Classifying British Sign Language Fingerspelling Using Machine Learning

UNIVERSITY OF
LINCOLN

Ben Gallard-Bridger
GAL19693078

19693078@students.lincoln.ac.uk

School of Computer Science

College of Science

University of Lincoln

Submitted in partial fulfilment of the requirements for the
Degree of MComp Computer Science

Supervisor Dr. John Atanbori

May 2023

Abstract

There are tens of thousands of people in the United Kingdom who cannot speak verbally. These people need to use sign language to communicate with others, however, only 6% of the population are able to communicate using British Sign Language (BSL) leading to those who cannot, being unable to access services. The purpose of this project is to create a piece of software that can correctly translate BSL fingerspelling, with the aim of being able to allow people who use BSL to communicate with those who do not know BSL.

This project document provides insight into the process of the entire project, including the research, development, and testing of a machine learning model which is able to classify British sign language fingerspelling at an accuracy of roughly 16%, although performance is highly accurate within live testing. In addition to this, this document provides an analysis of current literature, giving insight into the background and rationale for this project, along with informing stages of the development of the model.

Table of Contents

List of Figures

List of Tables

List of Source Codes

Chapter 1

Introduction

1.1 Scope and Rationale

In the United Kingdom, 16.3% of people suffer from some kind of hearing loss, which equates to about 11 million people. In addition to this, only 6% of people in the UK know more than two words in British Sign Language (BSL). With 87,000 people who are deaf and use BSL to be able to communicate, it means these people can only communicate with 0.2% of people in the country. This can lead to such people being unable to access essential services such as healthcare, public transport and grocery shopping (Ben Jaab, 2021; Central Digital & Data Office, 2017).

This project was undertaken to solve this issue. This project delved into the possibility of creating a computer vision/machine learning application which allows for the translation of BSL fingerspelling. The main aim, developing a model which has the capability to correctly classify BSL fingerspelling through the use of hand skeleton pretext, where the skeleton of the hand is extracted prior to classification. A fully working machine learning/computer vision application would enable people who are 'hard of hearing' (people who struggle to listen to external stimuli when compared to the general population) to be able to communicate with a larger body of people without the need for a human translator or other support methods. This was the goal of the project and the results are discussed further throughout this document, namely in the conclusion and reflective analysis.

In British sign language, there are over 1,800 signs for different words, when compared to the number of words in the English dictionary over 170 thousand words.

Although there might only be 1,800 signs, some signs have multiple meanings, allowing for the possibility to sign more words than just 1,800. This would mean that without fingerspelling, a person who uses BSL to communicate would only be able to use nearly one-hundredth of the total words that someone would understand. Due to this, the system will only be developed for use with fingerspelling to allow for all words to be communicated. It would be wise to state that although BSL fingerspelling would enable the use of any word in the English dictionary, BSL has many signs for singular and compound words, and as such these signs are used more frequently over fingerspelling, but given the time frame of this project, this smaller subset is chosen.

1.2 Document Structure

This document provides an analysis of the development of a machine learning model with the capabilities of classifying British Sign Language (BSL) fingerspelling (section 5). With this, an in-depth review of the current literature provides background and rationale for the project (section 2), whilst providing insight into the methods used within the project phases (section 4).

In addition to the above, this document discusses the outcomes of the project, informing about how this led to further development and changes throughout the process (section 6). Finally, this project document provides a conclusion to the project and advises on how the project could have been improved on in a reflective analysis of the entire process (sections 7 and 8).

1.3 Aims and Objectives

There were two aims for this project: To develop a machine learning model which is able to classify British sign language and to use this model in a developed system which processes live videos (through individual frames), showing the translation of BSL.

The objectives of the primary aim for this project were:

1. Find/Create a dataset of videos with a minimum of 100 images per letter by January 2022, as machine learning techniques require examples.
2. Research and implement five different pretext methods, to see which improves the accuracy of classification, by February 2023.
3. Train at least three different models on the data, training on each differing pretext method by March 2023. To allow for the best machine learning algorithm to be chosen for the system. Refine the most accurate model to make it as accurate as possible.

The objectives of the secondary aim for this project were:

1. Design an interface for the model, creating a Microsoft PowerPoint slideshow, showing how the pages will interact by mid-March 2023.
2. Develop an interface for the model, showing the text produced alongside the video by April 2023. To allow for more people to use the system effectively.
3. Blackbox test the interface, creating a file of testing tables along with the outcomes by mid-April 2023.

Chapter 2

Literature Review

2.1 Background and Rationale

Out of the 67 million people in the United Kingdom, there are 11 million people who suffer from hearing loss (Ben Jaab, 2021; Central Digital & Data Office, 2017), which is nearly one in six people or around 16%. In addition, there are 87 thousand people who are deaf, using British Sign Language (BSL) regularly as their main form of communication to those around them. When comparing this to the 150 thousand people in the UK that know more than two words in BSL and are able to communicate in BSL (Ben Jaab, 2021; Central Digital & Data Office, 2017), a minority of about 0.2% of people in the UK are able to communicate with this population.

With the small percentage of only 0.2% of people able to speak in BSL effectively, it means that 87 thousand people are unable to speak to roughly 99.8% of the total population in the United Kingdom. With over thirty-three percent of people in 2019 making up essential service workers (Office For National Statistics, 2020), a large majority of people unable to use BSL likely make up the majority of the workforce in essential services, which affects those who require sign language to communicate in their ability to access such services. Such essential services include, but are not limited to, medical care (Emond et al., 2015; Kuenburg et al., 2015), public transport, and grocery shopping.

The most dire services for someone to access are medical services. In the UK, to access non-emergency medical care the first step is to contact a General Practitioner

(GP). A GP is typically contacted through a phone call, although at some GP clinics, it is possible to contact through an online form. Phone calls are completely inaccessible to those who are hard of hearing or Deaf, whilst online forms usually take time for responses and do not allow for the user to gain access to same-day appointments the same way a phone call could. This led to nearly half of Deaf people (44%) finding that getting into contact with their GP is a difficult process (Emond et al., 2015). Even after a Deaf person has made contact with their GP and an appointment has been made, it was found that they were less likely to be able to communicate to their physician in their most fluent language when compared to those who spoke English as a second language (McEwen and Anton-Colver, 1988).

