factors, alongside some niche-specific terminology, culminates in the cost function for linear regression.

$$j(x) = \frac{1}{2}\sum_{i=1}^{m} h((x^i) - y^i)^2$$

Where m = number of training examples.

From which we can begin to find understanding in the cost function for classification, within logistic regression.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\Big(h_\theta(x^{(i)})\Big) + (1-y^{(i)})\log\Big(1-h_\theta(x^{(i)})\Big)\right]$$

This cost function effectively measures the penalty for incorrect predictions in classification problems, with the penalty increasing as the predicted probability diverges more from the actual "output", or target. As part of my next steps, detailed in *Section 9*, I wish to go into specific detail within this, and begin to understand the mathematical processes behind the algorithm such that both any optimisation it may require and the subsequent (or parallel) report-writing becomes second nature.

6.1.3 3D Convolutional Neural Network

Despite its relative complexity, a CNN (Convolutional Neural Network) is a staple of HGR systems. As an example within this document, all but one of the datasets mentioned in **Section 4.1** (the DvsGesture dataset, which uses Spiking Neural Nets) have implementations in CNNs. These find themselves in the top ranks of image classification algorithms (Sharma, Jain and Mishra, 2018), with almost perfect accuracy (99%+) under optimised conditions (Tan, Lim and Lee, 2021).
As such, this remains the proposed algorithm for this endeavour.

7 Standalone Application Development

The intended application was to be developed in the Unity game engine, within which an uploaded dataset could be analysed as a star object in an astral-themed environment. This called for a completely novel application to be developed for the sole purpose of this dissertation, aptly named ‘DataStellar’.

Considering the learning curve of the Unity game engine, several steps were required in order to have a successful standalone application available to integrate with the classifier algorithm, and summarised as follows:

1. Player
2. Viewport Control
3. Terrain
4. Skybox
5. Boundary
6. 3D object(s)
7. Conventional Interaction
8. Gesture-Based Interaction

In all, 5 hours 57 minutes of intensive Unity Engine footage was recorded over the weeks of the development process, minus unrecorded work, conceptual design, research, video tutorial use and design retractions, and a healthy estimate would assume 50-60 hours. This was recorded for archival purposes, and would have been possible to use as a timelapse in-presentation.

7.1 Project Setup

The project was created on a 3D URP to maximise the processing power provided by the PC, while also keeping graphical quality high in line with the status quo for current developments. Research on video tutorials guided a path to the style of development I preferred for the project, helped familiarisation with the IDE and provided useful information on how to instantiate each concept.

7.2 Player

This game object contains three sub-objects, which together define the player’s movement and the position of the viewport in 3D space using the ‘Cylinder’, ‘Main Camera’ and ‘GroundCheck’ objects.

A ‘Cylinder’ object was added to emulate a person’s shape while on the default plane rendered by the template. The height of the cylinder was doubled to 2 units shortly after addition. This was to keep an illusion of size when perspective was considered against the terrain, while the incomparable gigantism of the star objects (to be added later) imbued a sense of negligibility and insignificance to the player, as the viewing of interstellar bodies would in real life.

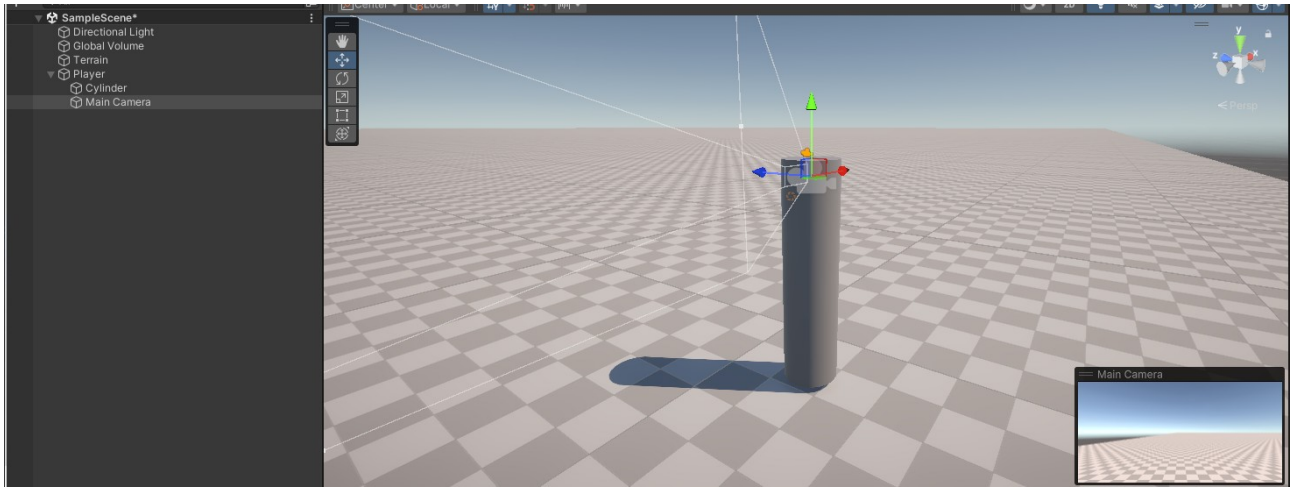


Figure 2: Camera Positioning

As shown in *Figure 2,* 'Main Camera' was placed at the top of the 'Cylinder' object, in order to reinforce the illusion of a human perspective, and allow intuitive control of the 'Player' object synonymous with real life. The importance of this placement ties into the intuitive theme, whereby perspectives shifted from normal viewing perspective require time to adjust as opposed to the instant familiarity with a known perspective (He et al., 2022), meaning that users could instantly begin with no perspective-based learning curve.

7.3 Movement Scripts

Two scripts were used for the purposes of player and camera control. The Unity Engine generated a boilerplate class structure template for both.

7.3.1 PlayerMovement.cs

Detailed in *Figure* 3 are the physics parameters that are imposed upon the 'Cylinder' object controlled by the user. All values are arbitrary except gravity. This was intentionally set to -1.625, or 0.16g to reflect the gravity on the moon.

However, dependent on the conditions of the planet, physics, speed and even the ability to jump could be significantly altered to provide a more immersive experience – which falls out of the current scope of the endeavour.

This is followed by the Update() method shown in *Figure 4.* This is called once per frame, and works as a contact check between the 'Cylinder' object and the ground. As seen in *Figure 2,* the 'Cylinder' object's height off the ground is indistinguishable from contact.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
    0 references
5   public class PlayerMovement : MonoBehaviour
6   {
        2 references
7       public CharacterController controller;
8
        1 reference
9       public float speed = 12f;
        2 references
10      public float gravity = -1.625f * 2;
        1 reference
11      public float jumpHeight = 3f;
12
        1 reference
13      public Transform groundCheck;
        1 reference
14      public float groundDistance = 0.4f;
        1 reference
15      public LayerMask groundMask;
16
        5 references
17      Vector3 velocity;
18
        3 references
19      bool isGrounded;
```

Figure 3: Player Object Physics Rules

However, in this case the height off the ground is set at an arbitrary 0.4, as shown in the 'groundDistance' variable in *Figure 3.*

Additionally, line 23 resets falling velocity on contact with the ground, the method starting line 41 both defines the 'jump' movement and checks whether its possible (player contact with ground). The input controls are natively integrated into the Unity Engine and are changeable using the Input Manager, accessible through:

Edit → Project Settings → Input Manager.

In all, this code documents allows omnidirectional user movement throughout the terrain, which may seem extraneous and unnecessary, but when considering a likely use case where several datasets are required, this would allow the user to explore a larger area of terrain, and therefore several more stars.

```
0 references
21  void Update()
22  {
23      isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance, groundMask);
24
25      if (isGrounded && velocity.y < 0)
26      {
27          velocity.y = -2f;
28      }
29
30      float x = Input.GetAxis("Horizontal");
31      float z = Input.GetAxis("Vertical");
32
33      Vector3 move = transform.right * x + transform.forward * z;
34
35      controller.Move(move * speed * Time.deltaTime);
36
37      if (Input.GetButtonDown("Jump") && isGrounded)
38      {
39          velocity.y = Mathf.Sqrt(jumpHeight * -2f * gravity);
40      }
41
42      velocity.y += gravity * Time.deltaTime;
43
44      controller.Move(velocity * Time.deltaTime);
45  }
46 }
```

Figure 4: Update() Method

7.3.2 MouseMovement.cs

Mouse movement is another essential part of navigating the application. An arbitrary number was chosen in accordance with my own familiarity, however, this can easily be tweaked in-engine, or in the future potentially with a dedicated settings system.

However, this addition would likely oppose the purpose of hand-gestures, and therefore an increase in unfamiliarity or discomfort in conventional movements might in fact incentivise users further to switch to gestures.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
    0 references
5   public class MouseMovement : MonoBehaviour
6   {
7
        2 references
8       public float mouseSensitivity = 100f;
9
        4 references
10      float xRotation = 0f;
        2 references
11      float YRotation = 0f;
12
        0 references
13      void Start()
14      {
15          Cursor.lockState = CursorLockMode.Locked;
16      }
17
        0 references
18      void Update()
19      {
20          float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity * Time.deltaTime;
21          float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity * Time.deltaTime;
22
23          xRotation -= mouseY;
24
25          xRotation = Mathf.Clamp(xRotation, -90f, 90f);
26
27          YRotation += mouseX;
28
29          transform.localRotation = Quaternion.Euler(xRotation, YRotation, 0f);
30
31      }
32  }
```

Figure 5: Mouse Movement Script

7.4 Environmental Design

Several themes were installed in order to generate a realistic version of an astral environment. From the Unity Store, the following addons were installed:

- 'MapMagic' – for terrain.
- 'Real Stars Skybox' for the skybox.

The 'skybox' is a Unity term for the omnidirectional background that dominates the sky, horizon and sub-horizon of any scene.

7.4.1 Skybox

The initial idea was to completely darken the sky, and splatter it with 3D models of stars at varying distances, including interactive constellations to receive and deposit stars (serving as a collection of datasets). This arrangement would produce a realistic parallax effect despite the limited range of movement of the camera when in-application. This was attempted using a skybox 'shader graph', as shown in *Figure 6.*

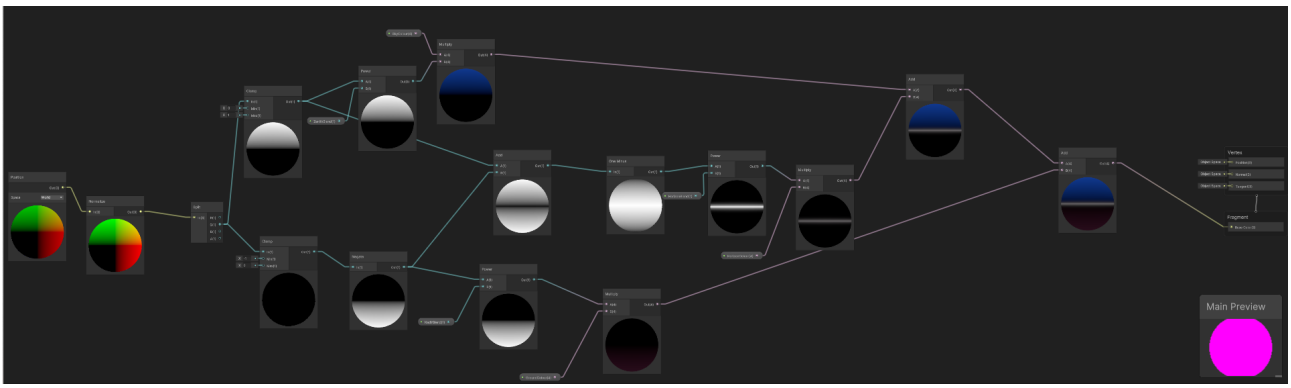


Figure 6: Original Skybox Shader Graph

This complicated structure produces a perfect space-like atmospheric, initially creating a horizon and sub-horizon, branching off to create a horizon based on the flipped image of their combination, then combining all three into one skybox. However upon completion it was revealed to be incompatible with the 3D URP version of Unity projects, in favour of the easier 'built-in renderer' version offered with much lower quality in exchange for ease. This is evidenced by the commonly seen bright pink colour in the bottom-right of *Figure 6,* which unless intentional, usually indicates a corrupted or missing texture.

Much later, a similar alternative was found in the 'Real Stars Skybox' asset found on the Unity Store, which didn't offer any parallax, movement, constellation or otherwise separation between sky, horizon and sub-horizon as the shader graph in *Figure* 6 might have offered. However, this was adequate for a realistic astral theme.

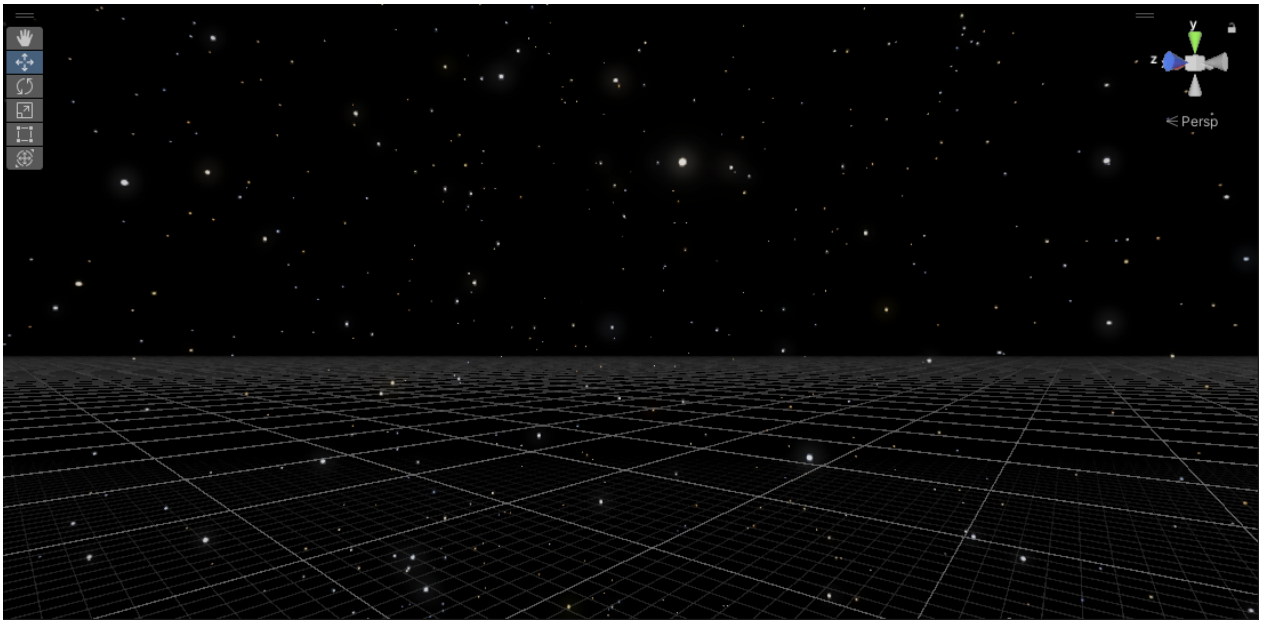


Figure 7: Astral Skybox

The skybox visible in *Figure 7* is a spherical asset with no disruption, detracting from the aesthetic but increasing the simplicity of the total system. Some aesthetic changes to improve the quality of this backdrop are discussing in the post-processing section of *Section 6.*

7.4.2 Terrain

The original idea was to emulate a planetary body in colour and texture. This was achieved by resizing the default terrain, applying a rocky texture, increasing its resolution for realism and creating uneven heights.

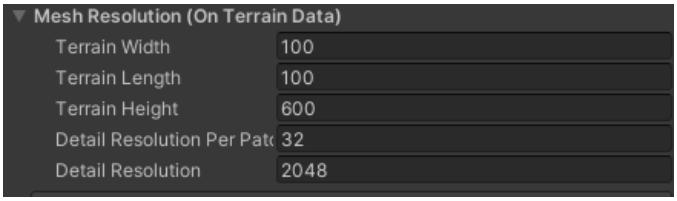


Figure 8: Terrain Size and Quality

The original terrain shown in *Figure* 2 was a practically infinite plane for the purposes of the application, in that more than a minute of movement in any direction would not cause the player to reach the edge of the terrain. This was resized to 100x100 distance units, and terrain quality was concurrently increased to over 16x the original. This resulted in massively increased texture resolution, allowing the user increased immersion from reasonable distances without increasing complexity of terrain objects.

Thereafter, the ‘TerrainURP’ texture from the ‘MapMagic’ addon was applied. Combined with the texture resolution increases and small terrain size, this bore more resemblance with a planetary body. Displayed clearly in *Figure 9* a mountainous boundary serves as the map perimeter, serving both purposes of aesthetic appeal and a physical border so that a curious user cannot fall off the terrain.

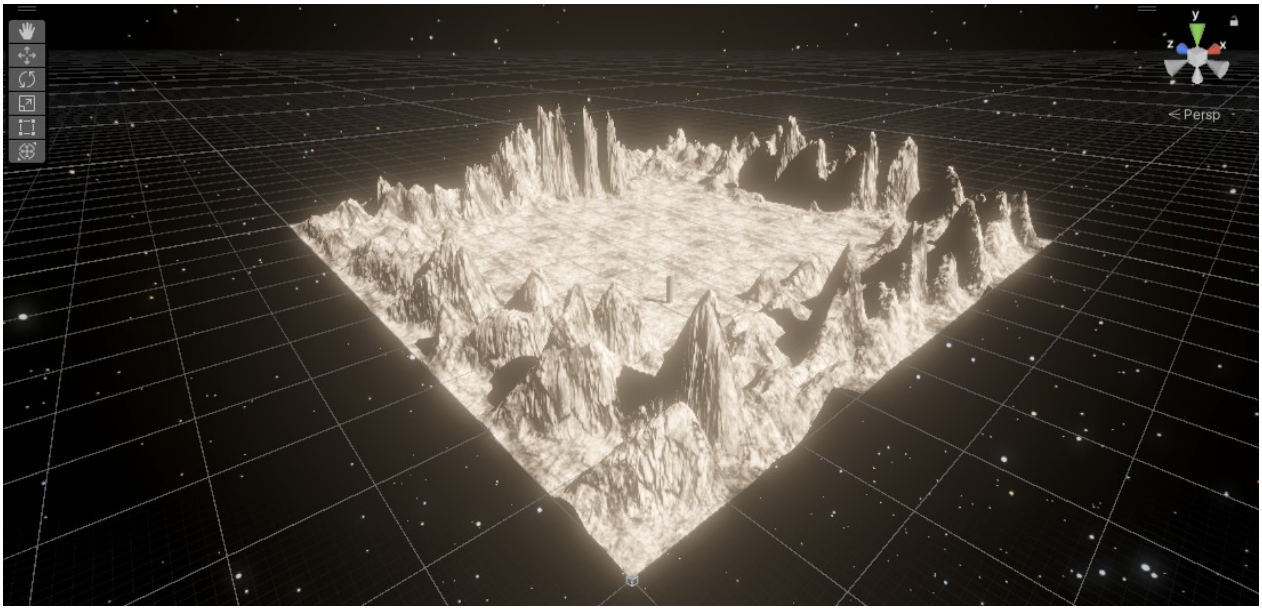


Figure 9: Terrain

The general movement area assumes about 70% of the terrain, and doesn't include any 3D surface changes for avoidance of unnecessary aesthetic pedanticism.
As mentioned prior, the post-processing element remains undiscussed till the end of this section.

7.5 Star

As the primary physical focus of the application, significant work went into the detail surrounding the star. This includes a custom shader graph similar to the initial development for the skybox mentioned in *Section 7.4.1.*

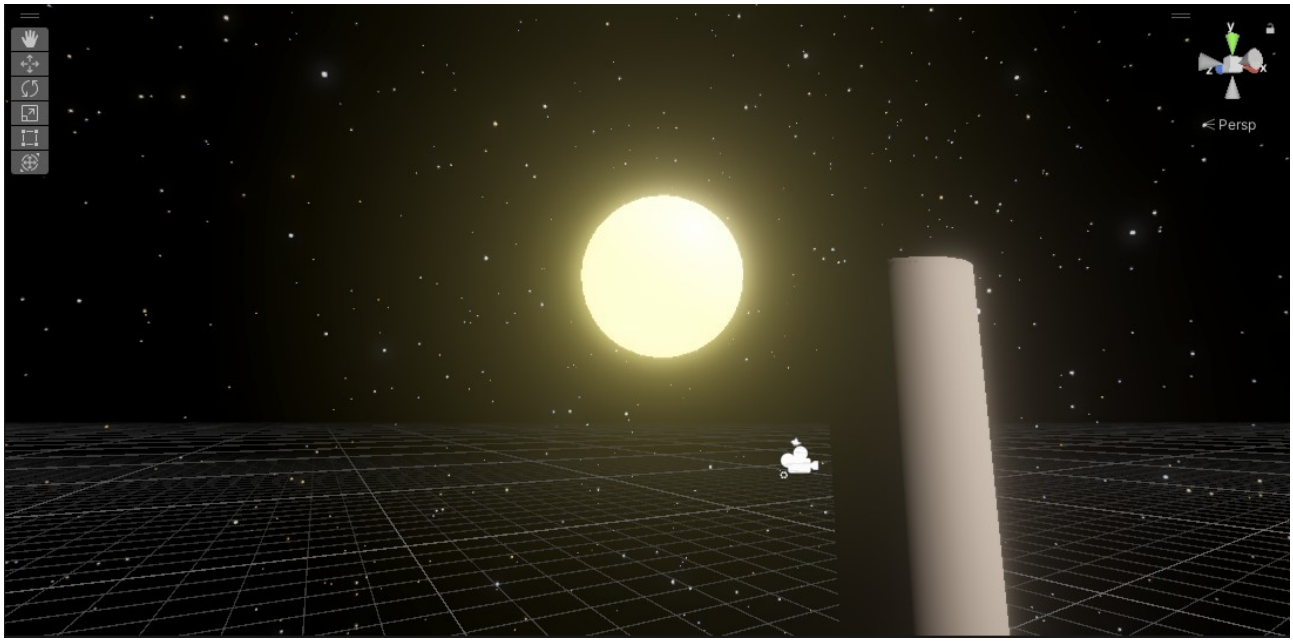


Figure 10: Star Object

Figure 10 shows a fully realised star object, with the player object positioned as if it were looking upon it. This position is an indication of the neglibility of the size of the player object in comparison to the astral body near it.

This colour scheme was achieved with the development of another custom shader graph much simpler in structure to *Figure 6.*

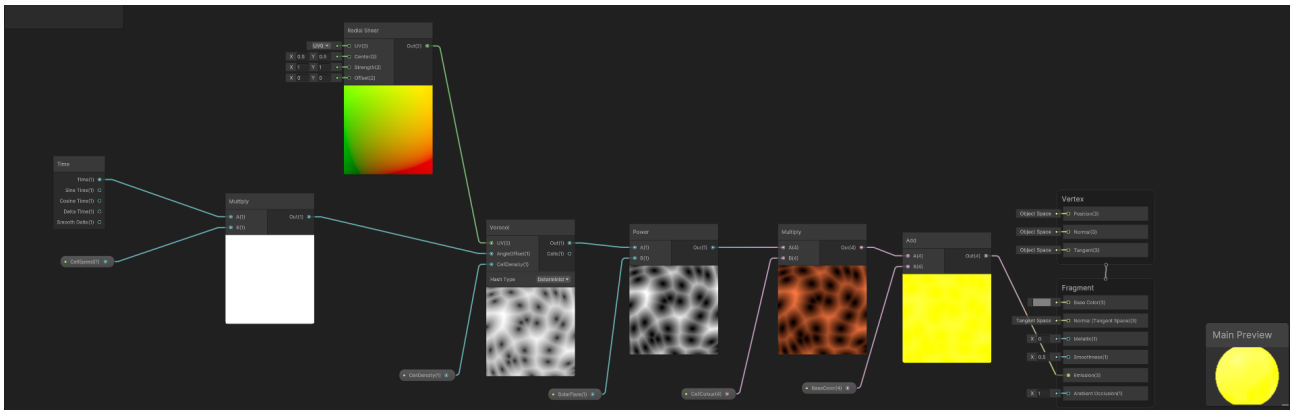


Figure 11: Star Shader Graph

A ‘voronoi’ node was used here, indicated in *Figure 11* by the first black and white square with an appearance of a ‘seeded soup’. A voronoi is a geometric concept that divides a space into regions based on distances to different points, as defined by ScienceDirect, and is commonly known by a lattice structure. Unlike *Figure 11* shows, the voronoi node used in the star shader graph is dynamic in nature and moves at random. This simulates the nature of nuclear fission within stars. It’s an effect near-invisible in-application due to post-processing, but remains foundational to the design aspect.

7.6 Final Environment

The yet undiscussed hidden feature that gave the application an ethereal, astral glow was down to post-processing techniques. The process to achieve this was simple but powerful:

1. 'Main Camera' object → Inspector → Camera → Rendering → Post-Processing ON
2. 'Global Volume' object → Inspector → Volume → Bloom → Threshold=0.5, Intensity = 1.5

The combination of these produces an astral glow not only in the stars and terrain, but also the skybox. The effects are evidenced in *Figures 13 and 15* following:

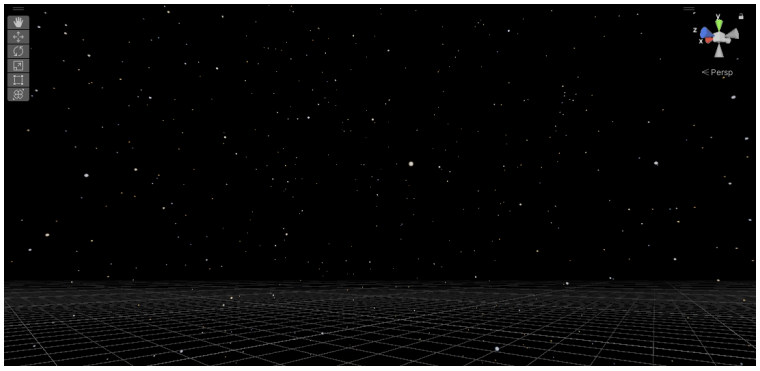


Figure 12: Skybox (no post-processing)

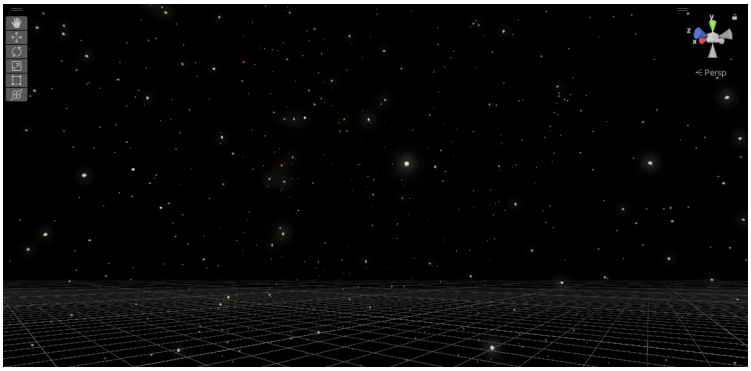


Figure 13: Skybox (post-processed)

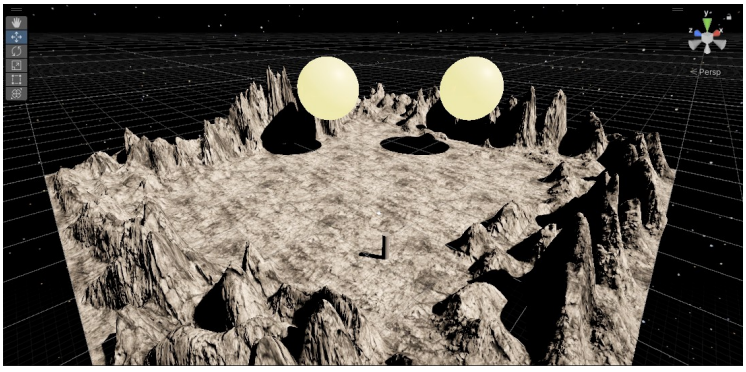


Figure 14: Terrain and Stars (no post-processing)

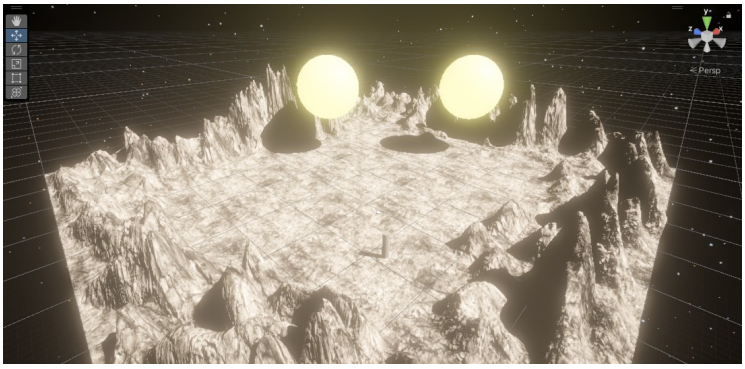


Figure 15: Terrain and Stars (post-processing)

The addition of post-processing to the project results in all surfaces resembling the videos of the closest planetary body to us – the moon, therefore keeping in line with the astral theme of the endeavour.

Tying together all the previous objects involved the larger-scale view of assembly for the purposes of the application. For the purposes of small-scale comparisons, the 3D stars were placed at the middle boundary of two adjacent walls, and the player object was set up nearer the opposite corner of the terrain, as shown in *Figure 14*. This gives the illusion of two paired astral bodies when considered against their lone viewer, which is an apt subconscious construction when the stars are taken as possessors of datasets to be compared.

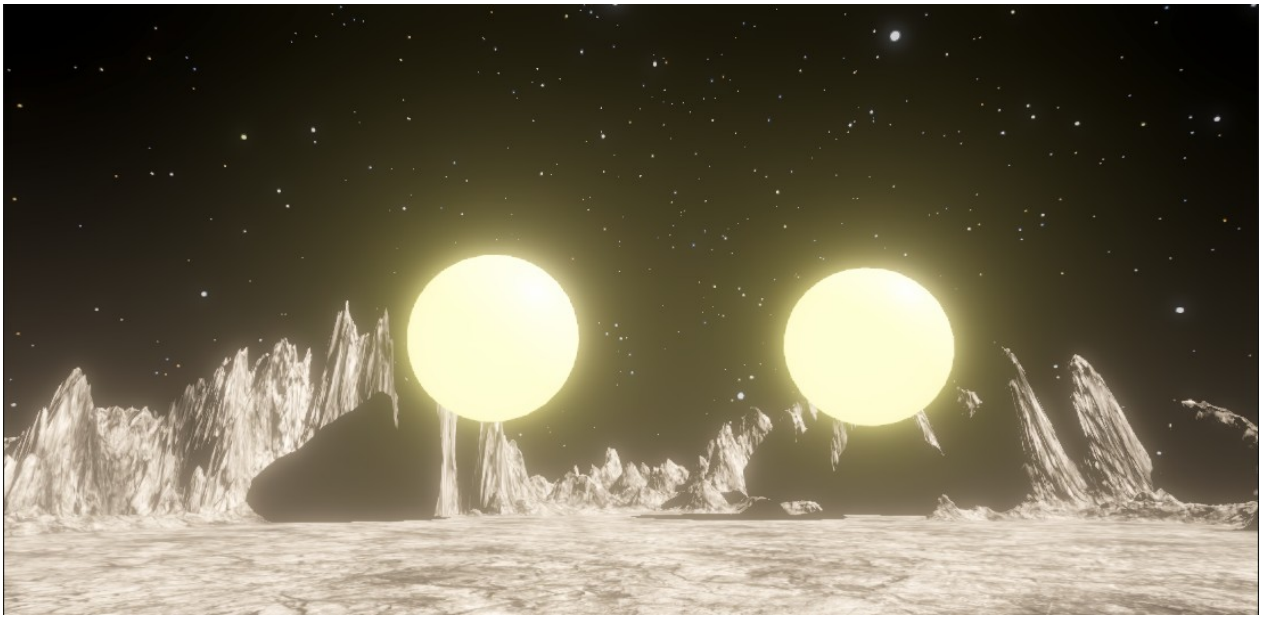


Figure 16: Final Environment

Following this, a number of pathways can be taken to achieve the required environment. The application is robust and thematically correct, and addition of complex code to integrate hand-gesture recognition with the in-application environment is not an endeavour to be considered worrisome for compatibility or failure, unlike reliance on questionable third party packages to patch-fix bridges between unsupported modules.

As the creation of objects ceased, an investigation into their analysis using the 11 gestures chosen out of the 27-class dataset follows.

7.7 Dataset Integration

As part of intuitive gesturing, the gestures must have made sense in that the ending portion of a single gesture would be conducive to the starting portion of any following gestural options. Great care was made in these choices, and a single example is given for the natural transition pathway for the first gesture to be explained following.

The gestures are split between ‘general’ and ‘precise’, to indicate the level of specificity in control that would be allocated to them. For example, *Gesture 9* would require a fist with two outstretched fingers for left-to-right precision control, as opposed to *Gesture 1* requiring a fully outstretched palm for a higher level left-to-right ‘general’ control.

7.7.1 General Gesturing

General use cases involve the “physical” manipulation of the data-point in the context of the application. This intends to be the most general method of not only navigating within the viewport, but also interacting with the 3D space within the application. The conventional navigation of the keyboard and mouse remains this would be made unfamiliar and uncomfortable as opposed to the intuition in gestural movement.

8 out of the 11 chosen classes have been chosen for the purpose of 'general gesturing':

1) **Class 2**: Left-to-Right (*scroll through stars*)
The palm is raised perpendicular to the camera's viewing angle, either at the midpoint or left of the camera's viewport. The palm is then moved to the right with a variable cadence*, to scroll through the available datasets, in the form of stars. The focus here lies on the user's perspective. As the user scrolls, they should be able to view all the available star objects, representing uploaded datasets.

An example of natural transitions follows:

1. Right-to-Left (*Gesture 2*)
   Panning **repeatedly** from left to right is something expected of a new user that simply aims to explore the environment. This should be optimised for fluidity as to give exemplary first impressions to a new user.
2. Fist-to-Palm (*Gesture 3*)
   As the main transitional gesture from the start stage to a star examination stage, this needed to be a suitably expansive movement, and the transition from the compact fist to a completely outstretched palm satisfies this condition.
3. Shake (*Gesture 7*)
   A much rarer command, used in tandem with *Gesture 11/Class 25.* Since this is available to use at all times regardless of position within the 3D space, it needed to be a more unique gesture, with a low probability of being mistaken for another gesture. Subsequently, the change in hand orientation is unique to this gesture.

Though continuing in this format would certainly uphold the dedication to unusual clarity, I find that such exceptional detail might be more suited to a better algorithm. Therefore, this remains an example of what intuitive transitional pathways a user could hope to achieve during usage.

2) **Class 1**: Right-to-Left (*scroll through stars*)
The palm is raised perpendicular to the camera's viewing angle, either at the midpoint or right of the camera's viewport. The palm is then moved to the left with a variable cadence*, to again scroll through available stars. Used in opposition to *Gesture 1/ Class 2,* this completes the ability to pan around the 3D space from the static axial position of the user.

3) **Class 11**: Fist-to-Palm (*expand star*)
Either the palm closes into a fist (not a recognised gesture), or a fist is brought into the camera's viewing angle. The fist then extends into an outstretched palm. The expansibility of this movement nearly doubles the volumetric occupancy of the hand, implying a revelation of 'true size' – which can comfortably map to an increase of information. This should reveal dataset information including category, as shown in *Figure 17.*

4) **Class 18**: Palm-to-Camera (*collapse star*)
The palm is brought towards the camera. The resultant increase in hand size, which can be visually compared to a pushing gesture, can be universally understood as a 'stay away' or 'get back' gesture. Within the context of examining data, this can be translated to a level decrease in information available to the user, and therefore the collapsing of a star back to expanded view.

5) **Class 16**: Palm-to-Ceiling Raise (*scroll dataset content*)
The palm is lowered, or brought into the view of the camera from the bottom, palm facing up. The hand is gradually raised. While the intuitive nature of this gesture can be argued *against*, I propose

that its efficacy lies more in the application concept. The direct effect of this gesture can be measured as the vertical scrolling of the contents of the dataset category.

6) **Class 17**: Palm-to-Floor Lower (*scroll dataset content*)
The palm is raised, or brought into the view of the camera from the top, palm facing down. The hand is gradually lowered. This works in tandem with *Gesture 5*/*Class 16* to produce a reliable method of scrolling through the dataset content available in each category of a multivariate dataset. The motion mimics a mouse scroll, which remains the conventional alternative.

7) **Class 12**: Shake (*clear comparison queue**)
The palm is repeatedly tilted in the sagittal plane. This means that the little finger is brought closer to the camera than the thumb, followed by the thumb being brought closer to the camera than the little finger. A sufficiently unique gesture is required for this, such that no accidental activations would occur in general navigation. This gesture exclusively works in tandem with *Gesture 3/Class 11*.

8) **Class 27**: Inverted Hand-to-Fist, Thumb Out, Horizontal Movement (*exit application*)
The hand is inverted, so that the back of the hand is visible to the camera. The fingers are closed while leaving the thumb outstretched, resulting in a similar visual to a 90 degree clockwise rotation of the 'thumbs up' sign. The hand is then moved laterally, and distally. By far the most complex movement in the set, this gesture was specifically chosen for its extraordinary obedience to the purpose required. Not only does the gesture offer relatively ridiculous complexity, but the end portion of it (lateral, distal movement) directly connotes a direction-order, commonly used while giving directions. It also bears notable resemblance to the thumb-over-the-shoulder gesture, which indicates an 'exit' suggestion in human communication.

*the variable cadence could imply to the application the speed of movement required. For example, if the user requires a precise examination of each star in the collection, the cadence should be slow, and the viewport should move in small degrees. If the user uses a violent cadence, the viewport should scroll through the datasets, given a sufficient number, till a gradual stop, as if it acts on an axis with reduced or minimal friction.

7.7.2 Precise Gesturing

Precise use cases give way to direct data analysis, as opposed to navigating the 3D space. 60% of the viability in this section relies on the post-expansion state using *Gesture 5/Class 16*. A comparison queue is available to interact with here.

The final 3 of the chosen 11 classes are explained below.

9) **Class 20**: Two-Finger Left-to-Right (*cycle through dataset categories*)

The user moves their hand to the left side of the camera's viewport, and raises two fingers. The user then moves their two-fingered gesture from the left side of the camera's viewport to the right. The prerequisite of this gesture is the expanded view of the star, where dataset categories are visible following *Gesture 3*. The use of this gesture allows the user to scroll through the categories available. A secondary usage not intended to be implemented is the slow and precise scrolling of the stars within the collection of datasets uploaded. This fits with the variable cadence property of *Gestures 1 and 2.*

10) **Class 21**: Two-Finger Right-to-Left (*cycle through dataset categories*)

The user moves their hand to the right side of the camera's viewport, and raises two fingers. The user then moves their two-fingered gesture from the right side of the camera's viewport to the left. Again, the result is dependent on application context.
The breakdown of this gesture is highlighted above, in *Gesture 9/Class 20*.

11) **Class 25**: 'Ok' Sign (*expand category/add star to comparison queue*)

The user raises their hand to the viewport, and touches the tip of their forefinger with the tip of the thumb, and splays the remaining three fingers. This (almost) universally signifies approval, dependent on culture – however American English users, which is a contemporary of the demographic this academic endeavour aims to entertain, generally understand the meaning behind the 'Ok' sign (Bergen, 2019). This connotation allows for a user to expand a highlighted category for content examination, or 'approve' a star to compare to another if at the collapsed stage. This gesture exclusively works in tandem with *Gesture 7/Class 12*.

7.7.3 Implementation

The requirements on a conceptual level are detailed in several hand-drawn diagrams shown below. This complex manipulation of the star objects is done in tandem with dataset integration. On all gesture-based choices, an additional last-resort conventional option is added to ensure integrity of exploration. No concurrent additions were made in the Unity application to accommodate this extra gesturing, however the following 3 sections allow the reader or future authors to comfortably expand upon these concepts using this section as a guide.

Approach

On user approach, at an arbitrary threshold of 5 virtual metres, the star expands to 1.2x its original size, with a label appearing above the star stating the name of the dataset with which that particular star is associated.