An inability to communicate effectively with healthcare professionals is causing Deaf people to be generally denied proper access to these services, which is leading to poor health among the population (Signhealth, 2014; Rogers et al., 2018). In parallel, the communication barrier between a doctor and a Deaf patient could lead to the patient receiving inadequate care, such as an incorrect diagnosis and therefore treatment (Signhealth, 2014). Moreover, Deaf patients who are correctly diagnosed are still more likely to receive inadequate treatment, when compared to the general population.

The preponderance of issues that Deaf people experience today probably stems from the inability to communicate effectively with others. The low capacity to understand other people or be understood by others is the root of their ineffectiveness in communication with those around them. Enabling communication between the two parties, BSL users and non-BSL users, is critical in alleviating these issues. Currently, there are a lack of methods which are implemented in the medical 'arena' to help facilitate effective communication between the Deaf/hard of hearing community and others. Some recommendations include ensuring that the view of the speakers' mouth is not obscured, using a sign-language interpreter, and having each party write/read the conversation (Barnett, 2002).

The justification that the view of the speaker's mouth is not obscured, it to enable Deaf persons' to lip-read. Being able to lip-read is a common expectation of the

Deaf/hard-of-hearing community, however it is rarely useful in conversation (Kyle et al., 2005). Lip-reading, the act of understanding ones speech from observing lip movements (National Deaf Children's Society, n.d.), is only possible given the correct conditions, such as the lighting and the direction that the speaker is facing (Kyle et al., 2005). Better lighting and having the speaker face toward the lip-reader would allow for better comprehension through lip-reading (National Deaf Children's Society, n.d.). Relying on lip-reading to understand others has some drawbacks, one of which is that most of the sounds in the English language are not formed using the lips, but rather with the throat and mouth (Barnett, 2002), leading to only 30% of English speech being readable using exclusively the lips (National Deaf Children's Society, n.d.; Barnett, 2002), which excludes most of the words in the English Dictionary. In addition to this, a majority of Deaf people are actually unable to lip-read (Kyle et al., 2005), which means that the expectation from others is unfounded. In addition tho this, with the recent Covid-19 pandemic, government legislation required the use of face masks by the general public, which obscured the mouth and rendered the use of lip-reading impossible.

Secondly, there is the use of an interpreter, a person who knows both British Sign Language and English that provides a translation service for both parties. There are, however, a few issues with using an interpreter, the first of which is the shortage of people in the country with the ability to converse in BSL, leading to a lack of qualified interpreters in the country (Kyle et al., 2005). In addition to this, some people may require the use of an interpreter privately, incurring the cost to them personally, which can be costly and unfeasible due to the scarcity (Kuenburg et al., 2015). The scarcity of qualified interpreters means that it could be impossible for someone to be able to access such a service, which may lead to some BSL users using unqualified interpreters, which may lead to a myriad of issues. Regardless of the qualified status of an interpreter, there are concerns when using an interpreter. The main concern when using an interpreter is privacy or lack thereof (Kyle et al., 2005). Interpreters are used by the Deaf community to allow for access to essential services such as banking or medical services, this would mean that an interpreter would be

present during conversations relating to these services and would be able to pass this information to anyone.

Finally, a method used for facilitating communication is to write the conversation down. This would require both parties to write down their side of the conversation , creating a record of the conversation. This would be a slow back and forth between the two parties and would take a long amount of time when compared to other solutions. Another issue with this solution is that the average reading level of the Deaf population is lower than the average reading level of the general population (Rogers et al., 2018; Barnett, 2002). With a lower reading level it would incur longer reading times, and slow down the process. Finally, there is a privacy issue with writing down the conversation. Creating a record of the conversation can act as proof of what is said, and if there is any sensitive information within such records, it would be possible for others to gain access to the information.

2.2 Project Literature

This project was developed with the intention to help with some of these issues, allowing BSL users to use their natural language to converse. This project proposed the development of a machine learning/computer vision model to translate BSL fingerspelling into text.

Machine learning models have been implemented and tested before for classifying sign languages, such as American Sign Language (ASL) (Njazi and Ng, 2021) and British Sign Language (Liwicki and Everingham, 2009; Albanie et al., 2007). These examples proved that it is a possibility to create a machine learning model capable of classifying sign languages, although no model has been developed into a working system capable of creating a transcript of a conversation.

Previous machine learning efforts, namely TESSA, have been made in a reverse direction, enabling translation of text and speech into British Sign Language (Cox et al., 2002). This software would transcribe a conversation from speech into text, and then lookup the phrases in a video database, displaying the correlating videos to

the BSL user. If implemented alongside a system which translated BSL into speech, a full conversation could be translated in real-time enabling free-flowing conversation between the two parties. This system was implemented and tested in a real world application, the Post-Office. Feedback from the users were positive, saying that they would prefer to have the system (Cox et al., 2002). This can infer that those in the Deaf community actively welcome the use of electronic systems to enable conversations.

Some of the systems which have been developed show the current classification of the shown sign language live (Njazi and Ng, 2021), whereas others provide no live system at all (Liwicki and Everingham, 2009). Being able to provide these models in a complete system would make the usability of the system better, meaning that more people would be able to access the system. Furthermore, the use of a piece of software to act as an interpreter for the BSL community would increase the availability of the system, rather than needing dedicated hardware for the system to operate on. To support this, electronic systems are already being used by some BSL users/people in the Deaf community (Central Digital & Data Office, 2017; National Institute on Deafness and Other Communication Disorders (NIDCD), 2019).

One issue which is common in training machine learning algorithms is the quality of the data provided. One method which can be used to increase the quality of the data is to provide more context, known as a pretext. Providing pretext to an image or the data usually involves artificially modifying the data to remove unnecessary information, such as the background, or adding extra information to show the relationship between relevant objects in the image, or both. Such examples can be seen in Albanie et al. (2007), where BSL was successfully classified using finger positions, and used added context with mouthing cues. Mouthing cues will not be implemented in this project, as BSL fingerspelling does not involve the use of mouthing cues to provide information to the recipient.