There are then two pathways:

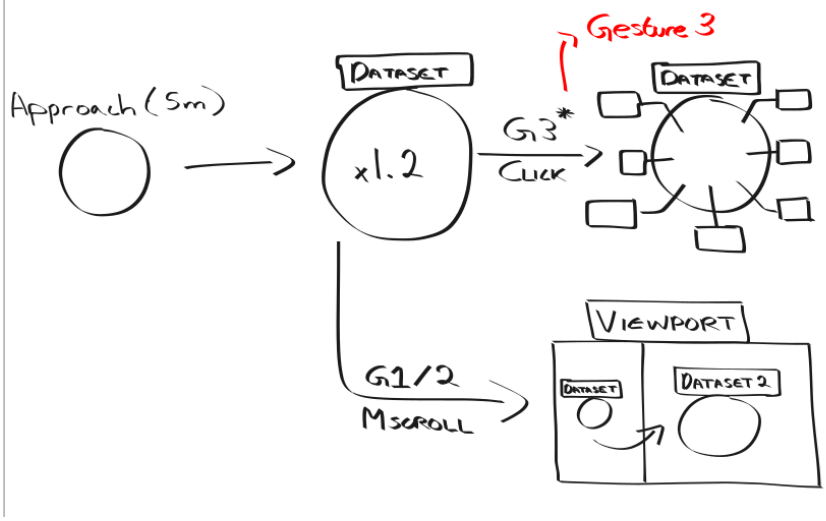


Figure 17: Star Approach Interaction Pathway

1. *Gesture 3* (fist to palm) or mouse click
   The enlarged star is dismantled into several categories in accordance with dataset categories.

2. *Gesture 1* (right to left) or *Gesture 2* (left to right) or mouse scroll
   This option shifts the viewport to an adjacent star, and therefore dataset, if it exists.

The implementation of these options are cornerstones of the intuitive theme to be explored in this endeavour. The enlarged star being further expanded by *Gesture 3* is a natural movement, or in the second case the user can scroll through nearby stars using *Gesture 1* or *Gesture 2*.

View

Since the only pathway that offers further options in *Figure 17* is *Gesture 3* onwards, *Figure 18* continues into the expansion. A highlighted selection is offered on each category of the dataset. Category boxes would increase dependent on the categories available in the dataset, up to a maximum of 10 for avoidance of overlap and visual clutter.

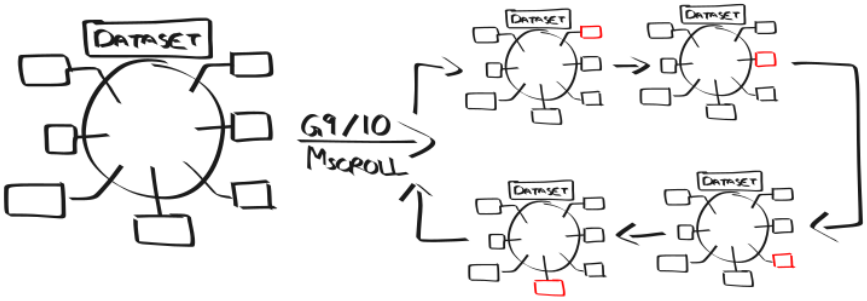


Figure 18: Star View Interaction Pathway

The category boxes can be scrolled through with the use of *Gesture 9* and *Gesture 10*, or the conventional option of mouse scroll. No further pathway is available.

Expand

On visual selection of a category for inspection, the user can then use *Gesture 3* or the ‘e’ button as a conventional keyboard alternative to expand the category and view its contents. With the category

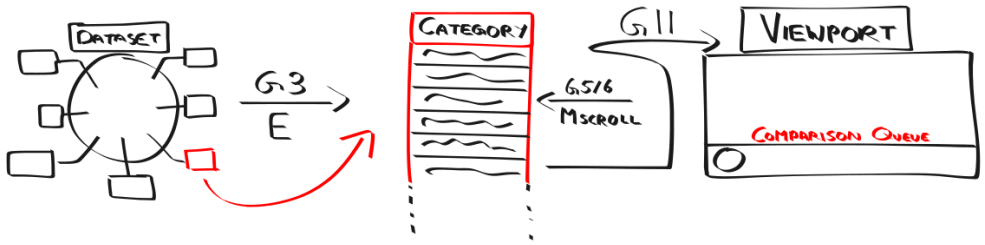


Figure 19: Star Selection Interaction Pathway

name highlighted at the top of the list of contents, the user can then use *Gesture 5* or *Gesture 6* to vertically scroll the list. The conventional alternative of mouse scroll is again available, however there is potential to incentivise gestural use by decreasing the ‘friction’ level of the scroll movement, thereby allowing the list to slide more or less than what convention and familiarity demands.

Thereafter, the entire category can be chosen for comparison using *Gesture 11*, in which case a comparison queue resembling an in-application menu would appear on the lower sliver of the viewport as a visual reminder of the datasets available for comparison. The implementation of *Gesture 7* could then clear this queue in later iterations.

7.7.4 Failures

The implementation of the dataset-integrated gesture options is explained more clearly earlier, but when specifically considered against the creation of the star object's animations and mapping integration, it became far too complex for the nature of this dissertation. The Unity Engine environment was completely novel area of learning separate from all modules of my degree studies, and the estimated total hours of creating the environment seen were too high to justify further time spent when the classifier algorithm was of more importance.

Therefore, although a novel standalone application for visual data analysis was created, the instantiation of gestural manipulation were given as diagrammatic concepts instead.

8 Classifier Algorithm Development

I employed a process of iterative improvement in order to arrive at the machine learning environment and algorithm that is presented in the next few sections.

In all, 7 hours 52 minutes of coding footage was recorded over the weeks of the development process, minus unrecorded programming, conceptual design, research, video tutorial use and design retractions, and a healthy estimate would put the total between 150-200 hours, considering the study of the CS229 course and the complexity of algorithm development. This was recorded for archival purposes.

The required commands were 11 out of the initial 27-class dataset.

8.1 Original Plan

I purpose-built a computer for this module, installing sufficient hardware to power any undergraduate-level machine learning application. I aimed to seek out the most efficient and powerful method of producing a machine learning model that would take full advantage of the processing power of this computer. To this end, I picked libraries that were conventionally used in the field at the time of writing.

I opted primarily to use a combination of the following:

- Ubuntu 24.04
- Python 3.12.3
- Tensorflow 2.16
- OpenCV 4.7.0
- VirtualBox

The VirtualBox VM was allocated 16GB of RAM, and 500GB of storage on creation.

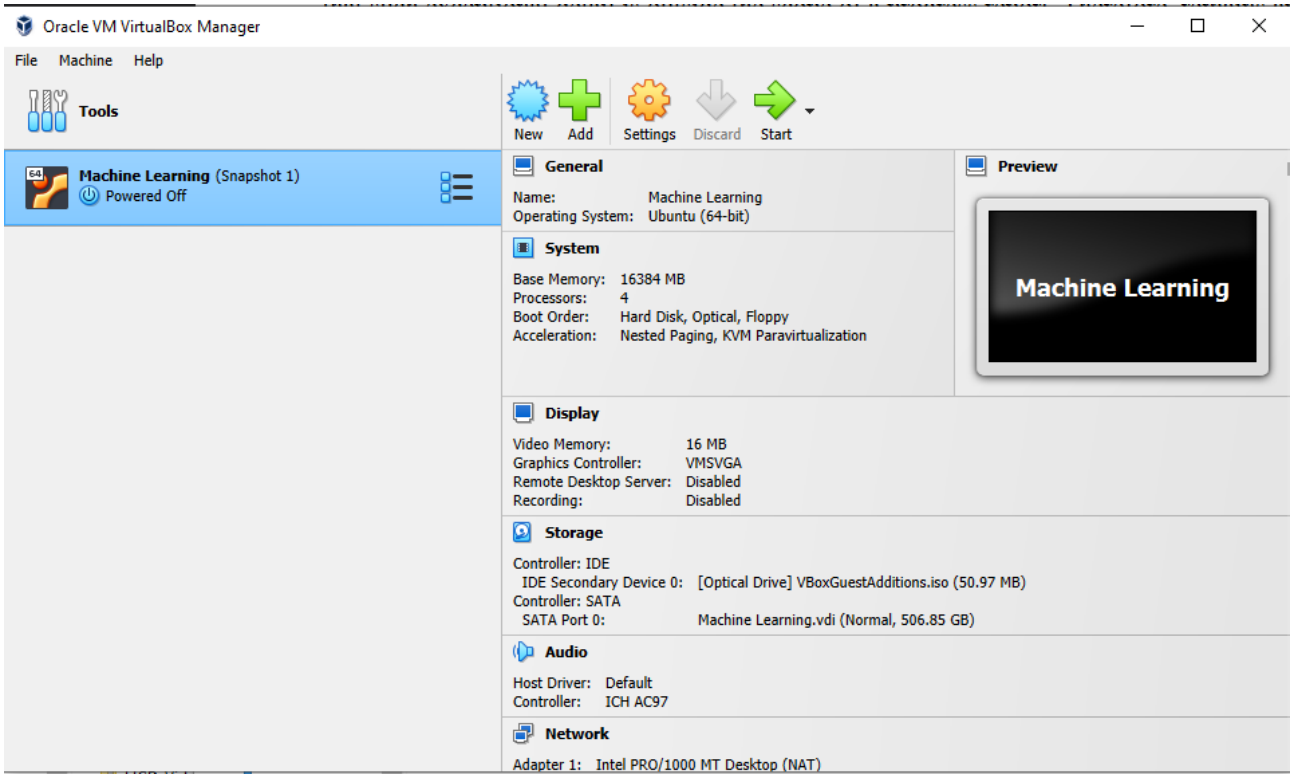


Figure 20: Virtual Machine used for Machine Learning

8.1.1 Ubuntu 24.04

As the classic base Linux flavour for most purposes, this choice was least likely to have OS compatibility or performance throttling issues, and also avoid unnecessary complexity/learning curve with a machine-learning specific distribution. Version 24.04 was also the latest with LTS, ensuring not only experimental reproducibility long after the submission of this dissertation, but also a higher likelihood of seamless, bug-free usage.

A 5.69GB (binary) .iso file of this OS was downloaded from the official Ubuntu website, then allocated to the creation of a new VirtualBox virtual machine.

8.1.2 Python 3.12.3

Python is the fundamental language for both of the most popular machine learning libraries at the time of writing – TensorFlow and PyTorch. Concurrently, it finds itself necessary for the tertiary OpenCV library, which means it is an irreplaceable building block of this dissertation. Naturally, the latest version was also chosen, but this led to a critical compatibility issue.

Though I had minimal experience of Python before this dissertation, a lack of familiarity amplified by the fact that my CST3170 Artificial Intelligence module was taught entirely in Java, I found that this was one of the many novel fields I had to learn and explore in pursuit of the problem highlighted in this paper.

8.1.3 Tensorflow 2.16

Tensorflow 2.x was a QoL upgrade from TensorFlow 1.x, since it integrated the deeply popular Keras API within the TensorFlow package, allowing users an upgrade in both performance and convenience. Keras was a required package for some outdated classifier algorithms I tested.

While PyTorch was continually considered as a potential alternative library in the case that I didn't find a TensorFlow classifier algorithm, the niche specificity of the requirement as referred to in meant that there was limited option regardless of whether I used TensorFlow or PyTorch.

8.1.4 cygwin64

This windows shell was used to emulate a linux environment, installed with the **git** library, which was used to clone every project in the following sections till Anaconda was discovered.

8.2 Initial Testing

After creating the virtual machine, installing the OS, and the dependencies outlined in the original plan using 'pip', a folder was created to house the dataset. This was downloaded direct from the source using the AWS-CLI.

To test that the dependencies were correctly installed, and that there were no runtime errors on their import, I used the following python code:

```
1  import tensorflow as tf
2  import pandas as pd
3  import opencv as cv
4
5  print(f"TensorFlow version: {tf.__version__}")
6  print(f"pandas version: {pd.__version__}")
7  print(f"opencv version: {sk.__version__}")
8
```

Figure 21: Import Error Check

After this, a suitable classifier algorithm was sought.

8.3 Classifier Algorithm

As a specific classifier remained an extraneous and non-essential part of this dissertation, I took to github in order to use work that already existed. However, it was important to ensure the one chosen was both accurate and conducive to the requirements of this project. The key requirements in this search were as follows:

- Hand-Gesture-Recognition
- TensorFlow
- OpenCV

Another unconscious preference in result search was that of CNN-based classifiers, since standalone CNNs rank the highest in accuracy among all machine learning algorithms for hand gesture recognition (Bhushan et al., 2022).

Note that at this stage I did not differentiate between static and dynamic hand gesture recognition. Notable results were as follows:

No.	Name	Author	Last Update	Dynamic/ Static	requirements.txt	Worked
1	Unified Gesture Recognition and Fingertip Detection	MahmudulAlam	2 years ago	Static	Yes (in description)	No
2	HandMovementTracking	Akshaybahadur21	2 years ago	Dynamic	Yes	Yes
3	HandGestureRecognition	Ha0Tang	3 years ago	Dynamic	No	No
4	Hand-Gesture-Recognition-Using-Background-Elllimination-and-Convolution-Neural-Network	SparshaSaha	4 years ago	Static	Yes (in description)	Yes

Table 1: Notable Projects Carried to Execution

8.3.1 Project 1 (MahmudulAlam)

This algorithm looked promising, and included fingertip detection as an additional positive. The requirements were as follows:

- TensorFlow-GPU 2.20
- OpenCV 4.2.0
- ImgAug 0.2.6

with an impressive result including fingertip-level detection.

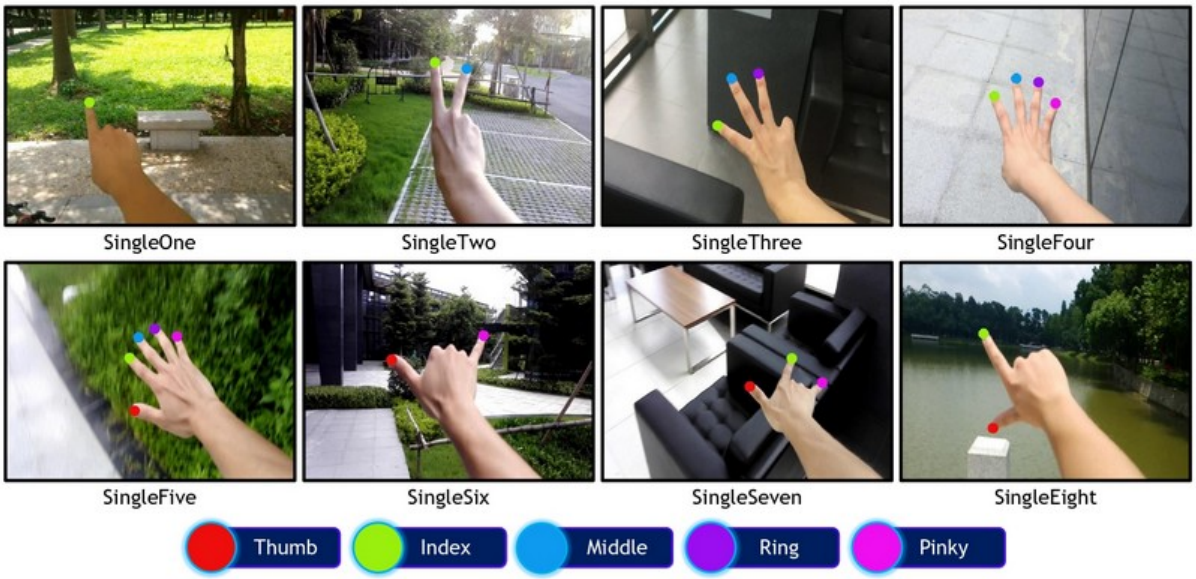


Figure 22: Classifier Algorithm 1 - Recognised Gestures with Fingertip Detection

Several runtime errors plagued execution even before the model could be tested. This was indicated to be because of compatibility issues – which in hindsight was likely because I wasn't using a dedicated package manager. After resolving with manual downloads of packages, fatal method errors occurred, whereby the system seemed to lack the compilation knowledge to understand where the definitions of the methods were. The errors are detailed in the section immediately following.

8.4 Revision: Hardware/VM Incompatibility

Despite arriving at this step with little to do with a classifier algorithm apart from sourcing an existing one for the purposes of this dissertation, this task was hindered due to the complexity of the circumstance of the project and hardware used.

Concurrently to failure of execution of the first project, a compiler warning indicated that TensorFlow lacked a software route to access the processing power of my GPU. The next several days were spent trying to initiate compatibility between the TensorFlow library installed on the virtual machine and my GPU. The following results were discovered.

AMD GPUs are not favoured for machine learning tasks. The only route to compatibility and use in TensorFlow and any train/preprocessing stage is through the patch-fix ROCm libraries, which aim to create a virtual bridge between the computational requirements of a machine learning task and an AMD GPU. This patchwork broke down completely within the fundamental virtual environment I had employed, and the hardware that backed it. The following required software compatibility and bridges that were currently unsupported in this specific configuration, massively increasing complexity.

- Virtual Machine
- Ubuntu
- TensorFlow
- AMD 6700XT (unsupported by ROCm)

Because of the rarity of this virtual and hardware environment configuration, almost all relevant instruction was catered to advanced users and unsuitable for the purposes of this dissertation. Prior to immediate switch out of a virtual machine, another option was sought.

8.4.1 Project 2 (Akshaybahadur21)

This dynamic hand gesture recognition classifier remains the most accurate out of all tested, tracking fingertips as if one were writing on a tablet. The potential use case of this seems the most useful and relevant to the purposes of this dissertation, which is why despite its lack of an explicit classifier, it was trialled and improvements were attempted.

After creation of the required modules and running the script, a plethora of runtime errors again prevented correct execution. Concurrently, there was no sign that the virtual machine could access the host's webcam device – a feature that would be a serious security risk without intentional and dedicated modification to the client/host relationship akin to the shared folder mentioned in *Section* .

A switch entirely to the host client of Windows 10 allowed me to continue to successful execution of the code. This switch is detailed in *Sections* . I will continue the methodology for testing of this classifier otherwise uninterrupted.

Figure 3 shows an example of the incredible accuracy achieved – which could have been critically useful for tab-level manipulation in Section . The outstretched hand's fingertip is detected and a clear trail is left.

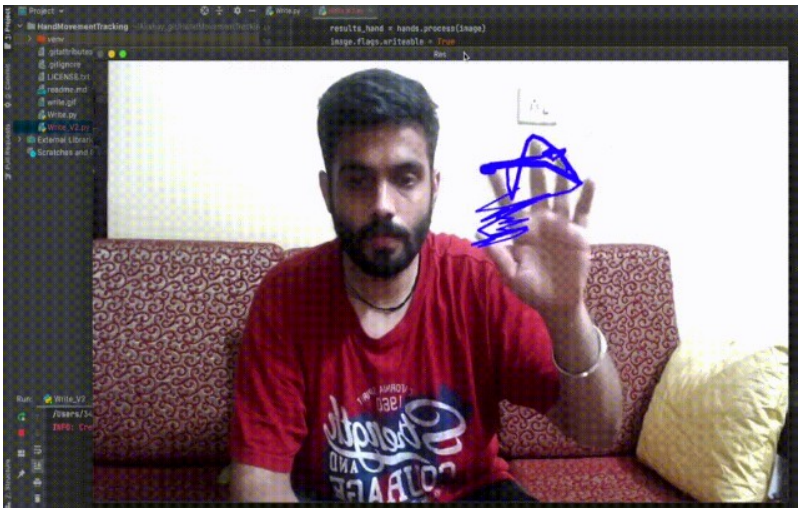


Figure 23: Classifier Algorithm 2 – Dynamic Fingertip Detection

One could imagine that simple shapes could be drawn in the place of hand gestures, in order to perform complex actions within the data analysis application – but 'drawing' in the air in a 2D plane parallel to a webcam would arguably not fit the scope of intuition that this dissertation seeks.

The lack of a defined machine learning algorithm here was the case to abandon this route, because however much a novel solution would be preferred, this was not the purpose of the dissertation. As mentioned earlier, after exploration of this algorithm, a return to the host client was decided.

8.5 Revision: Windows Host Transition

At this point, the decision was made to return to the host client in order to reduce the staggering amount of compatibility issues during execution of projects.

Alongside the transition, further research about package installation emulators on Windows 10 led to the solution of Anaconda Navigator – an open-source application centred around data science and machine learning tasks, offering a host of tools to make ML development far easier.

8.5.1 Anaconda

Alongside a terminal-based CLI akin to 'pip' for package installation and many data science tools, Anaconda provided access to virtual environment in a more accessible format than Python's 'venv'.

The most essential tools for the purposes of this dissertation included this CLI package manager and its access to a vast repository of ML-specific libraries, and the Conda Virtual Environment within which one could install specific combinations of them. These were used initially to reproduce the requirements of each project listed in , but were later used to compile and stack an efficient purpose-specific environment for the novel algorithm developed and referenced in *Section* . Further, .yml files were used in the same way as requirements.txt files mentioned in *Table 1* and across the rest of *Section* .

In order to create a virtual environment, the following command was used:

conda create –name <env-name> <python-ver>

For most of these projects, Python version 3.9 would be used, since TensorFlow support after that version was limited. Following this, a series of related modules would be installed, depicted below.

The addition of related modules on every manual install lends credence to the long list of dependencies of a fully realised virtual environment, example listed in *Figure* .

```
(base) C:\Users\Erwin>conda create --name SparshaSaha python=3.9
Retrieving notices: ...working... done
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Erwin\anaconda3\envs\SparshaSaha

  added / updated specs:
    - python=3.9


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    openssl-3.0.14             |       h827c3e9_0         7.8 MB
    ------------------------------------------------------------
                                           Total:         7.8 MB

The following NEW packages will be INSTALLED:

  ca-certificates    pkgs/main/win-64::ca-certificates-2024.3.11-haa95532_0
  openssl            pkgs/main/win-64::openssl-3.0.14-h827c3e9_0
  pip                pkgs/main/win-64::pip-24.0-py39haa95532_0
  python             pkgs/main/win-64::python-3.9.19-h1aa4202_1
  setuptools         pkgs/main/win-64::setuptools-69.5.1-py39haa95532_0
  sqlite             pkgs/main/win-64::sqlite-3.45.3-h2bbff1b_0
  tzdata             pkgs/main/noarch::tzdata-2024a-h04d1e81_0
  vc                 pkgs/main/win-64::vc-14.2-h2eaa2aa_1
  vs2015_runtime     pkgs/main/win-64::vs2015_runtime-14.29.30133-h43f2093_3
  wheel              pkgs/main/win-64::wheel-0.43.0-py39haa95532_0
```

Figure 24: Example Creation of 'Conda Virtual Environment'

Importance:
Several times over the testing for the classifier algorithm, a requirements.txt or similar file was shared in order to provide an exact virtual environment that will require no further installations or troubleshooting steps in order to run. Simply cloning the repository with git into a folder, activating a conda virtual environment and installing all the libraries in the requirements.txt file would result in success. For a handful of dependencies, this could simply be typed into Anaconda Prompt as:

- **conda** install *dependency1 dependency2 dependency3 …*

- **pip** install *dependency1 dependency2 dependency3 …*

Or in cases with double-digit dependencies, a simple bash script could be used, such as the one I prepared:

```
#!/bin/bash

ENV_NAME="my_new_env"

conda create --name $ENV_NAME --yes

source activate $ENV_NAME

while read package; do
    conda install --name $ENV_NAME $package --yes
done < requirements.txt

echo "Environment '$ENV_NAME' created and packages installed."

```

Figure 25: Bash script to install lengthier dependency lists

However, in some cases, especially with student projects, I'd find no such requirements/dependencies file. In this case, it became laborious, and often the solution I took was to run the file continuously and depend on the compiler error to tell me what library was missing, such as in .

Execution of projects after this was generally far more comprehensive and successful given the rigorous compatibility checks of the Conda package manager.

8.5.2 Project 3 (Ha0Tang)

This project was most favoured in the niche section of github that was browsed, and was submitted as part of a research paper (Tang et al., 2019), therefore making it a strong candidate for the classifier algorithm. The algorithm was robust and detailed, featuring key frame extraction, histograms and feature fusion to produce processed data akin to the type I produced myself in *Section* to present for classification.

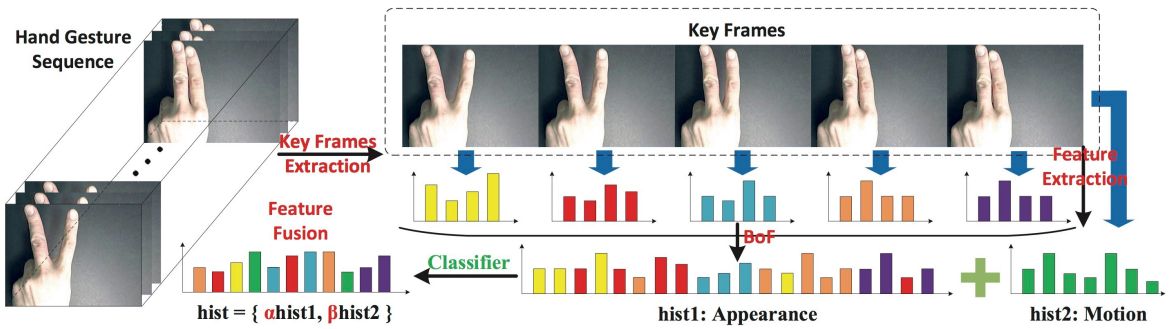


Figure 26: Feature Fusion Module (Ha0Tang)

The project was aptly flexible, considering its academic basis, and specified instructions on how to insert a new dataset for training. However, the complexity with which it was designed assumed an experienced user. The setup process required generous use of the terminal, and manual manipulation of the dataset including extracting frames and use of MATLAB, which imposed a steep learning curve. At this point, it was clear that more knowledge was required to be able to decipher and work with these high-level projects, and therefore I resolved to dedicate a large portion of time to piece together my own classifier algorithm using a base from a simpler project.

8.5.3 Project 4 (SparshaSaha)

The first real success of full hand gesture recognition came in the form of SparshaSaha's HGR system, combining CNNs with OpenCV and several other modules in order to detect three classes of gesture. The three gestures are 'palm' (open palm facing the camera), 'fist' (closed fist facing the camera), and 'swing' (closed fist with pinky finger and thumb outstretched; depicted in *Figure 6.*

With the addition of a custom dataset available, the only fundamental missing from this choice was its static nature being removed from the dynamic requirement of the dataset and dissertation.

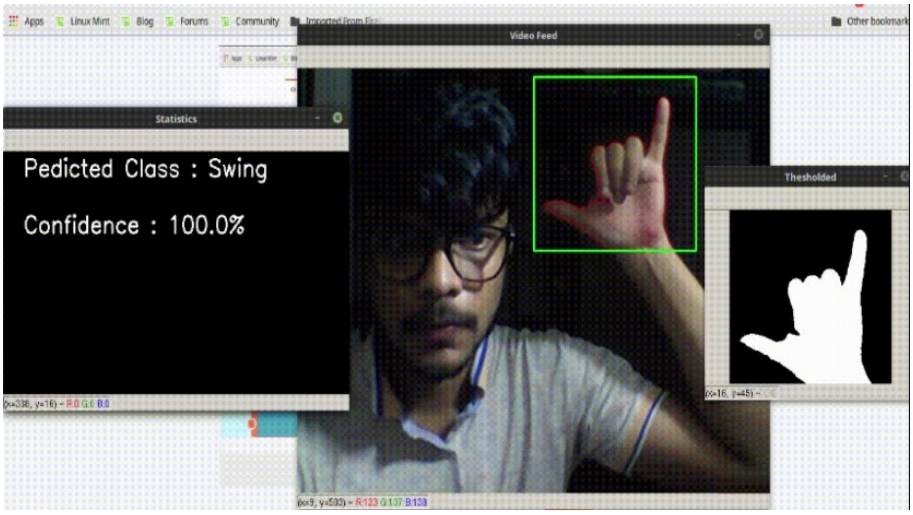


Figure 27: 'Swing' Class (SparshaSaha)

This was a common theme in these projects, which is further detailed and explained in *Section* . Usage of the project code was arduous considering its dependence on a deprecated version of TensorFlow (1.13.1) and the resulting plethora of runtime errors required several manual installations, starting with an upgrade to the Keras-supported TensorFlow 2.x.x version.

This project's success and simplicity led to its usage as a base for the development of the novel algorithm detailed in *Section* . The unforeseen requirement of this foray into creation of a classifier algorithm is explained in *Section* .

8.6 Dataset Transfer

At this point, the dataset had to be transferred in its entirety to the host machine. The well-known ssh was an obvious solution to achieve this, but I found the method unintuitive and convoluted when weighed against its limited use within this dissertation.

Nevertheless, ssh-server was installed on both the host and client, and terminal commands were employed to find the IP address for both within Conda Prompt, but the clear-cut path was corrupted by complex shortly after, I assume due to the circumstance of the client/host machine relationship. In any case, further troubleshooting was required. When the issue couldn't immediately be resolved, I switched to the native option given by VirtualBox – GuestAdditions.

After virtual insertion of the CD (in the form of content download and mount to drive), a single folder that would thereforth be shared by both host and client was selected. In order to avoid issues with data corruption or other extraneous software bugs, the folder was not the one the dataset currently resided in. The initial classes (9GB each) were moved at 20-80MB/s to the shared folder, and the rest of the 265GB dataset was moved using the *mv* command to attempt increased speed. This unusually low speed was due to the dataset's non-sequential nature, despite the disk reaching 5000MB/s on normal files.

After successful transfer, extraction of the directory structure in *Figure* was completed using the Python script shown in *Figure* .

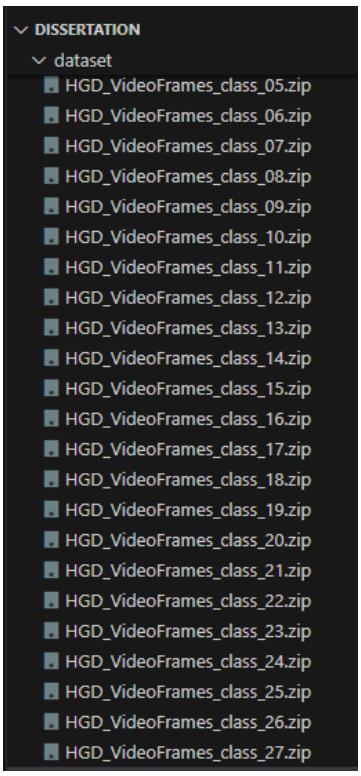


Figure 28: Zipped Classes

```
extract_all.py > ...
1   import os
2   import zipfile
3
4   dataset_path = 'dataset'
5   extract_to = 'extracted_data'
6
7   if not os.path.exists(extract_to):
8       os.makedirs(extract_to)
9
10  for zip_file in os.listdir(dataset_path):
11      if zip_file.endswith('.zip'):
12          zip_file_path = os.path.join(dataset_path, zip_file)
13          with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
14              zip_ref.extractall(extract_to)
15
16  print("Extraction complete.")
17
```

Figure 29: Dataset Extraction Script

Following extraction to the shared folder, the dataset was able to be used to develop a classifier algorithm.

8.7 Revision: Novel Algorithm

This build process took several iterations, keeping in line with the policy of iterative improvement that underlines this dissertation and any other academic endeavour.

In order to create a classifier similar to the static example by SparshaSaha in *Section* , a preprocessing model was developed, and trained on a CNN. In the medial stages of development, this was temporarily switched to an LSTM since they were inherently more accurate when classifying dynamic gestures. Whereas a combination of both could achieve remarkable accuracy of 98%+ (Wu et al., 2018), endeavour towards this goal was not the primary focus of this dissertation. The following stages were required for a simple machine learning model:

1. Preprocessing
    1. Load Data
    2. Extract Frames
2. Training
3. Classification

These were assembled into a complete pre-process, train, classify project contained within three python files: preprocess.py, train.py, start.py. By the end of this process, I had created a working deep CNN suitable for basic dynamic hand gesture recognition.

Three iterations were developed before arrival at the currently used algorithm. While saved in the working directory during time of development as up-to-date files, the discarded iterations were

moved to aptly labelled folders for archival purposes following the genesis of a new version. Iteration 2 and 3 are missing the main.py script, because it remained generally unchanged till the final revisions. Finally, the first iteration contains most of the explanation, while the rest focus on changes and their effect.

8.7.1 Iteration 1 (Failure)

This iteration was error-ridden and unreasonably tough on storage. Despite having over 500GB freed after the deletion of the virtual machine mentioned in *Section* , the pre-processing script ran out of storage before the end of execution, shown at the end of this explanation.

Preprocess.py

The first steps were to create a method to extract and resize the dataset frames from a specified directory. Since there were multiple folders in the extracted_data directory, a loop was required. To complete preprocessing, the acquisition of a list of frames and a list of labels were required. As such, we can break down the initial creation using sections and definitions within the code.
Figure 31 shows the extract_frames() method, in which each frame in the directories is read and resized using OpenCV. This is then saved to the list initialised at the start of the method. At initial testing of this code, this method would run (and therefore preprocess) on every execution, causing extremely inefficient and slow runtime. However, in later testing a section was added in Figure 33 that checked for the existence of pre-processed data and skipped this method if it found earlier data. This works because the dataset to train the model was static, while the datasets to be compared were not involved in this process, and is followed by the second method.

```
scripts > old > preprocess.py > load_data
import os
import numpy as np
import cv2
import pandas as pd

def extract_frames(video_path, label, img_size=(64, 64)):
    frames = []
    for frame_file in sorted(os.listdir(video_path)):
        if frame_file.endswith('.png'):
            frame_path = os.path.join(video_path, frame_file)
            frame = cv2.imread(frame_path)
            frame = cv2.resize(frame, img_size)
            frames.append(frame)
    return frames, [label] * len(frames)
```

Figure 30: Extract_Frames() Method

```
Processed data for class 2, user 18
Processed data for class 2, user 19
Processed data for class 2, user 20
Processed data for class 2, user 21
Warning: Directory extracted_data\class_11\User1_\User1_11 does not exist. Skipping.
Warning: Directory extracted_data\class_11\User2_\User2_11 does not exist. Skipping.
Warning: Directory extracted_data\class_11\User3_\User3_11 does not exist. Skipping.
Warning: Directory extracted_data\class_11\User4_\User4_11 does not exist. Skipping.
Warning: Directory extracted_data\class_11\User5_\User5_11 does not exist. Skipping.
```

Figure 31: Processing Error

A second addition to this code excerpt was a result of previously unsorted gesture class data. Though the directory was naturally ordered by number, and each image was appropriately numbered to keep the correct order, the python compiler assumed that 11 was of greater priority than 1. This is shown in Figure , where the compiler skipped a majority of the data in a directory. The load_data() method is split into several sections due to length.

The first section shown in Figure 32 defines some code that checks if any pre-processed data exists, as mentioned earlier. If no pre-processed data exists, then it begins the initialisation of empty lists to store both images and labels, and a dictionary to map gesture

```
def load_data(extracted_path, csv_path, img_size=(64, 64), save_path=None):
    if save_path and os.path.exists(f"{save_path}_X.npy") and os.path.exists(f"{save_path}_y.npy"):
        print("Loading preprocessed data from disk...")
        X = np.load(f"{save_path}_X.npy")
        y = np.load(f"{save_path}_y.npy")
        return X, y

    print("Loading data...")
    images = []
    labels = []
    target_classes = {
        'class_01': 0, 'class_02': 1, 'class_11': 2, 'class_12': 3, 'class_16': 4,
        'class_17': 5, 'class_18': 6, 'class_20': 7, 'class_21': 8, 'class_25': 9,
        'class_27': 10
    }
```

Figure 32: Load_Data() Method: Pre-Process Check and Initialisations

classes to numeric labels. These are based on the gestures used in this report, referenced in detail in Section .

Figure 33 then continues to the pre-processing section, most of which was taken from generic sources. In essence, the chronology of execution is as follows:

The list of gesture classes are received and readied for processing. An addition during development is seen at Line 36, where the received gesture classes are forcibly filtered through a checkpoint that ensures they are only of the approved classes instead of the full 27-class dataset. This was added after a test run whereby the entire dataset was extracted, decreasing code efficiency and increasing executional delay dramatically.

```
34      gesture_classes = os.listdir(extracted_path)
35      for gesture in sorted(gesture_classes):
36          if gesture in target_classes:
37              print(f"Processing {gesture}...")
38              gesture_path = os.path.join(extracted_path, gesture)
39              users = sorted(os.listdir(gesture_path), key=lambda x: int(''.join(filter(str.isdigit, x)) or -1))
40              for user in users:
41                  print(f"Processing {gesture}/{user}...")
42                  user_path = os.path.join(gesture_path, user)
43                  for gesture_performance in sorted(os.listdir(user_path), key=lambda x: int(''.join(filter(str.isdigit, x)) or -1)):
44                      gesture_performance_path = os.path.join(user_path, gesture_performance)
45                      frames, frame_labels = extract_frames(gesture_performance_path, target_classes[gesture], img_size)
46                      images.extend(frames)
47                      labels.extend(frame_labels)
48
49      X = np.array(images)
50      y = np.array(labels)
51
52      if save_path:
53          os.makedirs(os.path.dirname(save_path), exist_ok=True)
54          print(f"Saving preprocessed data to disk at {save_path}...")
55          np.save(f"{save_path}_X.npy", X)
56          np.save(f"{save_path}_y.npy", y)
57
58      print("Data loading complete.")
59      return X, y
```

Figure 33: Load_Data() Method: Extracting Images and Labels

Following, generic post-processing takes place, with a looping call to the extract_frames() method detailed in *Figure 31,* resulting in an accumulation of frames and labels for training use.

Another addition during development was progress statements. During initial runs, the terminal would display no information on the execution, leaving one guessing whether the program had frozen. This was alleviated by the addition of statements for the pre-processing of each gesture class, down to the image, resulting in a steady stream of progress information.

A major issue with this iteration mentioned at its introduction was the inability to cap storage space usage. An unreasonable amount of storage space was used due to *Line 19*'s inefficient runtime usage of space. This is shown in *Figure 35,* and was another reason why a major revision was required.

```
urn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
Starting data preprocessing...
Loading preprocessed data from disk...
Traceback (most recent call last):
  File "c:\Users\Erwin\Desktop\dissertation\main.py", line 20, in <module>
    X = X / 255.0  # Normalize pixel values
        ~~^~~~~~~
numpy.core._exceptions._ArrayMemoryError: Unable to allocate 173. GiB for an array with shape (63063, 30, 64, 64, 3) and data type float64
PS C:\Users\Erwin\Desktop\dissertation>
```

Figure 34: Unreasonable Storage Usage - Fatal Error

Thereafter, the results were saved to separate files ready for training, where a basic TensorFlow script was used with a ReLU CNN.

Main.py

This file brought together the pre-processed data generated with *preprocess.py* and the training model within *train.py*, to generate a classifier algorithm and a window similar to that of *SparshaSaha* in *Section* .

```
scripts > old > main.py > ...
1   import numpy as np
2   from sklearn.model_selection import train_test_split
3   from tensorflow.keras.utils import to_categorical
4   from tensorflow.keras.callbacks import ModelCheckpoint
5   from scripts.preprocess import load_data
6   from scripts.train import create_model
7   import cv2
8
9   extracted_path = 'extracted_data'
10  csv_path = 'dataset/hand_gesture_timing_stats.csv'
11  preprocessed_save_path = 'preprocessed_data/gesture_data'
12  model_save_path = 'saved_model/gesture_model.h5'
13  checkpoint_path = 'saved_model/checkpoint-{epoch:02d}-{val_accuracy:.2f}.h5'
14
15  print("Starting data preprocessing...")
16  X, y = load_data(extracted_path, csv_path, save_path=preprocessed_save_path)
17  X = X / 255.0
18  y = to_categorical(y, num_classes=11)
19  print("Data preprocessing complete.")
```

Figure 35: Loading Pre-Processed Data

Initially, the code in *Figure 36* was uninterrupted and led straight to preprocessing. However, as mentioned prior, the addition made to check whether a pre-processed batch of data already existed (to avoid unnecessary duplicate or overwritten pre-processing) continues to this file. The several statements prior to the first printed statement on *Line 15* ensure that the data is correctly referred to, and later in *Figure 36* the reference to the imported method load_data() confirms and executes this dependent on the state of the path object passed to the method. The number of classes used is correctly capped at 11 out of 27 for the third time in the code. Again, progress statements were added as development continued, and provide a useful insight into the inner workings of the script at all stages of runtime.

Following, *Figure 37* shows a middle excerpt that splits the data into training and testing sets using the *scikit-learn* module. The model is then created using the imported method from *train.py*, which was at that point a CNN based on *SparshaSaha's* project.

```
22  print("Splitting data into training and validation sets...")
23  X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
24  print("Data split complete.")
25
26
27  print("Creating the model...")
28  model = create_model(num_classes=11)
29
30  checkpoint = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_only=True, mode='max')
31  callbacks_list = [checkpoint]
32
33  print("Training the model...")
34  model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val), callbacks=callbacks_list)
35  print("Model training complete.")
36
37  print(f"Saving the model to {model_save_path}...")
38  model.save(model_save_path)
39  print("Model saved.")
40
41  print("Evaluating the model...")
42  loss, accuracy = model.evaluate(X_val, y_val)
43  print(f'Validation Accuracy: {accuracy * 100:.2f}%')
```

Figure 36: Splitting Data and Training

The script then begins training the model based on another imported method, using 10 epochs. At this stage, training was incomplete and the program broke, giving further credence to the idea of the next iteration, however due to the nature of the first iteration explaining all of the general code, this is included along with the next section on using OpenCV to open the window relevant to actual gesture recognition. Finally, the model would be saved and the evaluation metric generated.