Extra context does not always need to be adding information not directly relevant to the classification, sometimes overwriting the information in an image to show relationships such as directions can be useful. Liwicki and Everingham (2009) manage

to provide extra context to the image using hand shape descriptors and segmentation. These hand shape descriptors show the shape of the hand, along with the general orientation of the image, being represented through a Histogram of Orientation Gradients (Liwicki and Everingham, 2009). Similar efforts will be made in this project, using machine learning efforts to map a hand skeleton to each hand used in the signing, providing information such as the bones and joints in the signers' hand(s), with the hope of facilitating better accuracy.

In addition to this, it has been known for some implementations use image processing to alter the images before being used for training in the machine learning model. Such image processing methods include edge detection (Liwicki and Everingham, 2009) and histogram equalization (Kumar, 2022). The use of histogram equalization in Kumar (2022), was to provide a framework to make sure that all the images ended with similar lighting to reduce variability between images, one of the main issues when training and using a machine learning model.

Even though, Kumar (2022), Albanie et al. (2007) and Liwicki and Everingham (2009) perform pre-processing to allow for more information to be extracted from the originally captured data, Njazi and Ng (2021) shows that the use of pre-processing may not be necessary to still provide worthwhile results, with the model still able to correctly classify the signs.

When looking at the types of machine learning models used for this classification, the most common type of model used is a convolutional neural network (Albanie et al., 2007; Njazi and Ng, 2021; Liwicki and Everingham, 2009). Although this method is chosen commonly, the reasoning for this choice is not was not detailed. This project will compare differing machine learning models and their effectiveness on the classification problem, refining the most accurate model to allow for the highest accuracy without disregarding any models. Although not explicitly stated, the use of a convolutional neural network (CNN) allows for higher accuracy when compared to normal neural networks (NNs) (Elngar et al., 2014). Furthermore, the use of a CNN reduces the calculations when compared to a NN and allows for weight sharing, which reduces the computational complexity significantly, allowing for the

model to run faster and using less memory (Elngar et al., 2014). The use of other machine learning models may require the images to be processed prior to entering into the model. This could be through feature extraction or flattening the image into a one-dimensional array.

This project differs from the works of Kumar (2022), Albanie et al. (2007) and Liwicki and Everingham (2009), through its use of differing pretext systems, the use of a hand skeleton system is previously unseen in these works. In addition to this, the project is informed through these articles, where the project uses pretext to better inform the neural network. Also, this project was further influenced by Kumar (2022), Albanie et al. (2007) and Liwicki and Everingham (2009) by ensuring that a CNN is used, rather than manual feature extraction, to allow for higher accuracies in image-based machine learning applications.

This project was developed to translate the BSL subset, fingerspelling. Fingerspelling involves the use of spelling out each individual word rather than having separate signs for each word. This method of communication would take longer for the user to perform but would allow for a larger lexicon of words to be conveyed. Furthermore, this subset of BSL reduces the amount of data and also the number of classes to be classified. There are 26 letters in the English alphabet, each with its own sign in BSL fingerspelling, in comparison to the 1,800 words in the full BSL lexicon, fingerspelling would reduce the class problem by 98.5%. As shown in (Liwicki and Everingham, 2009), increasing the classes to be recognized in a model, reduces the accuracy the more classes that are introduced.

To conclude, BSL users including Deaf people and those hard of hearing can struggle with communicating with others, including those who work in essential services, such as healthcare. Being restricted in such communication can lead to being unable to access these services, and in the case of healthcare, can lead to a decline in their overall welfare. There have been attempts in the computer vision area to solve this issue, but never has a fully working system been implemented. This project is being developed to help with this, developing a Computer Vision / Machine Learning based

system to classify and ‘translate’ BSL fingerspelling into text, using an artificially generated pretext to help improve accuracy.

Chapter 3

Requirements Analysis

3.1 Toolsets and Machine Environments

This section will detail the specific environments and toolsets used during the design, development, and evaluation of the project.

3.1.1 Python

For the programming language used to develop the application, Python was selected. When researching different programming languages for machine learning applications, Python was a common selection by multiple sources (Mahendra, 2023; S. Gupta, 2021; Voskoglou, 2017). The reasons for this are below.

Firstly, with Python being a popular programming language, many different libraries and packages have been developed. This means that development can be completed faster, with a time restriction on this project, it was critical that development could be quickened by any means necessary. (S. Gupta, 2021)

Secondly, Python has a simple syntax that enables a high level of readability in its programs. This means that as a programming language, it can be easy to learn and allows the programmer to focus on the code rather than the syntax of the language (S. Gupta, 2021). In addition to this, if the project is to be reviewed or further developed by others, a simple syntax means that other developers would be able to understand the code easier.

Python being the best choice is further solidified by the fact that over 50% of data

scientists and machine, learning developers used Python for such applications in 2017 (Mahendra, 2023; Voskoglou, 2017). In addition to this, IEEE Spectrum ranked Python first among other programming languages (Mahendra, 2023).

Other programming languages which could have been used for this project include:

1. R
2. Java
3. LISP

Finally, Python was chosen over these programming languages due to its ease of use which is not present in LISP (S. Gupta, 2021), along with the availability of libraries and packages. In addition, the fast simple syntax enables quick development of models and machine learning models, which is crucial in a project where there is a short deadline and only one developer.

3.1.2 Visual Studio Code

Visual Studio Code was chosen for the development environment for this project.

Visual Studio Code (VSCode) is a development environment for a multitude of programming languages, which supports the use of Python. Within the VSCode environment, it is possible to run Python programs through the built-in terminal interface. This further enables the use of such a terminal to install separate packages such as TensorFlow without leaving the environment. Furthermore, VSCode is able to provide suggestions whilst programming to improve flow throughput without the need to look through documentation to find functionality. Lastly, the user is able to load an entire project folder, with each file stored in tabs to allow for the user to quickly switch between the active Python file, without leaving the environment.

Another development environment that could have been used would have been Python Integrated Development and Learning Environment (IDLE). This environment is also able to run Python code, however, does not provide the functionality of a built-in terminal or the ability to make suggestions whilst coding. IDLE also does

not support opening a directory to access the Python files but rather requires the user to open each file individually, with separate windows for each file.