Thereafter, *Figure 38* shows the call for a windows to be open for hand-gesture recognition - real-time preview of the camera input overlaid with the class prediction by the classifier. The exact environmental appearance will be revealed in the next iterations as iterative success allows execution at this final stage.

```
print("Starting real-time gesture recognition...")

def recognize_gesture(frame, model, img_size=(64, 64)):
    frame_resized = cv2.resize(frame, img_size)
    frame_normalized = frame_resized / 255.0
    frame_expanded = np.expand_dims(frame_normalized, axis=0)
    predictions = model.predict(frame_expanded)
    gesture_index = np.argmax(predictions)
    gesture_probability = np.max(predictions)
    return gesture_index, gesture_probability

cap = cv2.VideoCapture(0)
target_classes = ['class_01', 'class_02', 'class_11', 'class_12', 'class_16', 'class_17',
                  'class_18', 'class_20', 'class_21', 'class_25', 'class_27']

while True:
    ret, frame = cap.read()
    if not ret:
        break
    gesture_index, gesture_probability = recognize_gesture(frame, model)
    gesture_name = target_classes[gesture_index]
    cv2.putText(frame, f'{gesture_name}: {gesture_probability:.2f}', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2, cv2.LINE_AA)
    cv2.imshow('Gesture Recognition', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Figure 37: Gesture Recognition Windows

Until the selection of ‘q’ to quit, the window is continually regenerated on closure, leading to an uninterruptible process of hand gesture recognition. The integration of this keyboard command in the Unity Application would likely be difficult, and therefore to avoid massively increased complexity in controlling another program using uniquely mapped keyboard action in-application, a simple alt+tab command is recommended to switch windows.

A critical flaw of this iteration is that the pre-processing produces individual frames, which are more suitable for image classification as opposed to video classification – which becomes a very important difference in the distinction between static and *dynamic* hand gesture recognition. The later iterations focus more on dynamic hand gesture recognition, and pre-processing is likewise altered to do so.

8.7.2 Iteration 2 (Failure)

This iteration introduces a significant developmental leap, and introduces a proper *train.py* script exploring a LSTM model – generally recognised to be better with video-based classification.

During its development, the glaring weaknesses of the first iteration were revised to include specificity towards the project endeavour of *dynamic* hand gesture recognition, and the following changes were made in its consideration:

- Pre-processing now uses batches instead of saving the entire dataset at once (storage error).
- Pre-processing now creates sequences of frames for longer-form analysis.
- A sliding window was added to generate overlapping sequences of frames.
- Pixel normalisation was added.

- The CNN model was changed to a LSTM.

Despite the changes, a new fatal error was discovered during pre-processing that again inflicted upon storage usage, highlighted in *Figure 38.* This was improved upon and entirely removed as an error in the next iteration.

```
Processing class_27/User20_...
Processing class_27/User21_...
Saving batch 62 to disk at preprocessed_data/gesture_data...
Saving final batch 63 to disk at preprocessed_data/gesture_data...
Data loading complete.
Traceback (most recent call last):
  File "c:\Users\Erwin\Desktop\dissertation\main.py", line 19, in <module>
    X, y = load_data(extracted_path, save_path=preprocessed_save_path)
  File "c:\Users\Erwin\Desktop\dissertation\scripts\preprocess.py", line 73, in load_data
    return np.concatenate([np.load(f"{save_path}_X_{i}.npy", allow_pickle=True) for i in range(batch_count + 1)]), \
  File "c:\Users\Erwin\Desktop\dissertation\scripts\preprocess.py", line 73, in <listcomp>
    return np.concatenate([np.load(f"{save_path}_X_{i}.npy", allow_pickle=True) for i in range(batch_count + 1)]), \
  File "C:\Users\Erwin\anaconda3\envs\Dissertation\lib\site-packages\numpy\lib\npyio.py", line 456, in load
    return format.read_array(fid, allow_pickle=allow_pickle,
  File "C:\Users\Erwin\anaconda3\envs\Dissertation\lib\site-packages\numpy\lib\format.py", line 809, in read_array
    array = numpy.fromfile(fp, dtype=dtype, count=count)
numpy.core._exceptions._ArrayMemoryError: Unable to allocate 2.75 GiB for an array with shape (368640000,) and data type float64
PS C:\Users\Erwin\Desktop\dissertation>
```

Figure 38: Batch Save Error

No further images were added since these were relatively minute additions and modifications of the existing code shown in the previous iteration. However, the new *Train.py* file is shown below.

Train.py

Continuing in the theme of developmental revisions, the CNN in the first iteration was replaced with a LSTM in the second. Again, this choice of algorithm generally reigns superior in *dynamic* hand gesture recognition, while CNNs are superior in image classification and therefore static hand gesture recognition.

scripts > old2 > train.py

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, TimeDistributed, Conv2D, MaxPooling2D, Flatten, Dropout

def create_lstm_model(seq_length, img_size, num_classes):
    model = Sequential()
    model.add(TimeDistributed(Conv2D(32, (3, 3), activation='relu'), input_shape=(seq_length, img_size[0], img_size[1], 3)))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Flatten()))
    model.add(LSTM(64))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Figure 39: Early LSTM Model

8.7.3 Iteration 3 (Success)

More refined changes are made in this iteration, with a complete rehaul of the training method switching back to a CNN, leading to the first success in execution all the way to a hand-gesture recognition window.

- Pre-processing now uses a CSV file to determine specific frame ranges and user data.
- Label encoding is now used to handle class labels efficiently.
- Batch saving and memory management have been structured with explicit methods.
- Garbage collection is used to manage memory.
- Deeper CNN with three Conv2D layers and MaxPooling2D layers for better feature extraction.
- Training is now timed.

Because of the final milestone of the hand-recognition window, it is now relevant to release the epoch data. While previously only 10 epochs were selected, the number was increased to 25 for both an increase in accuracy and for experimental purposes. As seen in *Figure 40,* accuracy hovered around 99.3%, with no real change further. This could be an indicator of a poor algorithm, and this was noted for the next iteration. The total time required for training with 25 epochs was 2207.95 seconds, or 36 minutes and 48 seconds.

```
837/837 [==============================] - 89s 107ms/step - loss: 0.0255 - accuracy: 0.9914 - val_loss: 22.2701 - val_accuracy: 0.2822
Epoch 19/25
837/837 [==============================] - 87s 104ms/step - loss: 0.0205 - accuracy: 0.9926 - val_loss: 30.5667 - val_accuracy: 0.2526
Epoch 20/25
837/837 [==============================] - 88s 106ms/step - loss: 0.0243 - accuracy: 0.9918 - val_loss: 25.3191 - val_accuracy: 0.3550
Epoch 21/25
837/837 [==============================] - 90s 108ms/step - loss: 0.0223 - accuracy: 0.9929 - val_loss: 28.0136 - val_accuracy: 0.2402
Epoch 22/25
837/837 [==============================] - 94s 112ms/step - loss: 0.0174 - accuracy: 0.9947 - val_loss: 30.7749 - val_accuracy: 0.3321
Epoch 23/25
837/837 [==============================] - 88s 105ms/step - loss: 0.0249 - accuracy: 0.9919 - val_loss: 26.0093 - val_accuracy: 0.3960
Epoch 24/25
837/837 [==============================] - 88s 105ms/step - loss: 0.0225 - accuracy: 0.9932 - val_loss: 28.7185 - val_accuracy: 0.3347
Epoch 25/25
837/837 [==============================] - 85s 102ms/step - loss: 0.0216 - accuracy: 0.9930 - val_loss: 30.2251 - val_accuracy: 0.3800
Total training time: 2207.95 seconds
Saving the trained model...
Model saved.
```

Figure 40: 25 Epoch Training Data

In all, these changes culminated in a project more able to handle the detailed steps required in pre-processing, while also offering much greater flexibility to manipulate large datasets like the one used in this endeavour. However, the classifier algorithm proved unreliable and often incorrect.

Figure 41: 'OK' Gesture - Class 25

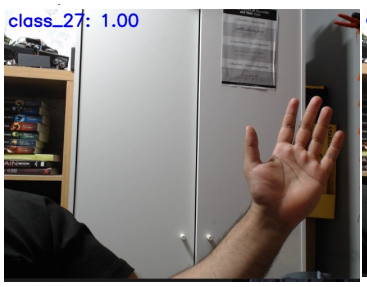


Figure 42: Right to Left - Class 1

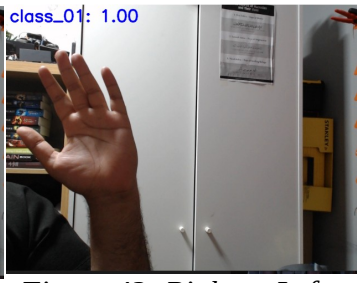


Figure 43: Right to Left - Class 1

As seen in *Figure 41,* a very simple *static* gesture on a relatively plain background proved incorrect, though the algorithm predicted with 100% accuracy. Concurrently, the single *dynamic* gesture shown in *Figures 42a and 42b* is predicted correctly. Therefore, it seems that this iteration had a preference towards dynamic gestures.

All gestures were assessed, and variable results were found in terms of accuracy. At this point, it was understood that alongside training the model, it was required to train myself in order to get the hand position and movement speed correct for the classifier to work properly. This is another obvious indication of a poor algorithm, but considering the niche of *dynamicism* against *static* gestures, it might be expected of an undergraduate dissertation.

There was also a significant lack of diagnostic ability, whereby it was difficult to understand what exactly the classifier was considering viable predictive stimulus. Therefore, the next and final iteration worked to resolve these issues.

8.8 Final Build

Continuing to the final iteration, a robust algorithm wasn't produced, but a reasonably well-documented revision sheet led to a working novel algorithm that steadily accepted much of the original 11 classes considered for this project. However, many curious issues were discovered during development, including an algorithmic preference for spatial position of the hand and a lighting preference. For further information about the classifier's action in real-time, several additions were introduced to the hand-gesture recognition system.

- Hand detection using HSV colour space and contours were added.
- Predictions are now stabilised (preventing from varying wildly) using chronological checks.
- A hand mask window has been added.
- A bounding box around the predictive stimulus was added.
- Predictions are now reset if no confident prediction is made for a certain period of time.
- Epochs were increased to 100 (for data-collection purposes).

With this final iteration, both the quality and consistency of the predictions by the classifier increased. Despite not harbouring commercial-level quality, some familiarity with the preferred regions of the classifier allows a reasonable degree of confidence in the algorithm's predictive capabilities.

Issues

The main problems with the final classifier occur in the following conditions: variable lighting, variable position from the camera, and no recognition of gestures 11, 12, 27.

Variable Lighting

Initial testing showed an incredible level of sensitivity to light. As shown in *Figure 45 and 46,* a bounding box appears on the most irrelevant of details within the camera's view. This detracts from the integrity of the algorithm, since it is coded to prevent recognition unless the algorithm is over 70% certain of a recognition.

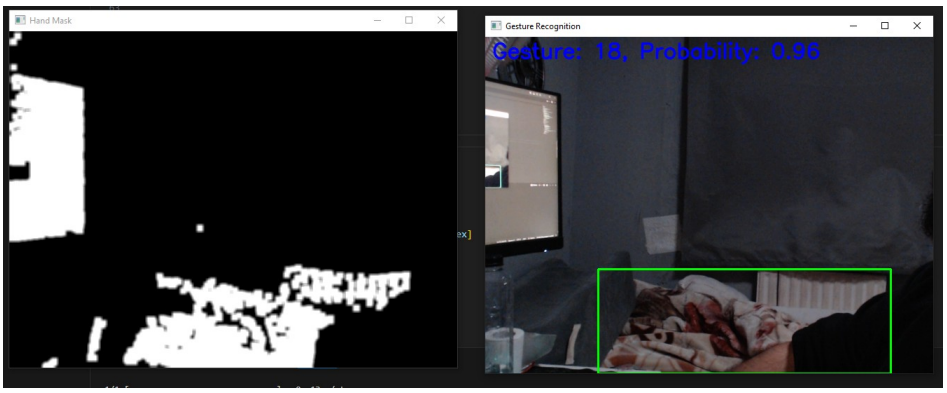


Figure 44: Blanket Detection

This continues regardless of coverage. *Figure 46* shows the hand mask on the left, recognising multiple anomalous objects in the webcam view that it treats like hands with a

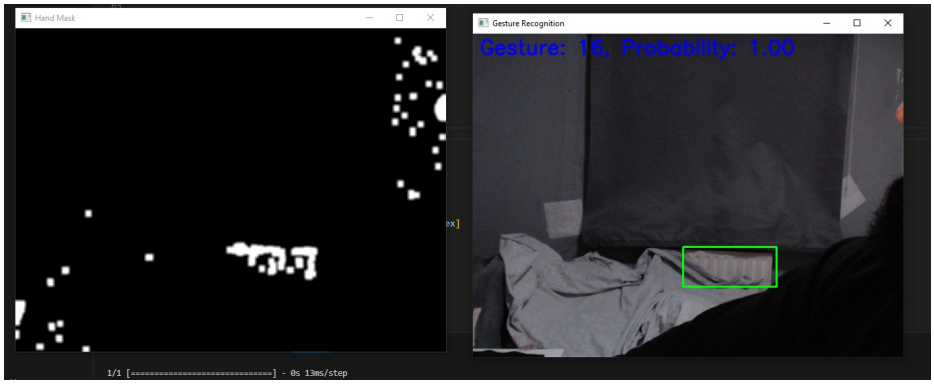


Figure 45: Sensitive Recognition

high degree of accuracy. However, when positioned correctly, dynamic hand gestures could work on this background.

In response, epochs were increased to 100. While this is an unusually high number, the metrics ended up showing that critical accuracy was achieved at around 20 epochs, while further training was useless for both data collection and performance metrics. Training time took 8907.12 seconds, or 2 hours 28 minutes and 27 seconds. An anomaly was detected at epoch 96, causing a loss metric of

```
837/837 [==============================] - 89s 106ms/step - loss: 0.0168 - accuracy: 0.9970 - val_loss: 47.7413 - val_accuracy: 0.2055
Epoch 93/100
837/837 [==============================] - 89s 106ms/step - loss: 0.0143 - accuracy: 0.9970 - val_loss: 45.6392 - val_accuracy: 0.2260
Epoch 94/100
837/837 [==============================] - 89s 107ms/step - loss: 0.0037 - accuracy: 0.9990 - val_loss: 44.1680 - val_accuracy: 0.2342
Epoch 95/100
837/837 [==============================] - 89s 106ms/step - loss: 0.0035 - accuracy: 0.9992 - val_loss: 44.6099 - val_accuracy: 0.2738
Epoch 96/100
837/837 [==============================] - 88s 106ms/step - loss: 2.4878e-04 - accuracy: 0.9999 - val_loss: 51.2045 - val_accuracy: 0.2777
Epoch 97/100
837/837 [==============================] - 89s 106ms/step - loss: 0.0221 - accuracy: 0.9953 - val_loss: 45.1805 - val_accuracy: 0.2570
Epoch 98/100
837/837 [==============================] - 89s 106ms/step - loss: 0.0061 - accuracy: 0.9982 - val_loss: 44.7745 - val_accuracy: 0.2679
Epoch 99/100
837/837 [==============================] - 88s 105ms/step - loss: 0.0119 - accuracy: 0.9974 - val_loss: 49.7924 - val_accuracy: 0.2408
Epoch 100/100
837/837 [==============================] - 88s 105ms/step - loss: 0.0027 - accuracy: 0.9994 - val_loss: 44.6576 - val_accuracy: 0.2710
Total training time: 8907.12 seconds
```

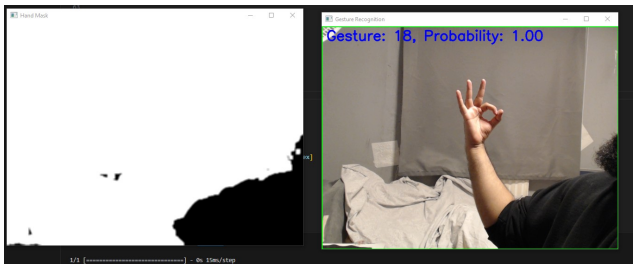


Figure 47: 3000K Light - Far

Figure 48: 3000K Light – Close

2.4878e-04, as shown in . Some residual effect was retained at epoch 97, with the loss metric skyrocketing to 0.0221, but returned to normal averages at epoch 98. The full training data for 100 epochs is available to view in *Appendix A* at the end of the document.

Variable Position from Camera

Interestingly, the hue of the light also mattered in hand recognition. It was only at 3000K, at a mid position that the easiest gesture in the selected group ('OK' gesture – class 25) would work. The following *Figures 47, 48, 49* show failures to recognise in 6500K light, while *Figures 50 and 52* fail. The highest success rate came with *Figure 51*, which used 3000K light at a medium distance from the camera. These results are despite the fact that 3000K light completely filled the hand mask.

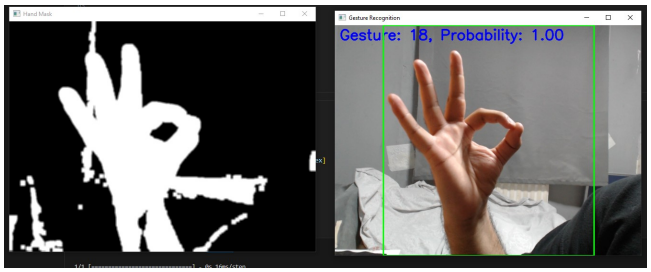


Figure 49: 6500K Light - Close

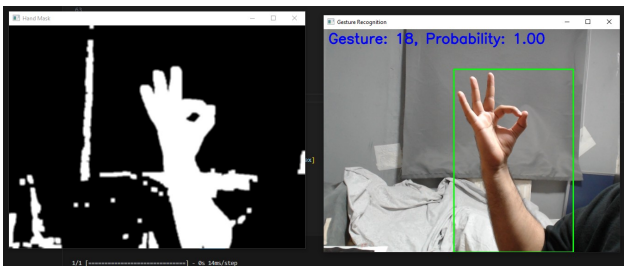


Figure 50: 6500K Light - Mid

All three positional checkpoints failed more often than not in 6500K lighting conditions. While the bounding box accurately captured the hand,

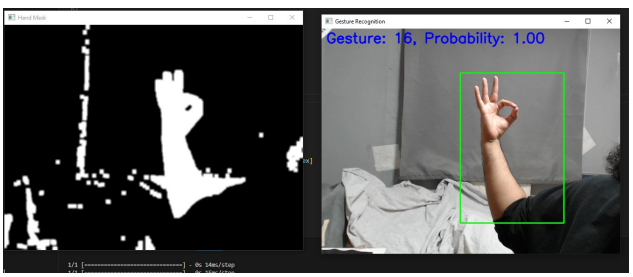


Figure 51: 6500K Light - Far

predictions were unreliable and often varied between correct and incorrect. However, the hand mask displays a very clear contrast between the coloured target and the general background. While great effort was made to reduce the amplitude of colour differentials in the background, the hand mask retains great sensitivity to excessively outstanding difference in light. Additionally, there was no difference in accuracy when the sleeve was pulled to full-length when considered against a rolled-up sleeve.

When compared to 3000K lighting conditions, the difference is surprisingly large, but results in a far more favourable result across all three positions despite the hand mask's obscurity. *Figure 52* is the position in which the current configuration of both classifier algorithm and environment seem to give reliable feedback on the static gesture of *Class 25*, and therefore this was used to test further. Additionally, this system of recognition preferential to this algorithm took me, its producer, about 30 minutes to familiarise myself with. This included speed of dynamic gestures (120 frames @ 60fps) and start/end positions based on the videos included in the dataset. The figures in this section do not show the dynamic gesture testing, since capturing dynamic movement isn't possible without a dynamic media capture format. However, classes 1, 2, 16, 17, 18, 20, 25 worked with the 3000K 'mid' lighting/position configuration.

Figure 52: 3000K Light - Mid

Considering the impressive size of the dataset, classification accuracy is expected to be much more robust (Rajput et al., 2023). However, performance was limited to the conditions described. There remains space for deeper investigations in future revisions of this endeavour.

8.9 Final Environment

The final iteration of the working environment retains some of the original libraries, with the underlying CNN algorithm complete.

The final dependency list (manually installed) is as follows:

- Python 3.9.18
- TensorFlow 2.10
- opencv-python 4.10
- pandas 2.2.1
- pillow 10.3
- scikit-learn 1.4.2
- h5py 3.11

The remaining dependencies were installed either as additions to the main dependencies, or were automatically installed through some other method. Regardless, the full dependency list was retrieved through the '*conda list*' command in the activated virtual environment:

```
# packages in environment at C:\Users\Erwin\anaconda3\envs\Dissertation:
#
# Name                    Version                   Build  Channel
_tflow_select             2.3.0                       mkl
abseil-cpp                20211102.0           hd77b12b_0
absl-py                   2.1.0              py39haa95532_0
aiohttp                   3.9.5              py39h2bbff1b_0
aiosignal                 1.2.0                pyhd3eb1b0_0
astunparse                1.6.3                      py_0
async-timeout             4.0.3              py39haa95532_0
attrs                     23.1.0             py39haa95532_0
blas                      1.0                         mkl
blinker                   1.6.2              py39haa95532_0
bottleneck                1.3.7              py39h9128911_0
brotli-python             1.0.9              py39hd77b12b_8
c-ares                    1.19.1               h2bbff1b_0
ca-certificates           2024.3.11            haa95532_0
cachetools                5.3.3              py39haa95532_0
certifi                   2024.6.2           py39haa95532_0
cffi                      1.16.0             py39h2bbff1b_1
charset-normalizer        2.0.4                pyhd3eb1b0_0
click                     8.1.7              py39haa95532_0
colorama                  0.4.6              py39haa95532_0
cryptography              41.0.3             py39h3438e0d_0
flatbuffers               2.0.0                h6c2663c_0
freetype                  2.12.1               ha860e81_0
frozenlist                1.4.0              py39h2bbff1b_0
gast                      0.4.0                pyhd3eb1b0_0
giflib                    5.2.1                h8cc25b3_3
google-auth               2.29.0             py39haa95532_0
google-auth-oauthlib      0.4.4                pyhd3eb1b0_0
google-pasta              0.2.0                pyhd3eb1b0_0
grpc-cpp                  1.48.2               hf108199_0
grpcio                    1.48.2             py39hf108199_0
h5py                      3.11.0             py39hed405ee_0
hdf5                      1.12.1               h51c971a_3
icc_rt                    2022.1.0             h6049295_2
icu                       58.2                 ha925a31_3
idna                      3.7                py39haa95532_0
importlib-metadata        7.0.1              py39haa95532_0
intel-openmp              2023.1.0           h59b6b97_46320
joblib                    1.4.0              py39haa95532_0
jpeg                      9e                   h2bbff1b_1
keras                     2.10.0             py39haa95532_0
keras-preprocessing       1.1.2                pyhd3eb1b0_0
lcms2                     2.12                 h83e58a3_0
lerc                      3.0                  hd77b12b_0
libcurl                   8.7.1                h86230a5_0
libdeflate                1.17                 h2bbff1b_1
libpng                    1.6.39               h8cc25b3_0
libprotobuf               3.20.3               h23ce68f_0
libssh2                   1.10.0               hcd4344a_2
libtiff                   4.5.1                hd77b12b_0
libwebp-base              1.3.2                h2bbff1b_0
lz4-c                     1.9.4                h2bbff1b_1
markdown                  3.4.1              py39haa95532_0
markupsafe                2.1.3              py39h2bbff1b_0
mkl                       2023.1.0           h6b88ed4_46358
mkl-service               2.4.0              py39h2bbff1b_1
mkl_fft                   1.3.8              py39h2bbff1b_0
```

Figure 53: Final Dependency List (Part 1)