The final development environment which could have been used is Visual Studio. Visual Studio is an Integrated Development Environment, which has the ability to run and use multiple programming languages. Although very similar in its functionality to VSCode, it lacks in the number of plugins available for development, which might allow for faster development if used.

Table 3.1: Comparison Matrix for Development Environments

	Development Environment		
Feature	VSCode	IDLE	Visual Studio
Python Support	Y	Y	Y
Directory Loading	Y	N	Y
Programming Suggestions	Y	N	Y
Built In Terminal	Y	N	Y
High availability of Plugins	Y	N	N
Git Integration	Y	N	Y

3.1.3 Google Colab

Google Colaboratory, shortened to Colab, is an online service which enables the use of cloud computing to perform Python programming. Although any and all python programming can be performed in a Colab environment, it is typically used for machine learning applications.

Google Colab can be directly compared to Jupyter Notebook, another service which provides similar functionality, only through the use of local machines rather than cloud computing. Google Colab was chosen instead of Jupyter due to its ability to directly link to a Google Drive, where data was stored to be processed by the project. In addition to this, using cloud computing rather than a local machine, enables the program to run on the web whilst other processes happen on the local machine.

Google Colab and Jupyter Notebook alike, process Python code in blocks, storing

the variables from previously run blocks, being able to run such blocks in any given order. For this reason, Google Colab was used as a 'playground' for testing code without the need to re-run the whole training process if any changes were made to code after the model had been trained. This enabled a faster testing methodology, such that if there were any bugs in the development of the graphs and/or figures that they could be rectified without waiting for the model to be re-trained. In addition to the above, the programs developed would be automatically saved to the cloud, enabling access on any device and ensuring that the work was saved.

3.1.4 GitHub

GitHub, based on the Git service, is a tool used by developers that implement version control. GitHub specifically enables the user to store their repositories online to be accessed anywhere. In this project, GitHub was used to implement version control, backup storage along with storage to allow access to the code base on differing devices.

GitHub can be accessed mainly in two methods, through a command line-based application such as Git-bash or through software such as GitHub Desktop. For this project, GitHub desktop was used, as it provides an easier method to the same results, whilst being able to show graphical changes in each commit to the repository. This enables visibility when changes are made, allowing for such changes to be reverted if needed.

There are many other options to store the code, such as using an online or cloud-based storage solution such as Microsoft OneDrive or Google Drive. Such services would enable the user to access and modify the code base from anywhere with an internet connection but do not support version control. Without version control, any changes made would have to be manually reverted, requiring the user to remember the changes being made. Furthermore, with these online solutions, changes made will need to be manually uploaded each time, whereas GitHub through the desktop application makes this easier as you only need to press a singular button to make the

changes. These reasons are why GitHub was chosen for the storage of the application, and further details can be seen in the comparison matrix below.

Table 3.2: Comparison Matrix for Storage Methods (Gallard-Bridger, 2022)

	Storage Methods			
Features	Google Drive	OneDrive	GitHub	Local Storage
Cloud Storage	Y	Y	Y	N
Version Control	N	N	Y	N
Free Max Size	15GB	5GB	100GB	N/a
Paid Maximum Size	100GB	1TB	100GB	N/a

3.1.5 Python Packages/Libraries

This section will detail the packages used in the development of the application, comparing them to other packages which perform similar tasks (if available).

TensorFlow

TensorFlow is a library used for machine learning applications, provided by Google. Although available in other programming languages, this package is being used as a Python library. It also offers the possibility to create and use models on many platforms. With a variety of model types and additional functionality through the built-in Keras module, it allows for flexibility unmatched by other packages.(Costa, 2020)

Other libraries which could have been used are the PyTorch library and sci-kit learn library. The reason that PyTorch was not chosen for this project is that PyTorch is not optimized for speed and therefore, with the intentions of a live application, it would provide a slower model. The main reason that sci-kit learn is not used in the project is its lack of functionality, and unable to create a convolutional neural network. (Costa, 2020)

OpenCV

OpenCV is a package that provides computer vision based functionality in Python. For this project, OpenCV was included to provide a method in which the camera functionality can be used. This project required the use of a camera, to capture the dataset for training the models and for the live classification application.

OpenCV was chosen specifically as when it stores images it does so in NumPy arrays, which are compatible with a variety of other packages such as MediaPipe 3.1.5. In addition to this, it is the most recommended method for accessing the webcam, shown by when searching "loading the webcam in python" on Google, in the first two pages under 10% of the results involved non-OpenCV based solutions.

MediaPipe

The MediaPipe package was used to provide the pretext functionality for the program, with its ability to perform hand landmark detection. MediaPipe is a package provided by Google to allow for the development of applications with their pre-built Machine Learning (ML) functionality. The MediaPipe library works seamlessly with the OpenCV library allowing for simple and quick development using ML tasks without the need to train the model.

Without the use of MediaPipe, there are few pre-built options for hand skeleton detection. Although this meant that MediaPipe must be used, the ease of use in the package, and where hand skeletons could be detected in a few lines, and examples on the internet from Google allowed for ease of development in the pretext.

Another approach could have been to program the hand skeletons by hand, involving hours of development. This approach would not have been wise, due to the time constraints in the project and the amount of work required, when such a package has been previously created.

MatplotLib & Seaborn

MatplotLib and Seaborn were used to provide the graphs for the analysis of the model. These two packages work hand in hand to allow for the graphs and plots to be displayed and saved.

MatplotLib is a base package used by many other packages, such as Seaborn, to provide graphing functionality to programs. These allow for the input of NumPy arrays along with other arrays to provide data, allowing for the previous methods to be used directly without the translation of the array types.

Although MatplotLib typically cannot be avoided for the plotting of graphs, there are many alternatives to Seaborn. Such alternatives include the use of MatplotLib natively, using Bokeh or pygal. Seaborn was chosen over these as it acts as a wrapper for the MatplotLib library, enabling graphs to be created with less programming, speeding up the development process. The other alternatives can be embedded into web applications and provide extra functionality, but are rivaled by the simplistic nature of Seaborn.(Bierly, 2022)

3.2 Risk Analysis

This section of the report details the risk assessment taken. Detailing the risks themselves, their likelihood, impact, and management. This enabled precautions to be made to ensure that the project was able to be properly conducted.

Table 3.3: Risk Assessment Table

Risk	Likelihood	Impact	Management
The dataset collected is not of the required quality.	Low	High – Would not allow for the machine learning model to be trained.	Supplement the data with self-made videos and images.
The dataset collected does not contain enough videos.	Medium	High – Would not allow for the machine learning model to be trained.	Supplement the data with self-made videos and images.
The dataset collected is not stored securely.	-	High – Would cause ethical issues for the accidental release of personal data.	Store the video files only on the cloud, storing videos of only hands to reduce the ability to confirm who the person is.

3.3 Testing and Evaluation

To ensure that the application conforms to the goals set out in the aims and objectives, it is integral to ensure that the testing of the application is completed. The testing of the application will be performed at the end of every iteration using the following metrics and methods:

Accuracy - The accuracy of the model will be tracked during training using the following equation *(True Positives / Total) * 100 = Accuracy* (Where True Positives is the number of correctly classified samples and Total is the number of samples in the set). This accuracy measurement will be used for the training dataset and validation dataset of the models.

Confusion Matrix - A confusion matrix will be produced at the end of training, using the testing dataset. This will show which letters are being classified correctly by the model, giving insight into which classes may be more difficult to classify.

Live Testing - The models will be tested live, where each model is tested in a development environment. This will be a non-objective view of the model, seeing if it 'feels' as if the correct letters are being classified. This would show if the model is being over-trained.

In the end, the live testing is the most important test of the model, allowing us to check if the model is capable of running live, without performance issues. In addition to this, we can check if the model is able to classify the letters in the image properly, by having a person sign multiple letters during the test will check the difference between letters and the latency between being able to check

Although the above is true, to save time, the differences between the models will be checked first by the accuracy and confusion matrix metrics. To ensure that the model is capable of classification first. Performing this in this method will enable preliminary checks on the model before checking it live, saving time on checking the models on a live framework if they couldn't classify the testing images properly.

Chapter 4

Design and Methodology

4.1 Project Management/Software Development

This project was managed through the use of a Kanban/Waterfall hybrid methodology.

The waterfall methodology enabled the use of a Gantt chart to plan the overarching tasks of the project, planning the ideal amount of time for each task. This rigidity allowed for the project to be completed, by the deadline of the project, the 11th of May 2023. The Gantt Chart can be seen below in figure 4.2. Furthermore, if a task was completed earlier, than planned, the next task in the order was able to be started earlier, enabling a truly flexible method of working, ensuring that time was most efficient.

The Kanban portion of the methodology was the use of a Kanban board (Figure 4.1)to manage the smaller tasks which combine to make the overarching tasks of the project, as part of the software development. This managed to allow for differing steps to be performed at the same time, whilst ensuring that all the tasks together were performed in a timely manner.

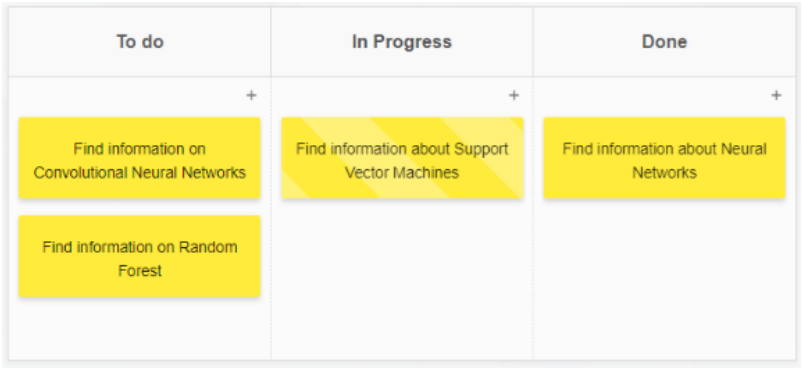

Figure 4.1: Kanban Board Example (Research Machine Learning Models)

In the figure below, the Gantt chart used to plan this project is shown. This Gantt chart enabled planning of the overall project, ensuring that tasks that needed to be completed were completed in a timely manner. Accompanying the Gantt chart is a table which gives details on the names of each of the tasks to be performed for this project, where each row in the table is a different task. Having a table to accompany the Gantt chart allowed for the possibility to view the tasks left to be completed, without cross-referencing them with the Gantt chart.

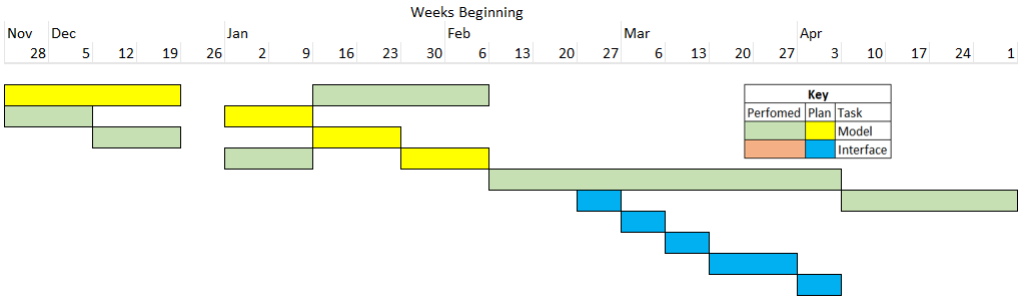

Figure 4.2: Gantt Chart for the Project Plan

Table 4.1: Table of Tasks to Accompany Gantt Chart

Task	Duration	Start Date	End Date
Classification Model Development			
Find/Create Dataset	4 Weeks	28/11/2022	25/12/2022
Research Pretext Tasks	2 Weeks	02/01/2023	15/01/2023
Implement Pretext Tasks	2 Weeks	16/01/2023	29/01/2023
Research Machine Learning Models	2 Weeks	30/01/2023	12/02/2023
Implementation/Training of Models	2 Weeks	13/02/2023	26/02/2023
Refinement of Model	1 Week	27/02/2023	05/03/2023
Interface Development			
Requirement Analysis	1 Week	06/03/2023	12/03/2023
Design the User Interface	1 Week	13/03/2023	19/03/2023
Implement the Interface	2 Weeks	20/03/2023	02/04/2023
Test the interface	1 Week	03/04/2023	09/04/2023

As shown by the Gantt Chart above, the tasks in the project were performed in a different order than planned. This was due to time constraints and the ability to collect the data. Initially, there was a need for additional ethical applications, which would have inhibited the ability to collect such data, however, the data collection method was changed to use only one person (the researcher) instead of outsourcing the data collection to third parties. This decision caused the dataset to be collected at a later time, but due to project management methodology, the development was able to be commenced whilst decisions were made and ethical applications were sorted.