```
mkl_random                1.2.4              py39h59b6b97_0
multidict                 6.0.4              py39h2bbff1b_0
numexpr                   2.8.7              py39h2cd9be0_0
numpy                     1.26.4             py39h055cbcc_0
numpy-base                1.26.4             py39h65a83cf_0
oauthlib                  3.2.2              py39haa95532_0
opencv-python             4.10.0.82                pypi_0    pypi
openjpeg                  2.4.0                h4fc8c34_0
openssl                   1.1.1w               h2bbff1b_0
opt_einsum                3.3.0                pyhd3eb1b0_1
packaging                 23.2               py39haa95532_0
pandas                    2.2.1              py39h5da7b33_0
pillow                    10.3.0             py39h2bbff1b_0
pip                       24.0               py39haa95532_0
protobuf                  3.20.3             py39hd77b12b_0
pyasn1                    0.4.8                pyhd3eb1b0_0
pyasn1-modules            0.2.8                      py_0
pybind11-abi              5                    hd3eb1b0_0
pycparser                 2.21                 pyhd3eb1b0_0
pyjwt                     2.8.0              py39haa95532_0
pyopenssl                 23.2.0             py39haa95532_0
pysocks                   1.7.1              py39haa95532_0
python                    3.9.18               h6244533_0
python-dateutil           2.9.0post0         py39haa95532_2
python-flatbuffers        2.0                  pyhd3eb1b0_0
python-tzdata             2023.3               pyhd3eb1b0_0
pytz                      2024.1             py39haa95532_0
re2                       2022.04.01           hd77b12b_0
requests                  2.32.2             py39haa95532_0
requests-oauthlib         1.3.0                      py_0
rsa                       4.7.2                pyhd3eb1b0_1
scikit-learn              1.4.2              py39h4ed8f06_1
scipy                     1.13.1             py39h8640f81_0
setuptools                69.5.1             py39haa95532_0
six                       1.16.0               pyhd3eb1b0_1
snappy                    1.1.10               h6c2663c_1
sqlite                    3.45.3               h2bbff1b_0
tbb                       2021.8.0             h59b6b97_0
tensorboard               2.10.0             py39haa95532_0
tensorboard-data-server   0.6.1              py39haa95532_0
tensorboard-plugin-wit    1.8.1              py39haa95532_0
tensorflow                2.10.0           mkl_py39ha510bab_0
tensorflow-base           2.10.0           mkl_py39h6a7f48e_0
tensorflow-estimator      2.10.0             py39haa95532_0
termcolor                 2.1.0              py39haa95532_0
threadpoolctl             2.2.0                pyh0d69192_0
typing_extensions         4.11.0             py39haa95532_0
tzdata                    2024a                h04d1e81_0
urllib3                   2.2.1              py39haa95532_0
vc                        14.2                 h2eaa2aa_1
vs2015_runtime            14.29.30133          h43f2093_3
werkzeug                  3.0.3              py39haa95532_0
wheel                     0.43.0             py39haa95532_0
win_inet_pton             1.1.0              py39haa95532_0
wrapt                     1.14.1             py39h2bbff1b_0
xz                        5.4.6                h8cc25b3_1
yarl                      1.9.3              py39h2bbff1b_0
zipp                      3.17.0             py39haa95532_0
zlib                      1.2.13               h8cc25b3_1
zstd                      1.5.5                hd43e919_2
```

Figure 54: Final Dependency List (Part 2)

8.10 Dynamic Hand Gesture Recognition's Niche

Throughout this dissertation, it became apparent to me that the field of dynamic hand gesture recognition was severely under-researched when compared to other machine learning areas. This may be because of limited commercial interest in the field. The top 4 out of 7 most cited papers in 2019 were image recognition themed (*Google Scholar*, 2019), and the entirety of the most popular AI research papers in 2023 were dominated by LLMs and image recognition (Yao, 2023), which shows that when considered against the boom of LLMs and image recognition algorithms as the

flagships of AI's global reknown, hand gesture recognition falls out of the limelight in terms of current practicality and funding. This is likely the reason why almost no viable classifier algorithms were available for my specific purpose, with the addition of the 'dynamic' modifier further reducing results into unusability.

For any future endeavour into the field of strictly *dynamic* hand-gesture recognition, there is a strong recommendation of experience with building image classification algorithms alongside an understanding of time-analysis evaluation.

9 Discussion

The general purposes of this dissertation include the development of a novel application for data analysis, and an unexpected concurrent development of a novel algorithm for dynamic hand gesture recognition, necessitated by the lack of such projects in the field. This dissertation was conducted with constant shifts in research and implementation priority as I came to understand the topic of research more deeply.

The high-level objectives of this dissertation were as follows:

- **Investigation** – Investigate the pathway to developing a standalone application for visual data analysis and the classifier algorithm in tandem with which it would become intuitive.

- **Development** – The development of a novel application for visual data analysis, and the acquisition of a classifier algorithm.

- **Iterative Improvement –** the continual documented improvement of each portion of the research project over time.

These objectives were achieved through thorough exploration of the research area and significant experimentation. The acquisition of a classifier algorithm was instead developed as a novel addition to the research area. As a result, the integration between the application and the classifier algorithm was detailed, but not implemented.

The problems with this project were discussed concurrently with the development, leading to a chronologically accurate document that reflects the journey of iterative development I made during the writing of this dissertation.

While extraordinary advances are being made in both machine learning and deep learning fronts, I believe the value of HGR systems remains unseen, or irrelevant at this point in view of the laser focus on neural networks. While replicating the brain's miraculous processing power is certainly an objective to attribute great time and funding to, there is a clear use case for increasing the rate of flow of information between the human and the computer, which on a larger scale could not only cause tangible positive shifts in individual productivity, but could cause the interweaving of some basal multitasking instinct that could make computer navigation seamless, easy and uninterruptible, even for elders. After all, we experience this 3D world far more than the virtual.

10 Conclusion

While hand-gesture recognition within the niche of visual data analysis is not currently touted as the most modern technology at the forefront of funding and technological advances (Mohamed, Mustafa and Jomhari, 2021), the recent advancements in machine learning and image classification present fertile grounds for further research in this area.

The main objective of this project was to develop and deploy an application for visual data analysis with a classifier algorithm that would classify real-time hand-gestures and map them to in-application interactions. To achieve this goal, a novel application was developed in the Unity Engine, and a CNN-based classifier algorithm was developed.

Throughout the development of the classifier algorithm, two major algorithmic choices were explored for training the machine learning model: CNN and LSTM. After several iterations, the CNN outperformed the LSTM both in reliability and in classification at the novel development stage, though this was likely due to poor developmental practices since LSTMs generally reign superior with video classification (Wu et al., 2018). Up to 100 epochs were tested with the final CNN environment, and it was discovered that maximal efficiency and accuracy was achieved by 20 epochs using the current configuration and dataset.

Of the 27-class dataset, 11 gestures were chosen for use within the Unity application for the purposes of data analysis. The purpose of each of these gestures were detailed, and though implementation was not sought due to the unexpected development of the novel classifier algorithm, diagrams were drawn detailing the requirements and ensuing project design, allowing ease for readers and future authors to continue the research started in this dissertation.

To conclude, development and research into dynamic hand gesture recognition within the field of visual data analysis purports to offer crucial change to the fast-changing landscape of data in general. With the recent bloom in AI and big data analysis, more efficient methodologies are constantly sought, and data manipulation using a more efficacious and natural input method than the conventional promises to present ease and comfort in future use for generations ahead.

Reference list

Alexey Kapitanov, Makhlyarchuk, A. and Kvanchiani, K. (2022). HaGRID - Hand Gesture Recognition Image Dataset. *arXiv (Cornell University)*. doi:https://doi.org/10.48550/arxiv.2206.08219.

Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., Nayak, T., Andreopoulos, A., Garreau, G., Mendoza, M., Kusnitz, J., Debole, M., Esser, S., Delbruck, T., Flickner, M. and Modha, D. (2017). A Low Power, Fully Event-Based Gesture Recognition System. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. doi:https://doi.org/10.1109/cvpr.2017.781.

Bergen, B.K. (2019). Do Gestures Retain Mental Associations with Their Iconic origins, Even after They Become emblematic? an Analysis of the middle-finger Gesture among American English Speakers. *PLOS ONE*, 14(4), p.e0215633. doi:https://doi.org/10.1371/journal.pone.0215633.

Cerfoglio, S., Ferraris, C., Vismara, L., Amprimo, G., Priano, L., Pettiti, G., Galli, M., Mauro, A. and Cimolin, V. (2022). Kinect-Based Assessment of Lower Limbs during Gait in Post-Stroke Hemiplegic Patients: A Narrative Review. *Sensors*, 22(13), p.4910. doi:https://doi.org/10.3390/s22134910.

Chang, V., Rahman Olamide Eniola, Golightly, L. and Xu, Q. (2023). An Exploration into Human–Computer Interaction: Hand Gesture Recognition Management in a Challenging Environment. *SN computer science*, [online] 4(5). doi:https://doi.org/10.1007/s42979-023-01751-y.

Devineau, G., Moutarde, F., Xi, W. and Yang, J. (2018). *Deep Learning for Hand Gesture Recognition on Skeletal Data*. [online] IEEE Xplore. doi:https://doi.org/10.1109/FG.2018.00025.

Fronteddu, G., Porcu, S., Floris, A. and Atzori, L. (2022). A Dynamic Hand Gesture Recognition Dataset for Human-Computer Interfaces. *Computer Networks*, 205, p.108781. doi:https://doi.org/10.1016/j.comnet.2022.108781.

Gao, Q., Chen, Y., Ju, Z. and Liang, Y. (2022). Dynamic Hand Gesture Recognition Based on 3D Hand Pose Estimation for Human–Robot Interaction. *IEEE Sensors Journal*, [online] 22(18), pp.17421–17430. doi:https://doi.org/10.1109/JSEN.2021.3059685.

Gibran Benítez-García, Jesús Olivares-Mercado, Sánchez-Pérez, G. and Yanai, K. (2020). IPN Hand: A Video Dataset and Benchmark for Real-Time Continuous Hand Gesture Recognition. *arXiv (Cornell University)*. doi:https://doi.org/10.48550/arxiv.2005.02134.

Gyulten Hyusein and Tilbe Göksun (2023). The creative interplay between hand gestures, convergent thinking, and mental imagery. *The Creative Interplay between Hand gestures, Convergent thinking, and Mental Imagery*, 18(4), pp.e0283859–e0283859. doi:https://doi.org/10.1371/journal.pone.0283859.

Ma, X. and Peng, J. (2018). Kinect Sensor-Based Long-Distance Hand Gesture Recognition and Fingertip Detection with Depth Information. *Journal of Sensors*, 2018, pp.1–9. doi:https://doi.org/10.1155/2018/5809769.

Nuzzi, C., Pasinetti, S., Pagani, R., Coffetti, G. and Sansoni, G. (2021). HANDS: an RGB-D dataset of static hand-gestures for human-robot interaction. *Data in Brief*, 35, p.106791. doi:https://doi.org/10.1016/j.dib.2021.106791.

Ophelia, M. and Keerthijith, P. (2018). Licensed under Creative Commons Attribution CC BY Survey on Static and Dynamic Hand Gesture Recognition Techniques. *International Journal of Science and Research (IJSR) ResearchGate Impact Factor*. doi:https://doi.org/10.21275/ART20196946.

Oudah, M., Al-Naji, A. and Chahl, J. (2020). Hand Gesture Recognition Based on Computer Vision: A Review of Techniques. *Journal of Imaging*, 6(8), p.73. doi:https://doi.org/10.3390/jimaging6080073.

Rayhan, A. and Gross, D. (2023). The Rise of Python: A Survey of Recent Research. *The Rise of Python: A Survey of Recent Research*. doi:https://doi.org/10.13140/RG.2.2.27388.92809.

Sharma, N., Jain, V. and Mishra, A. (2018). An Analysis Of Convolutional Neural Networks For Image Classification. *Procedia Computer Science*, 132, pp.377–384. doi:https://doi.org/10.1016/j.procs.2018.05.198.

stanfordonline (2020). *Lecture 1 - Welcome | Stanford CS229: Machine Learning (Autumn 2018)*. *YouTube*. Available at: https://www.youtube.com/watch?v=jGwO_UgTS7I.

Tan, Y.S., Lim, K.M. and Lee, C.P. (2021). Hand gesture recognition via enhanced densely connected convolutional neural network. *Expert Systems with Applications*, 175, p.114797. doi:https://doi.org/10.1016/j.eswa.2021.114797.

Thioux, M., Gazzola, V. and Keysers, C. (2008). Action Understanding: How, What and Why. *Current Biology*, 18(10), pp.R431–R434. doi:https://doi.org/10.1016/j.cub.2008.03.018.

Zhang, Q., Zhu, W. and Zhu, Q. (2019). Real World Hand Gesture Interaction in Virtual Reality. *Journal of Physics: Conference Series*, 1229, p.012027. doi:https://doi.org/10.1088/1742-6596/1229/1/012027.

Bhushan, S., Alshehri, M., Keshta, I., Chakraverti, A. K., Rajpurohit, J., & Abugabah, A. (2022). An Experimental Analysis of Various Machine Learning Algorithms for Hand Gesture Recognition. *Electronics*, *11*(6), 968. https://doi.org/10.3390/electronics11060968

Tang, H., Liu, H., Xiao, W., & Sebe, N. (2019). Fast and robust dynamic hand gesture recognition via key frames extraction and feature fusion. *Neurocomputing*, *331*, 424–433. https://doi.org/10.1016/j.neucom.2018.11.038

Google Scholar reveals its most influential papers for 2019. (2019, August 2). Nature Index. https://www.nature.com/nature-index/news/google-scholar-reveals-most-influential-papers-research-citations-twenty-nineteen

Wu, Y., Zheng, B., & Zhao, Y. (2018, November 1). *Dynamic Gesture Recognition Based on LSTM-CNN*. IEEE Xplore. https://doi.org/10.1109/CAC.2018.8623035

He, C., Chrastil, E. R., & Hegarty, M. (2022). A new psychometric task measuring spatial perspective taking in ambulatory virtual reality. *Frontiers in Virtual Reality*, *3*. https://doi.org/10.3389/frvir.2022.971502

Rajput, D., Wang, W.-J., & Chen, C.-C. (2023). Evaluation of a decided sample size in machine learning applications. *BMC Bioinformatics*, *24*(1). https://doi.org/10.1186/s12859-023-05156-9

Appendix A: 100 Epoch Training Data

Epoch 1/100

837/837 [==============================] - 93s 111ms/step - loss: 0.8705 - accuracy: 0.6574 - val_loss: 6.9444 - val_accuracy: 0.2517

Epoch 2/100

837/837 [==============================] - 91s 109ms/step - loss: 0.3601 - accuracy: 0.8452 - val_loss: 8.8826 - val_accuracy: 0.2238

Epoch 3/100

837/837 [==========] - 91s 108ms/step - loss: 0.2339 - accuracy: 0.8970 - val_loss: 11.1461 - val_accuracy: 0.1976

Epoch 4/100

837/837 [==========] - 91s 108ms/step - loss: 0.1635 - accuracy: 0.9329 - val_loss: 11.0058 - val_accuracy: 0.2574

Epoch 5/100

837/837 [==========] - 91s 108ms/step - loss: 0.1217 - accuracy: 0.9499 - val_loss: 12.5107 - val_accuracy: 0.2063

Epoch 6/100

837/837 [==========] - 88s 105ms/step - loss: 0.0964 - accuracy: 0.9624 - val_loss: 13.0148 - val_accuracy: 0.2710

Epoch 7/100

837/837 [==========] - 87s 104ms/step - loss: 0.0721 - accuracy: 0.9717 - val_loss: 15.2043 - val_accuracy: 0.2230

Epoch 8/100

837/837 [==========] - 89s 107ms/step - loss: 0.0646 - accuracy: 0.9757 - val_loss: 16.3825 - val_accuracy: 0.2335

Epoch 9/100

837/837 [==========] - 91s 108ms/step - loss: 0.0541 - accuracy: 0.9797 - val_loss: 14.6690 - val_accuracy: 0.2625

Epoch 10/100

837/837 [==========] - 89s 106ms/step - loss: 0.0476 - accuracy: 0.9832 - val_loss: 17.8554 - val_accuracy: 0.2253

Epoch 11/100

837/837 [==========] - 89s 106ms/step - loss: 0.0389 - accuracy: 0.9863 - val_loss: 16.0533 - val_accuracy: 0.3063

Epoch 12/100

837/837 [==========] - 91s 108ms/step - loss: 0.0393 - accuracy: 0.9868 - val_loss: 16.1900 - val_accuracy: 0.2734

Epoch 13/100

837/837 [==========] - 90s 108ms/step - loss: 0.0346 - accuracy: 0.9885 - val_loss: 19.0121 - val_accuracy: 0.3164

Epoch 14/100

837/837 [==========] - 90s 108ms/step - loss: 0.0250 - accuracy: 0.9907 - val_loss: 18.6331 - val_accuracy: 0.2871

Epoch 15/100

837/837 [==========] - 86s 102ms/step - loss: 0.0325 - accuracy: 0.9893 - val_loss: 19.4381 - val_accuracy: 0.2613

Epoch 16/100

837/837 [==========] - 88s 105ms/step - loss: 0.0285 - accuracy: 0.9906 - val_loss: 21.1267 - val_accuracy: 0.2531

Epoch 17/100

837/837 [==========] - 89s 106ms/step - loss: 0.0278 - accuracy: 0.9905 - val_loss: 19.2641 - val_accuracy: 0.2622

Epoch 18/100

837/837 [==========] - 90s 107ms/step - loss: 0.0209 - accuracy: 0.9934 - val_loss: 19.8258 - val_accuracy: 0.2798

Epoch 19/100

837/837 [==========] - 89s 106ms/step - loss: 0.0204 - accuracy: 0.9939 - val_loss: 24.2738 - val_accuracy: 0.2770

Epoch 20/100

837/837 [==========] - 90s 107ms/step - loss: 0.0264 - accuracy: 0.9920 - val_loss: 21.2679 - val_accuracy: 0.2317

Epoch 21/100

837/837 [==========] - 87s 104ms/step - loss: 0.0221 - accuracy: 0.9933 - val_loss: 25.6836 - val_accuracy: 0.3073

Epoch 22/100

837/837 [==========] - 89s 106ms/step - loss: 0.0148 - accuracy: 0.9950 - val_loss: 26.4794 - val_accuracy: 0.3350

Epoch 23/100

837/837 [==========] - 89s 106ms/step - loss: 0.0205 - accuracy: 0.9936 - val_loss: 25.9742 - val_accuracy: 0.2015

Epoch 24/100

837/837 [==========] - 90s 108ms/step - loss: 0.0224 - accuracy: 0.9931 - val_loss: 24.1261 - val_accuracy: 0.2976

Epoch 25/100

837/837 [==========] - 90s 107ms/step - loss: 0.0191 - accuracy: 0.9941 - val_loss: 25.2150 - val_accuracy: 0.2009

Epoch 26/100

837/837 [==========] - 89s 106ms/step - loss: 0.0168 - accuracy: 0.9952 - val_loss: 23.3132 - val_accuracy: 0.2211

Epoch 27/100

837/837 [==========] - 89s 107ms/step - loss: 0.0169 - accuracy: 0.9949 - val_loss: 25.9884 - val_accuracy: 0.3242

Epoch 28/100

837/837 [==========] - 89s 106ms/step - loss: 0.0200 - accuracy: 0.9937 - val_loss: 25.5714 - val_accuracy: 0.2773

Epoch 29/100

837/837 [==========] - 89s 107ms/step - loss: 0.0088 - accuracy: 0.9970 - val_loss: 29.3867 - val_accuracy: 0.2915

Epoch 30/100

837/837 [==========] - 89s 106ms/step - loss: 0.0141 - accuracy: 0.9958 - val_loss: 23.6108 - val_accuracy: 0.2900

Epoch 31/100

837/837 [==========] - 89s 106ms/step - loss: 0.0160 - accuracy: 0.9951 - val_loss: 30.9229 - val_accuracy: 0.2550

Epoch 32/100

837/837 [==========] - 89s 106ms/step - loss: 0.0195 - accuracy: 0.9953 - val_loss: 27.8905 - val_accuracy: 0.2211

Epoch 33/100

837/837 [==========] - 89s 106ms/step - loss: 0.0217 - accuracy: 0.9940 - val_loss: 32.2479 - val_accuracy: 0.2822

Epoch 34/100

837/837 [==========] - 89s 107ms/step - loss: 0.0094 - accuracy: 0.9968 - val_loss: 30.2939 - val_accuracy: 0.2263

Epoch 35/100

837/837 [==========] - 90s 107ms/step - loss: 0.0151 - accuracy: 0.9959 - val_loss: 29.4406 - val_accuracy: 0.1834

Epoch 36/100

837/837 [==========] - 89s 107ms/step - loss: 0.0148 - accuracy: 0.9958 - val_loss: 28.2982 - val_accuracy: 0.2466

Epoch 37/100

837/837 [==========] - 88s 105ms/step - loss: 0.0100 - accuracy: 0.9971 - val_loss: 32.6597 - val_accuracy: 0.2311

Epoch 38/100

837/837 [==========] - 89s 106ms/step - loss: 0.0112 - accuracy: 0.9969 - val_loss: 29.3640 - val_accuracy: 0.2149

Epoch 39/100

837/837 [==========] - 89s 107ms/step - loss: 0.0174 - accuracy: 0.9955 - val_loss: 25.5599 - val_accuracy: 0.2402

Epoch 40/100

837/837 [==========] - 88s 106ms/step - loss: 0.0083 - accuracy: 0.9979 - val_loss: 31.0582 - val_accuracy: 0.2405

Epoch 41/100

837/837 [==========] - 89s 106ms/step - loss: 0.0229 - accuracy: 0.9944 - val_loss: 26.7153 - val_accuracy: 0.2145

Epoch 42/100

837/837 [==========] - 88s 105ms/step - loss: 0.0069 - accuracy: 0.9978 - val_loss: 28.9280 - val_accuracy: 0.2389

Epoch 43/100

837/837 [==========] - 89s 106ms/step - loss: 0.0138 - accuracy: 0.9959 - val_loss: 30.8552 - val_accuracy: 0.2300

Epoch 44/100

837/837 [==========] - 89s 107ms/step - loss: 0.0129 - accuracy: 0.9959 - val_loss: 30.6473 - val_accuracy: 0.2311

Epoch 45/100

837/837 [==========] - 88s 105ms/step - loss: 0.0056 - accuracy: 0.9987 - val_loss: 32.8349 - val_accuracy: 0.2329

Epoch 46/100

837/837 [==========] - 89s 107ms/step - loss: 0.0142 - accuracy: 0.9965 - val_loss: 26.1524 - val_accuracy: 0.2199

Epoch 47/100

837/837 [==========] - 89s 107ms/step - loss: 0.0158 - accuracy: 0.9957 - val_loss: 31.3500 - val_accuracy: 0.2395

Epoch 48/100

837/837 [==========] - 88s 105ms/step - loss: 0.0047 - accuracy: 0.9985 - val_loss: 29.1591 - val_accuracy: 0.2395

Epoch 49/100

837/837 [==========] - 90s 107ms/step - loss: 0.0207 - accuracy: 0.9941 - val_loss: 29.6365 - val_accuracy: 0.2439

Epoch 50/100

837/837 [==========] - 88s 105ms/step - loss: 0.0055 - accuracy: 0.9982 - val_loss: 35.7788 - val_accuracy: 0.1915

Epoch 51/100

837/837 [==========] - 89s 106ms/step - loss: 0.0072 - accuracy: 0.9981 - val_loss: 38.2227 - val_accuracy: 0.2667

Epoch 52/100

837/837 [==========] - 87s 104ms/step - loss: 0.0276 - accuracy: 0.9935 - val_loss: 32.8271 - val_accuracy: 0.2546

Epoch 53/100

837/837 [==========] - 90s 107ms/step - loss: 0.0043 - accuracy: 0.9986 - val_loss: 36.7248 - val_accuracy: 0.2882

Epoch 54/100

837/837 [==========] - 89s 106ms/step - loss: 0.0095 - accuracy: 0.9974 - val_loss: 37.5499 - val_accuracy: 0.2272

Epoch 55/100

837/837 [==========] - 89s 106ms/step - loss: 0.0093 - accuracy: 0.9973 - val_loss: 31.2106 - val_accuracy: 0.2590

Epoch 56/100

837/837 [==========] - 89s 107ms/step - loss: 0.0073 - accuracy: 0.9976 - val_loss: 34.0163 - val_accuracy: 0.1915

Epoch 57/100

837/837 [==========] - 88s 106ms/step - loss: 0.0134 - accuracy: 0.9958 - val_loss: 32.5987 - val_accuracy: 0.3211

Epoch 58/100

837/837 [==========] - 89s 107ms/step - loss: 0.0207 - accuracy: 0.9952 - val_loss: 35.3750 - val_accuracy: 0.2701

Epoch 59/100

837/837 [==========] - 89s 106ms/step - loss: 0.0082 - accuracy: 0.9977 - val_loss: 37.3271 - val_accuracy: 0.2680

Epoch 60/100

837/837 [==========] - 89s 106ms/step - loss: 0.0056 - accuracy: 0.9983 - val_loss: 37.3793 - val_accuracy: 0.3093

Epoch 61/100

837/837 [==========] - 90s 107ms/step - loss: 0.0062 - accuracy: 0.9980 - val_loss: 37.0451 - val_accuracy: 0.3190

Epoch 62/100

837/837 [==========] - 89s 107ms/step - loss: 0.0141 - accuracy: 0.9963 - val_loss: 42.3631 - val_accuracy: 0.2271

Epoch 63/100

837/837 [==========] - 90s 108ms/step - loss: 0.0073 - accuracy: 0.9981 - val_loss: 43.1308 - val_accuracy: 0.2280

Epoch 64/100

837/837 [==========] - 90s 107ms/step - loss: 0.0135 - accuracy: 0.9967 - val_loss: 39.0986 - val_accuracy: 0.2405

Epoch 65/100

837/837 [==========] - 89s 106ms/step - loss: 0.0093 - accuracy: 0.9977 - val_loss: 45.6062 - val_accuracy: 0.2302

Epoch 66/100

837/837 [==========] - 90s 107ms/step - loss: 0.0207 - accuracy: 0.9957 - val_loss: 38.9505 - val_accuracy: 0.2626

Epoch 67/100

837/837 [==========] - 89s 106ms/step - loss: 0.0073 - accuracy: 0.9984 - val_loss: 40.6371 - val_accuracy: 0.2652

Epoch 68/100

837/837 [==========] - 88s 105ms/step - loss: 0.0118 - accuracy: 0.9968 - val_loss: 42.4125 - val_accuracy: 0.2241

Epoch 69/100

837/837 [==========] - 90s 108ms/step - loss: 0.0136 - accuracy: 0.9966 - val_loss: 38.2990 - val_accuracy: 0.2484

Epoch 70/100

837/837 [==========] - 88s 106ms/step - loss: 0.0092 - accuracy: 0.9983 - val_loss: 36.9036 - val_accuracy: 0.2798

Epoch 71/100

837/837 [==========] - 88s 105ms/step - loss: 0.0084 - accuracy: 0.9983 - val_loss: 40.2203 - val_accuracy: 0.2508

Epoch 72/100

837/837 [==========] - 89s 106ms/step - loss: 0.0136 - accuracy: 0.9969 - val_loss: 44.6369 - val_accuracy: 0.2218

Epoch 73/100

837/837 [==========] - 89s 107ms/step - loss: 0.0091 - accuracy: 0.9979 - val_loss: 44.3371 - val_accuracy: 0.2407

Epoch 74/100

837/837 [==========] - 89s 107ms/step - loss: 0.0146 - accuracy: 0.9966 - val_loss: 41.0346 - val_accuracy: 0.2694

Epoch 75/100

837/837 [==========] - 90s 107ms/step - loss: 0.0030 - accuracy: 0.9991 - val_loss: 40.7757 - val_accuracy: 0.2758

Epoch 76/100

837/837 [==========] - 90s 107ms/step - loss: 0.0059 - accuracy: 0.9985 - val_loss: 46.7774 - val_accuracy: 0.1546

Epoch 77/100

837/837 [==========] - 89s 107ms/step - loss: 0.0166 - accuracy: 0.9963 - val_loss: 36.3408 - val_accuracy: 0.2614

Epoch 78/100

837/837 [==========] - 89s 107ms/step - loss: 0.0018 - accuracy: 0.9995 - val_loss: 39.8291 - val_accuracy: 0.1949

Epoch 79/100

837/837 [==========] - 88s 105ms/step - loss: 0.0154 - accuracy: 0.9962 - val_loss: 39.7164 - val_accuracy: 0.2167

Epoch 80/100

837/837 [==========] - 89s 107ms/step - loss: 0.0073 - accuracy: 0.9978 - val_loss: 42.0896 - val_accuracy: 0.3033

Epoch 81/100

837/837 [==========] - 89s 106ms/step - loss: 0.0130 - accuracy: 0.9969 - val_loss: 45.7411 - val_accuracy: 0.2356

Epoch 82/100

837/837 [==========] - 90s 108ms/step - loss: 0.0048 - accuracy: 0.9990 - val_loss: 41.3587 - val_accuracy: 0.2022

Epoch 83/100

837/837 [==========] - 88s 105ms/step - loss: 0.0135 - accuracy: 0.9973 - val_loss: 39.3027 - val_accuracy: 0.2102