In addition to this, the project length was increased compared to the original plan, this was due to some issues with the development of the model, where the data collected was not of a high enough quality and led to issues with the classification quality. This required the dataset to be recollected and delayed the overall project (detailed in section 6.1.3). Finally, the interface tasks for the project were undeveloped as planned, although this was a secondary aim, and was only to be implemented if time

permitted. This is due to the issues presented earlier. A rudimentary interface was developed, although working, it was purely for developmental purposes.

4.2 Model Choice

The model chosen for this project was a Convolutional Neural Network(CNN). Within recent research, CNNs were found to have greater accuracy than feature extraction and neural networks, due to their convolutional capabilities. In A CNN, the convolutional layers allow for feature extraction, rather than having to perform it separately. The CNN needs to be able to extract the correct features for classification, a common technique (adding pretext) is used. Adding pretext to the image where information is added to the image to enhance the intended features in the image.

There was also the opportunity to compare this CNN with other types of models, such as Decision trees, random forests, neural networks, and support vector machines. These approaches were chosen at the beginning to compare and ensure that the best approach was taken. Due to time constraints, these model types were not compared with a CNN, although research would dictate that it would be unlikely that these would be able to improve on the accuracy supplied by the CNN.

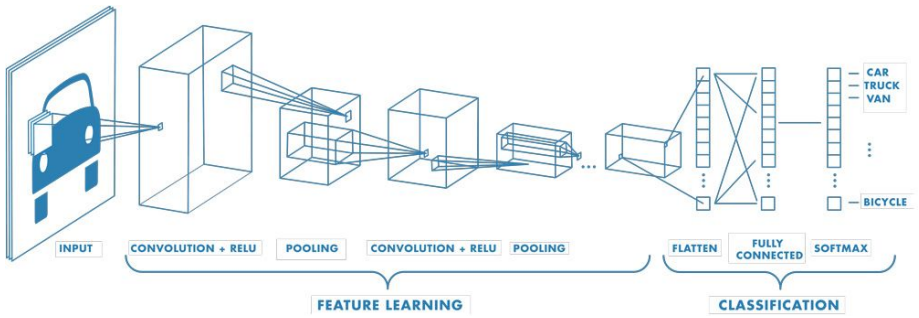

Figure 4.3: An image showing the CNN process (MathWorks, n.d.)

4.3 Design

As the secondary aim of this project was not reached, a design for the user interface in this project was not developed, however, there were some design choices relating

to the overall structure of the programs and the pretext for the model. In addition to this, there were design elements to the structure of the model itself.

4.3.1 Overall Project Structure

As shown in the development section of this project, there were two elements to the project, the developmental aspect and the release aspect. The developmental aspect of the project involved the use of programming to enable the developer to create and train the models. Whilst the release aspect involved the use of programming to enable the final model to be used live and to show the user the current letter being classified.

To allow for both of these sections to be developed in the fastest time, without restricting the functionality of the project, the re-usability of the code was crucial. This meant developing code which could be modular, working with classes and packages to ensure that each item of the program that was required in both poritons of the project could be used without any issues.

One of the main positives of this approach is that it ensures that the same results are provided, whether the code is being run from the developmental main program, or whether the code is being run from the release program. This ensures that when the model is being trained on an image, the same process will be used for all the other images used in the states of development and during the release stage, ensuring that the data is not being differently manipulated in any way.

The second positive of this approach is that it reduces the time taken to develop multiple pieces of code, as the release aspect did not require re-developing the ability. This reduction in time was crucial to the project, as the project required extra time at the end of the development phase which would not have been possible otherwise.

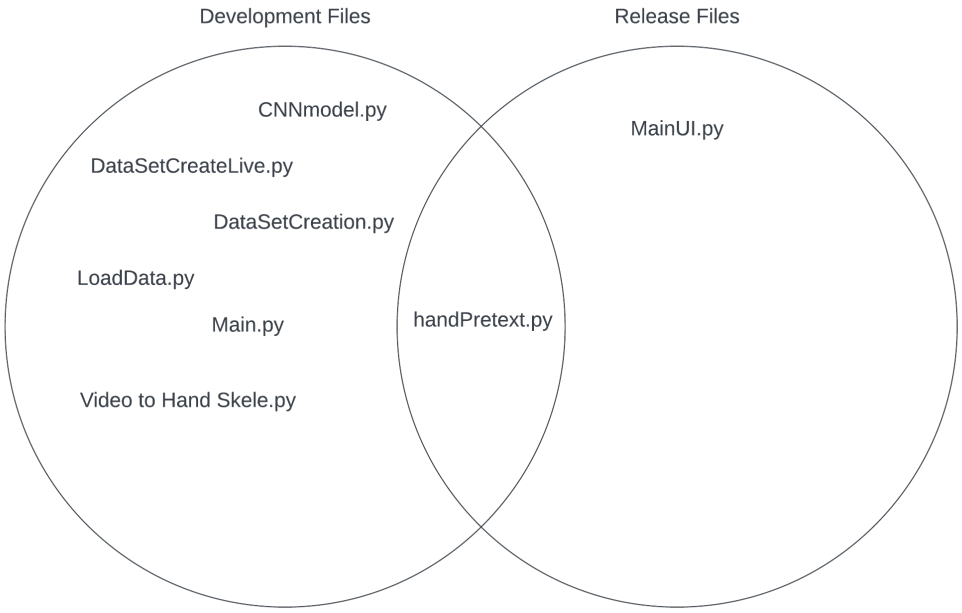

Figure 4.4: Venn Diagram Showing the Programs Made Modular

Although only one piece of code is being used by both phases in the project, as shown by figure 4.4, this sets up the project and allows for the processes in 'handPretext.py' (detailed in section 5.1.1) to be easily used by other projects or items throughout the development process.

4.3.2 Model Pretext

For the design of the model pretext, there were many possible options to feed into the model, all based on the hand skeletons.

Within the hand skeletons, there were many customization options to develop the pretext. The first and most obvious of which is which colour should the hands be. With each hand having twenty-one bones and joints, there could be the possibility to have 41 different colours for each hand. In addition to this, there was the outline of each of the joints, and potentially even the shape of the joints.

Secondly, there are different backgrounds that the hand skeleton image could have. This could be the image supplied to the hand skeleton method, or any block colour background for the image. If the background colour is too similar to the colour of

the hand skeleton, it could reduce the accuracy of the model, as it would be unable to discern between the hand skeleton and the background.

The final customization option would be to zoom the image to only fit the hand skeletons in, removing any and all unnecessary space from the image. For machine learning models, the input shape needs to be the same, whether it is training or testing or in use. This means that the image would have to be the same size as the training data. For this project, the images were given a predetermined size, 480x480. This was due to the technical capabilities of the camera on the testing device. Although it would have been possible to downscale the image to smaller dimensions, it was deemed unnecessary for this project.

Each of these could have an effect on the accuracy and effectiveness of the model, and some could even affect the others, such as the colour of the hand skeleton could affect what the most effective background colour would be and so forth. For this project, the use of a red hand skeleton and a black background was deemed the best option (see section 6.1.1 for more information).

4.3.3 Model

For the design of the model, there are many key features, such as the number and type of layers in the model, the number of epochs, and many more.

The Layers

To start, neural networks are able to perform better when the values inside the array passed into the model are smaller, typically between zero and one. For this reason, the model starts with a rescaling layer, in which all the elements are scaled down to fit between zero and one.

The next stage is to add the convolutional layers. For a convolutional network to work, there must be one or more convolutional layers, and in between such layers, there must be a pooling layer. As we are dealing with images, we need to use the two-dimensional versions of these layers. This model has been designed to have three convolutional layers and therefore requires two pooling layers.

Once the convolutions have been completed, the model needs to translate the two-dimensional values into a one-dimensional array for classification, this is completed through a flatten layer.

Finally, the model requires layers for the classification, in TensorFlow it is standard to perform this through the dense layers. This model uses two dense layers, with the final layer having an output shape of the number of classes for the model to be able to classify in this case five.

Table 4.2: Table Showing the Structure of the Model

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 480, 480, 3)	0
conv2d (Conv2D)	(None, 478, 478, 32)	896
max_pooling2d (MaxPooling2D)	(None, 239, 239, 32)	0
conv2d_1 (Conv2D)	(None, 237, 237, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 118, 118, 64)	0
conv2d_2 (Conv2D)	(None, 116, 116, 64)	36928
flatten (Flatten)	(None, 861184)	0
dense (Dense)	(None, 64)	55115840
dense_1 (Dense)	(None, 5)	325

Total params: 55,172,485

Trainable params: 55,172,485

Non-trainable params: 0

Epochs and Other Settings

The number of epochs a model takes to train can drastically change the outcome of the model. More epochs generally increase the accuracy of the model. This however usually leads to the model being overtrained, where the model expects the same data, and when new data is given to it, it performs considerably worse. To ensure that this does not happen, early stopping is usually implemented. This would mean that the number of defined epochs would not affect the training of the model, but

rather the model would stop training at its peak epoch. For this program, however, the maximum epochs for the model to train over is set at ten.