Epoch 84/100

837/837 [==========] - 89s 106ms/step - loss: 0.0081 - accuracy: 0.9987 - val_loss: 41.2483 - val_accuracy: 0.1963

Epoch 85/100

837/837 [==========] - 89s 107ms/step - loss: 0.0094 - accuracy: 0.9980 - val_loss: 50.6557 - val_accuracy: 0.1263

Epoch 86/100

837/837 [==========] - 90s 108ms/step - loss: 0.0156 - accuracy: 0.9964 - val_loss: 47.6634 - val_accuracy: 0.2556

Epoch 87/100

837/837 [==========] - 89s 106ms/step - loss: 0.0053 - accuracy: 0.9986 - val_loss: 47.9948 - val_accuracy: 0.2102

Epoch 88/100

837/837 [==========] - 88s 106ms/step - loss: 0.0283 - accuracy: 0.9947 - val_loss: 43.9353 - val_accuracy: 0.2963

Epoch 89/100

837/837 [==========] - 89s 107ms/step - loss: 0.0052 - accuracy: 0.9988 - val_loss: 51.2394 - val_accuracy: 0.2535

Epoch 90/100

837/837 [==========] - 89s 107ms/step - loss: 0.0064 - accuracy: 0.9991 - val_loss: 48.8371 - val_accuracy: 0.2759

Epoch 91/100

837/837 [==========] - 89s 106ms/step - loss: 0.0079 - accuracy: 0.9981 - val_loss: 44.9972 - val_accuracy: 0.3025

Epoch 92/100

837/837 [==========] - 89s 106ms/step - loss: 0.0168 - accuracy: 0.9970 - val_loss: 47.7413 - val_accuracy: 0.2055

Epoch 93/100

837/837 [==========] - 89s 106ms/step - loss: 0.0143 - accuracy: 0.9970 - val_loss: 45.6392 - val_accuracy: 0.2260

Epoch 94/100

837/837 [==========] - 89s 107ms/step - loss: 0.0037 - accuracy: 0.9990 - val_loss: 44.1680 - val_accuracy: 0.2342

Epoch 95/100

837/837 [==========] - 89s 106ms/step - loss: 0.0035 - accuracy: 0.9992 - val_loss: 44.6099 - val_accuracy: 0.2738

Epoch 96/100

837/837 [==========] - 88s 106ms/step - loss: 2.4878e-04 - accuracy: 0.9999 - val_loss: 51.2045 - val_accuracy: 0.2777

Epoch 97/100

837/837 [==========] - 89s 106ms/step - loss: 0.0221 - accuracy: 0.9953 - val_loss: 45.1805 - val_accuracy: 0.2570

Epoch 98/100

837/837 [==========] - 89s 106ms/step - loss: 0.0061 - accuracy: 0.9982 - val_loss: 44.7745 - val_accuracy: 0.2679

Epoch 99/100

837/837 [==========] - 88s 105ms/step - loss: 0.0119 - accuracy: 0.9974 - val_loss: 49.7924 - val_accuracy: 0.2408

Epoch 100/100

837/837 [==========] - 88s 105ms/step - loss: 0.0027 - accuracy: 0.9994 - val_loss: 44.6576 - val_accuracy: 0.2710

Total training time: 8907.12 seconds

Saving the trained model...

Model saved.

Appendix B: Dissertation Proposal

Middlesex University
CST3990

Project Proposal
12/11/23

Machine Perception: Advancing Visual Data Analysis through Gesture-Based Interaction

1 Project Description

The field of visual data analysis via human-computer interaction (HCI) has remained commercially stagnant since the genesis of Virtual Reality (VR) and Augmented Reality (AR). Advancements in various computer science disciplines since, have re-ignited interest in this field. Progression has been limited to academic and developmental scopes, where considerable improvements in brain-computer interfaces (BCI) and rich 3D visualisations edge us towards a more intuitive control over ever-more powerful machines. This project aims to prototype a refinement to existing human-computer interaction through the use of gestures within both a native OS desktop, and a chosen data-rich sandbox application. The prototype proposed will conform to the growing reliance on ease-of-accessibility and convenience to provide intuitive options for gesture-based interaction that may even take precedence over the default keyboard and mouse input method. The resulting application aims to natively and intuitively navigate a native Windows or Linux desktop, and a chosen sandbox application. Despite a lack of steely clarity, the decidedly cautious nature of this proposal aims to allow modifications to the project wherever required as the period for implementation nears. For this reason, a decision crossroads has been added to sections where pathway definitions are visibly unset. As a whole, this proposal will provide a skeletal definition of the work to be carried out by the end of the academic year, with the purpose of developing an easily implementable, intuitive gesture-recognition software to assume basic control of your machine without the requirement for a conventional keyboard and mouse.

Supervisor
Dr. Clifford DeRaffaele
Middlesex University

Author
Abdur-Rehman Nasir
M00729406

Table of Contents

2 Deliverables

This project consists of four sections, underpinned by three required courseworks. Each section, except for the final, has included a decision crossroads which details the possible pathways considered for that particular section. Beyond each decision crossroads lies a brief outline of the deliverable, the documentation to record during it, and the progression expected.

2.1 Coursework 1: Project Proposal (Nov 12th)

2.2 Development Environment (Nov – Dec)

Creating a comfortable development environment lays unshakeable foundations for the duration of a project. Visualising how best to initialise one, requires scanning every section underpinning the project with appropriate foresight. Incompatible software dependencies and volumising add-ons in an effort to integrate at a later stage, is counterproductive to developmental efficiency.

2.2.1 Decision Crossroads (Multiple Pathways)

At a decision stage, various options were available for this academic endeavour.

- **The Leap Motion**:
  - A small, flatly designed peripheral that reads the 3D space opposite to it via two monochromatic cameras (Johnny Phung, 1:37), allowing hand-gestures to be mathematically interpreted and mapped to on-screen controls.
  - Unfortunately, this exciting device didn't fit this project's requirements. Despite having boasted technology ahead of its time, and a now-defunct open-area for developers' apps including full company support ('Ultraleap', 2017), I believe a broader scope of functionality and user-preferential improvement lies in a machine learning and modelling equivalent to the Leap Motion's now dated use of mathematical frame comparison.

- **Microsoft Kinect**:
  - A sensor bar released in 2010 as a peripheral to the Xbox 360, the Kinect used infrared light depth-sensing technologies such as time-of-flight measurements (akin to sonar) to deduce the entirety of a 3D space in the view of the sensor, including persons within it and their complex movements (Sarbolandi, Lefloch and Kolb, 2015).
  - Realising the Kinect's advertised potential required processing power far above its capabilities, and so it remains at base a developmentally inept device. It had an early discontinuation due to lacklustre adoption, and due to its outdated hardware, better technology can be made available through simple cameras.

- **Arduino**/**Pi**:
  - Both options are extremely well-documented, with full community support, vast peripheral connection availability and each lead their relative fields.
  - While the technology itself is undoubtedly reliable, the hardware in particular falls short for the purposes of this academic endeavour.

- **AR**/**VR**:
  - Due to the complexity of this realm, and its difficulty of both implementation and replication, I chose not to engage in AR/VR development.

Developmental opportunities for hand-gestures with any pairing of these devices are undoubtedly exciting, especially AR/VR and the Leap Motion controller. However, a critical component of success in an individual project is knowing where your skills end, and the time constraints begin to squeeze. Therefore, I've chosen a TensorFlow/OpenCV pairing to maximise time spent developing a tangible output instead of continuing to setup my development environment well into December/January.

2.2.2 Current Decision

My final meeting conducted with Dr. DeRaffaele confirmed the use of the following technologies:

- **TensorFlow**:
  - A free and open-source software library for machine learning and artificial intelligence that can be utilised alongside a sizeable dataset to train a gesture-recognition model.
  - Alternatives to this include Google's Teachable, or Meta AI's Pytorch.

- **OpenCV**:
  - With no further integration than the hardware-installed (or peripheral) webcam on your device, this library allows interpretation of real-time computer vision. Allowing the processing power of the PC to be used to interpret gestures alongside an easily accessible, simple camera device will maximise the academic value of the project.
  - While OpenCV is coded in C++, language bindings are available for Python, Java and MATLAB. My language of choice remains Java, and so JavaCV is an optional addition, but considering the differences do not generally affect language fluency, this remains open to confirmation till function trials.

The rest of November will be used to become comfortable with each application within the environment, through the use of courses, exemplar models and relevant research. I've added the option for an additional foray into early December in case integration with a test dataset seems more complicated than it seems.

Documentation:
Technologies used and why.
Training methodology in preparation for dataset work.
Read and note-take on relevant literature.

2.3 Create a basic gesture recognition model (Dec):

This stage involves the acquisition of an appropriately comprehensive dataset, and the training of a model on it via TensorFlow. Once training is complete, this model is to be tested with OpenCV integration under a webcam. The prototype produced here can contribute to the literature review's 'initial steps' component.

2.3.1 Decision Crossroads (Dataset Creation)

At some point, I considered the creation of my own dataset as a staple of achievement that could apply a stamp of comprehensive originality to the creation of the final prototype. This proved to be a task well beyond the scope of this endeavour, and also the general time advisory for the dissertation (300 hours). I will outline below why this was unattainable for the project, before moving on to the progression expected at this point.

Pictures

Possibly the most enticing part of the dataset creation process, your own hand can serve as a model to begin to grow a dataset. Images should be collected from multiple angles and under different lighting conditions for robustness. However, the quantity of images required at minimum is enormous - a commonly suggested beginning for a single class (gesture) is 1000. Therefore, at minimum, a set of ten simple hand gestures will require 10,000 images. The number of unique pictures required can be significantly reduced by data augmentation methods (rotating, scaling, flipping etc.), and so at this stage, this became a high yet achievable standard for the project.

Validation Dataset

Since the model cannot confirm for itself what it's seeing, a validation dataset is required for an unbiased estimation of the model's performance. This is suggested to be 20% of the total dataset's size, and so in the case of 10,000 images, the validation dataset will total 2000 images. Is is then obvious that these are annotated and labelled manually by the dataset creator, which leads us to the downfall of this idea.

Labelling and Annotating

Assuming 10,000 images over ten gestures, the latter of which is a reasonable expectation for this project, and assuming that labelling and annotating is also done manually in the vein of the validation dataset by the creator of it, the time taken exceeds the boundaries of this project. Averaging 1 minute per image, over 10,000 images, would require >160 hours of continuous, uninterrupted work. This does not include edge cases and subsequent re-training, dataset errors and the extreme likelihood of further image necessity. Finally, discussing this with Dr. DeRaffaele confirmed that use of publicly available datasets was a more beneficial use of the time commitment of the project.

2.3.2 Current Decision

Many hand-gesture specific datasets are available with >500,000 images preprocessed and ready to train on a model. However, Dr. DeRaffaele suggested implementing a subset of these, appropriate to the use cases of this project. Therefore, the focus will remain solely on gestures that can be applied efficiently and intuitively to both a native Windows or Linux environment and within a sandbox application. The requirements for these gestures will find bases in intuitive, easily replicable movements with minimal chance for misrepresentation. An optional, but highly recommended base would be designing the subset such that transitions from one gesture to an (or any) other is both intuitive and seamless.

Training a model on TensorFlow or alternatives using an acquired dataset-subset remains the general purpose of December's section of development. Thereafter, integration with OpenCV will culminate in a complete prototype with basic, unoptimised gesture recognition. Successful webcam tests will draw eligibility for comprehensive literature review documentation

Documentation:
Process of dataset acquisition.
Model training process.
Testing stage.
Optimisations and difficulties.

2.4 Coursework 2: Literature Review (Jan 14th)

Submitting the literature review alone is not the purpose of this coursework. Substantial developments and progress updates should be included, alongside initial steps for a software. As mentioned previously, I plan to submit a prototype of the hand-gesture recognition model as early as this coursework.

2.5 Application Development and Integration (Jan – March):

The project dependencies centre around using intuitive hand-gesturing in two environments: Windows or Linux desktop, and a sandbox application. During the development of this proposal, data-rich and educational programs were the focus for making available intuitive hand-gesture control. Development on these is still a possibility, but a wider, more exciting opportunity. was offered by Dr. DeRaffaele.

2.5.1 Decision Crossroads (Application Choice)

The initial premise attracted suggestions for development on an existing application, such that a previously rendered environment with embedded data could be analysed. I briefly explored the creation of my own simulation, but struck it down due to difficulty. However, Dr. DeRaffaele supported the idea, and the development of my own application within the Unity game engine. In 2.5.2, I detail the concept of such a simulation. Other potential applications I came across span the breadth of the educational simulation genre, with some especially interesting examples in the following:

- **Simbad Astronomy**
  - An extremely detailed astronomical navigator for academic use cases, made publicly available. Offers intuitive controlled analysis of data on interstellar matter and celestial bodies, usually paired with astronomical spectroscopy. Data-rich environment.
- **EarthSci 3D**
  - Another extremely detailed geophysics tool to integrate and visualise seismic datasets under a 3D model of the Earth. Complete with a native window to view relevant publications and sources, and also to view raw data for manual analysis.
- **Anatomyou VR**
  - Viewed from the perspective of a variably sized camera travelling in the human body, the user is able to view organs, hematology, orthology and other structures at a microscopic level. Navigation in here would be akin to a game, and the applications for medical education for individuals with attention-deficit disorders would be astounding. However, the developmental environment is within VR, which is outside the limitations suggested. In addition, there are likely no particular hand gestures required outside of navigation tools.

Whereas previously, the creation of an original tool struck as too difficult a task, the suggestion that it could be done within the scope of this project is a prospect that could combine some of the elements of the previously researched games.

2.5.2 Current Decision

As a base requirement, the hand-gesture recognition should allow mouse-free navigation to the application to fulfil the first project dependency – that of navigation in a Windows or Linux desktop. As a minimum, a click gesture, with a clear and unequivocal option for double-click, should form the fundamentals of this environment. Additionally, a continuous gesture with real-time tracking should be mapped to the cursor for

navigation. This movements suggests a choice with minimal potential to be misinterpreted. Initialisation of this concept should be possible by late-January.

The second project dependency relies on the creation of an original application to analyse datasets with hand-gesture navigation. The concept of this application revolves around viewing multivariate datasets as collapsible and expandable points in 3D space, with the option of grouping points together, categorising and labelling them, along with other organisational tools. An additional implementation could be to reflect these changes in the original dataset and return a set of categorised and labelled data using some method. Within the application, available gestures would be dependent on the subset allocated from the larger acquired dataset in the previous stages. Repetitive gestures, such that multiple uses would be required as a fundamental point of the application, will be given priority for developing seamlessness. Other niche uses could involve double-handed gestures, such as zooming, shifting positions within the space, changing camera orientation, or even manipulating two data points at once.

Further research is required with respect to the development and implementation of this application. Unity as a game engine is understandably well-documented, but a higher production-quality alternative comes in the form of Unreal Engine. Either is worth consideration for the application-based uses case suggested, but imposed on both are a strict level of time-condition. The application itself should be fully functional by late March at the maximum, which gives a dedicated development time of around 2 months. The final week should be spent integrating the application with the dataset, ± a week.

At this stage, a level of comfort with the dataset should be achieved. Assessing any further additions, as well as optimisations is recommended. Where the model requires re-training, this is also suggested.

Documentation:
Process of program creation for desktop navigation.
Process of program creation for application navigation.
Integration between TensorFlow and the application engine.
Interpretation of results and translation into the application environment.
Mapping between the native application controls and the camera feed.
Choices made during development process.
Performance metrics.
Hand gesture additions & why.
Improvements & Optimisations.
Difficulties and re-evaluations of chosen pathways.

2.6 Final Refinements and Report Write-up (March – April):

By the end of March, the project prototype should be fully functional. This means both the native desktop environment and the created application should be working with the selected hand gestures, generally bug-free.

For the final stretch, any and all optimisations will be employed to speed up the real-time tracking. Beyond this, the issues to deal with will be case-specific. I have allocated the final month of April to iron out the final details, as well as to collate the documentation thus far into a single dissertation.

Documentation:
Optimisation techniques.
Strengths & weaknesses of programs and model.
Pathways unable to be traversed, and alternatives chosen.
Developmental failures.

2.7 *Coursework 3: Final Report* (April 28th)

3 Problem Addressed

As humans become more accustomed to efficiency and convenience while simultaneously dealing with larger and more data-rich problems, the mouse and keyboard as input devices remain painstaking to use. We often find significant delays between a thought, and its input into a computer, interred by tragically slow input devices, exacerbated by age, or unfamiliarity. The introduction of VR and AR represented a rebirth of interest in this field, but both endeavours remain commercially lacklustre. The purpose of this project is underpinned by the motive of bridging the gap between our higher processing and output capabilities, and the very limited input bandwidth of the conventional controllers of computers. By the end of the project, I will have rediscovered the foundations of intuitive and natural-bodied application control, and prototyped not only an easy-to-use substitute to conventional computer control, but also an application sandbox to upload and intuitively analyse multivariate datasets within.

4 Self-Evaluation

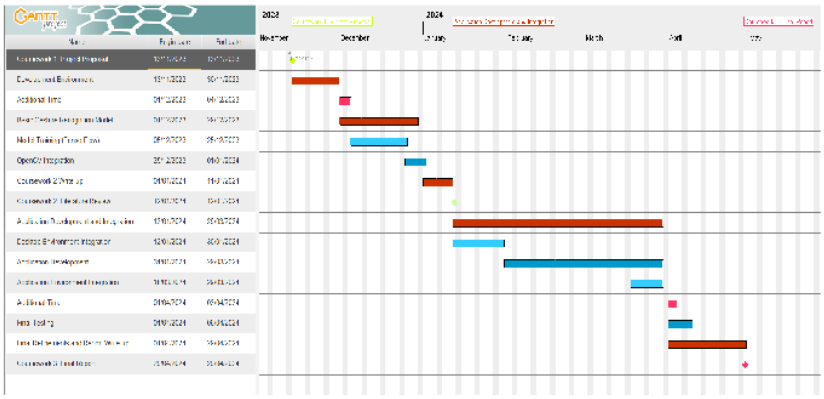
For the duration of the project, I will view my work as purely a striving endeavour. This means that every discovery or difficulty of reasonable effect will be documented as part of the development process. This could mean dataset acquisition difficulties, optimisation discoveries, program creation difficulties, mapping discoveries etc. Included in this, would be the potential use of alternate pathways in case of extreme resistance. These challenges deserve documentation, as they represent a vitally overlooked sect of learning that lays fertile grounds for further development, if the resistance is overcome.

Nearer the end, I will reflect on the knowledge I acquired, and how I may have initialised differently given a slightly more experienced hindsight. I will cover my own personal development throughout the project, and possible improvements to the project as a whole – especially areas that were completely under my control, and perhaps areas where my own shortcomings led to comparatively unimpressive results.

5 Resource Requirement

Laptop
OpenCV
TensorFlow
Dataset
Eye Tracker (optional)

6 Gantt Chart



Created in 'GanttProject' - BarD Software s.r.o.

7 Relevant Publications

Reference List:

Phung, J. (2013) *Leap motion teardown in macro HD.* 27 July.
Available at: https://www.youtube.com/watch?v=uF0NSUmxFYA (Accessed: 4 November 2023)

Ultraleap (2017) *App store retirement FAQS.* 30 June.
Available at:
https://support.leapmotion.com/hc/en-us/articles/360004321998-App-Store-Retirement-FAQ
(Accessed: 4 November 2023)

Sarbolandi, H., Lefloch, D., Kolb, A. (2015) 'Kinect range sensing: Structured-light versus time-of-flight Kinect', *Sciencedirect*. doi:10.1016/j.cviu.2015.05.006

Relevant Publications:

Chalmers, C. A. (2003) 'The role of cognitive theory in human-computer interface', *Sciencedirect.* doi:10.1016/S0747-5632(02)00086-9

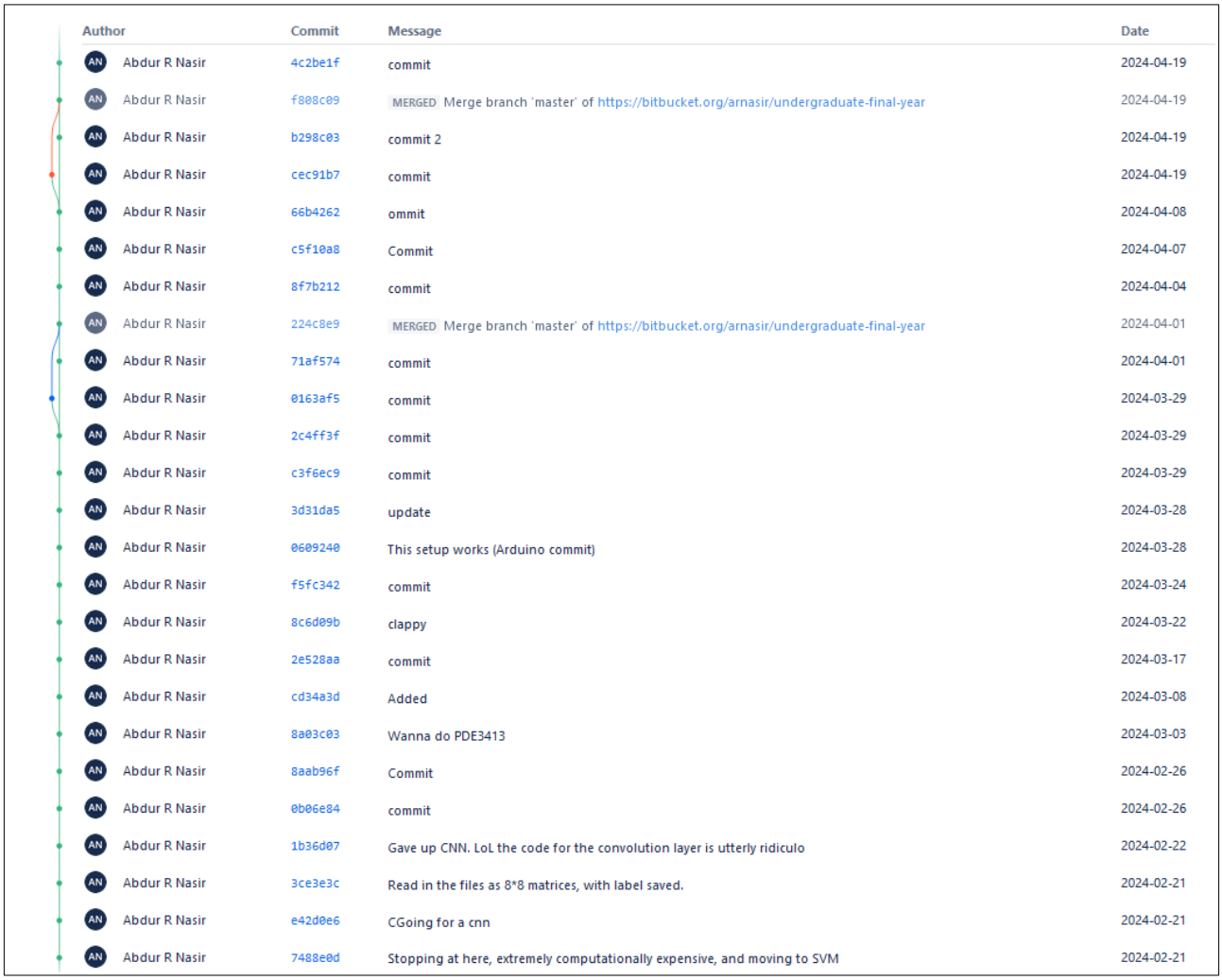
George, L., Lecuyer, A. (2010) 'An overview of research on "passive" brain-computer interfaces for implicit human-computer interaction', *HAL Open Science*.
Available at: https://inria.hal.science/inria-00537211 (PDF)

Modelo3d (2020) *Modelo with Leap Motion.* 19 Jan.
Available at: https://www.youtube.com/shorts/9fj96xXL1u8 (Accessed: 10 November 2023)

This project will use Harvard referencing.
https://www.scribbr.co.uk/referencing/harvard-style/
https://www.scribbr.co.uk/referencing/harvard-bibliography/

Appendix C: Git Evidence

Author	Commit	Message	Date
Abdur R Nasir	9c90cfd	commit	2 hours ago
Abdur R Nasir	af79be0	commit	14 hours ago
Abdur R Nasir	a30f13c	commit	16 hours ago
Abdur R Nasir	ae8b057	commit	18 hours ago
Abdur R Nasir	2e5d60c	commit	2 days ago
Abdur R Nasir	1564273	commit	2 days ago
Abdur R Nasir	b81acb0	commit	3 days ago
Abdur R Nasir	54abbd4	Completed failure algos	4 days ago
Abdur R Nasir	0359c2b	comit	5 days ago
Abdur R Nasir	492055c	Working on dissertation, and also re-exported Dissertation Proposal as higher quality document for Appendix additio...	5 days ago
Abdur R Nasir	2c5cae5	dissertation	7 days ago
Abdur R Nasir	929afee	coimmit	2024-06-07
Abdur R Nasir	d00d10d	some notes for dissertation, WIP	2024-06-07
Abdur R Nasir	2d6ba4f	dissertation plan for the next 20 days	2024-06-02
Abdur R Nasir	a8ad0a8	Final PDE3413 commit complete. Congrats!	2024-05-26
Abdur R Nasir	5f3109a	nearly complete	2024-05-22
Abdur R Nasir	0cbeb86	Commit	2024-05-22
Abdur R Nasir	6dcec7a	commit	2024-05-19
Abdur R Nasir	51d0e07	commit	2024-05-19
Abdur R Nasir	52147f1	commit	2024-05-18
Abdur R Nasir	66c45e2	commit	2024-05-18
Abdur R Nasir	a3fc1be	commit	2024-05-13
Abdur R Nasir	8dd5e87	arbaaz	2024-04-26
Abdur R Nasir	8fcea15	expense	2024-04-26
Abdur R Nasir	152a361	commit	2024-04-19



Author	Commit	Message	Date
Abdur R Nasir	4c2be1f	commit	2024-04-19
Abdur R Nasir	f808c09	MERGED Merge branch 'master' of https://bitbucket.org/arnasir/undergraduate-final-year	2024-04-19
Abdur R Nasir	b298c03	commit 2	2024-04-19
Abdur R Nasir	cec91b7	commit	2024-04-19
Abdur R Nasir	66b4262	ommit	2024-04-08
Abdur R Nasir	c5f10a8	Commit	2024-04-07
Abdur R Nasir	8f7b212	commit	2024-04-04
Abdur R Nasir	224c8e9	MERGED Merge branch 'master' of https://bitbucket.org/arnasir/undergraduate-final-year	2024-04-01
Abdur R Nasir	71af574	commit	2024-04-01
Abdur R Nasir	0163af5	commit	2024-03-29
Abdur R Nasir	2c4ff3f	commit	2024-03-29
Abdur R Nasir	c3f6ec9	commit	2024-03-29
Abdur R Nasir	3d31da5	update	2024-03-28
Abdur R Nasir	0609240	This setup works (Arduino commit)	2024-03-28
Abdur R Nasir	f5fc342	commit	2024-03-24
Abdur R Nasir	8c6d09b	clappy	2024-03-22
Abdur R Nasir	2e528aa	commit	2024-03-17
Abdur R Nasir	cd34a3d	Added	2024-03-08
Abdur R Nasir	8a03c03	Wanna do PDE3413	2024-03-03
Abdur R Nasir	8aab96f	Commit	2024-02-26
Abdur R Nasir	0b06e84	commit	2024-02-26
Abdur R Nasir	1b36d07	Gave up CNN. LoL the code for the convolution layer is utterly ridiculo	2024-02-22
Abdur R Nasir	3ce3e3c	Read in the files as 8*8 matrices, with label saved.	2024-02-21
Abdur R Nasir	e42d0e6	CGoing for a cnn	2024-02-21
Abdur R Nasir	7488e0d	Stopping at here, extremely computationally expensive, and moving to SVM	2024-02-21

Document Styling Parameters

This document was written in Times New Roman, with the following parameters.

Default Paragraph Style: Font 12

Heading 1: Font 15

Heading 2: Font 13

Heading 3: Font 13

In-line Section Reference: Italic

Figures:

- *Wrap: Optimal*
- *Padding: 0.2cm synced*

Appendix A: Font 9

Appendix B:

- *60% size*
- *90dpi*
- *Border: black, hairline width (0.05mm)*