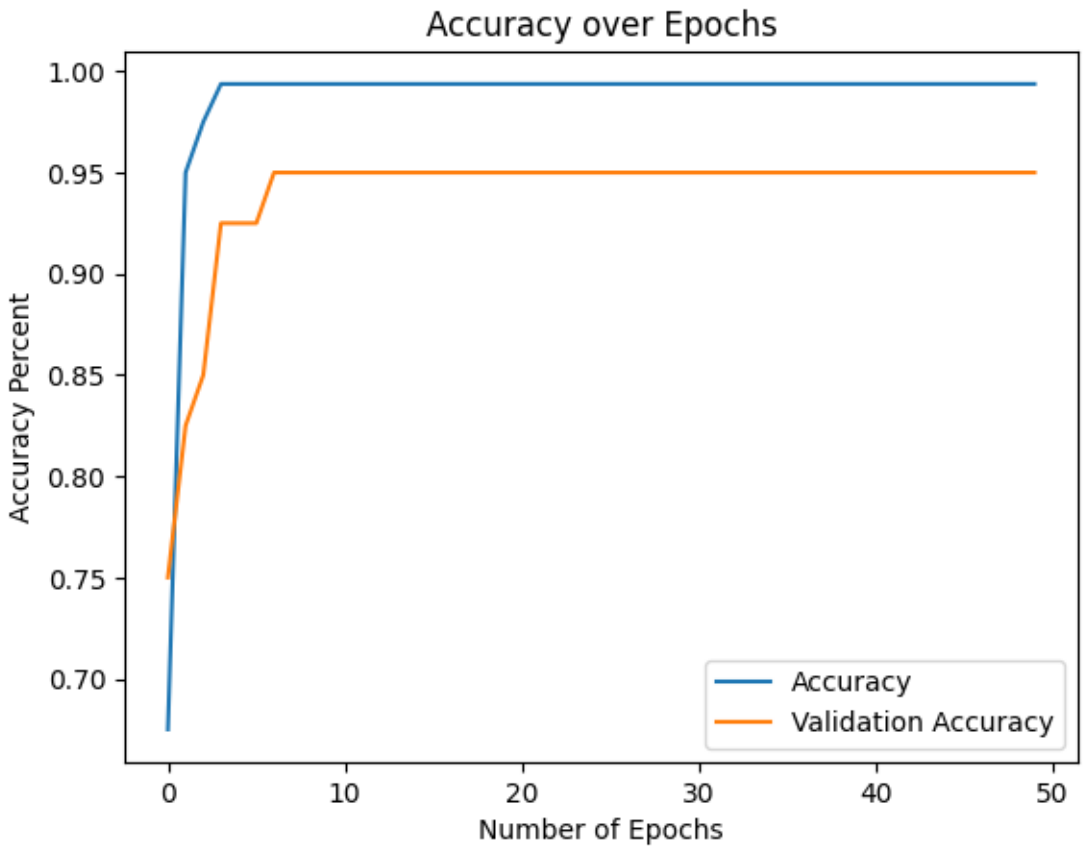

Figure 4.5: Graph Showing how the Accuracy Increased With Epochs

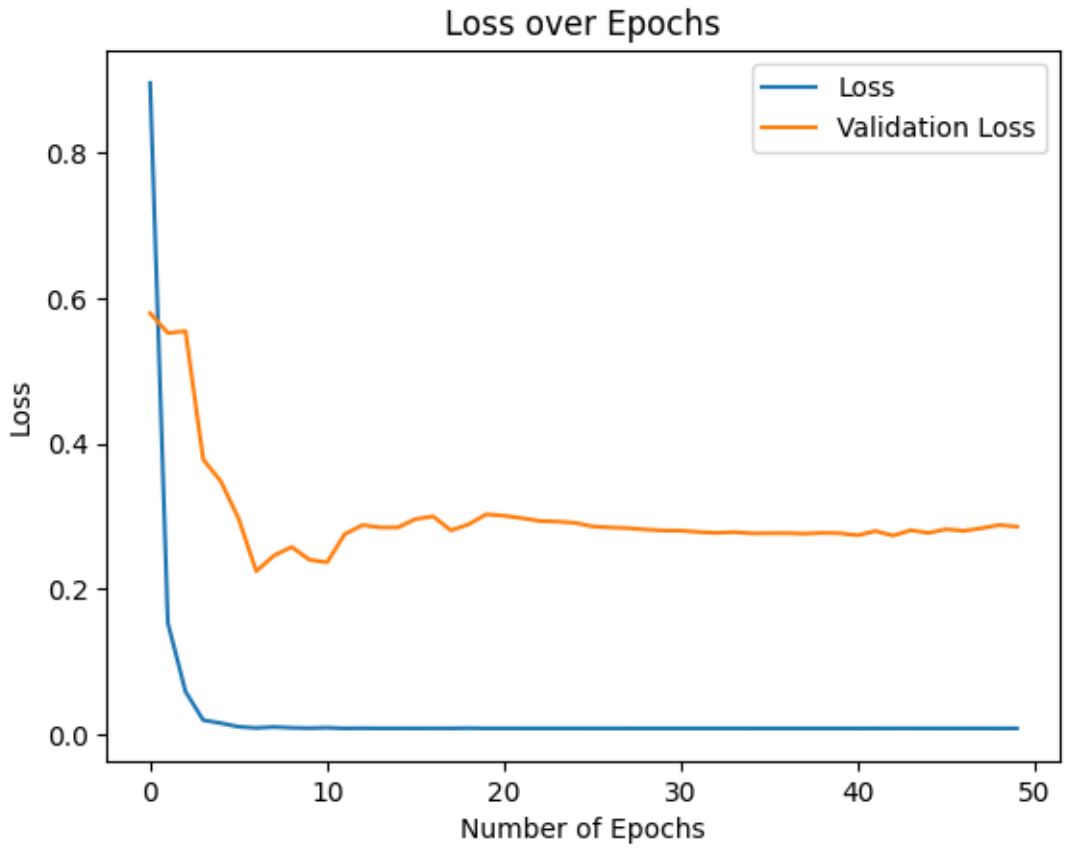

Figure 4.6: Graph Showing how the Loss Decreased With Epochs

As shown by figure 4.5, the accuracy for both the validation and training accuracy increases as the model goes through more epochs in training, and also stagnates between five and ten epochs. This would insinuate that the best number of epochs for training would be between these values. This is further consolidated by figure 4.6, where the loss for both training and validation data experiences its lowest between five and ten epochs. Through further inspection, six epochs would be the best for this model. Although training statistics can change depending on how the data is fed to the model, to ensure this does not change, the random generators can be supplied with the same seed.

One commonly used method in machine learning is to use early stopping for the training of models. This is the use of checking values over a pre-set number of epochs (amount of times the model looks at the full dataset) and seeing if they are stagnating or declining. This can ensure that the model is not overtrained. From

the above two graphs, it is clear that the early stopping would suggest that it should stop between five and ten epochs, to ensure the best algorithm.

Another setting which can be changed and may alter the accuracy of the model is the optimizer function. For the model developed, the optimizer function chosen was the 'adam' optimizer. The 'adam' optimizer was chosen as it is typically faster and usually provides better results than other optimization methods (A. Gupta, 2021).

The final setting for a neural network model is the loss function. The loss function produces a value in every epoch detailing how badly it was able to classify a single sample (Google, 2022). This loss function can provide useful insight into the usefulness of a model, typically used as the determinant for early stopping. For this project, the loss function sparse categorical cross entropy was used. This loss function is best used when there are more than two classes to be classified and the class names have not been one-hot encoded (TensorFlow, 2023). In this project, one-hot representation was not used and therefore sparse categorical cross entropy was the best option.

Chapter 5

Implementation

This section of the report will discuss the implementation of the solution, describing how the project has reached the current stage that it is at.

5.1 Pretext

For this project, the use of pretext was integral in allowing the machine learning model to extract the important information from each of the images. This was completed through the use of hand-tracking, which was implemented through the python library 'mediapipe' and imported into this project, as stated in the requirements analysis (section 3).

5.1.1 handPretext.py

The first file developed for the use of this project was the handPretext.py file. This file contains a class, 'hands', which can be initialized in other files, this was completed this way to allow for the re-usability of code, without the need to repeat functions, wasting time and space on the machine. There are four subroutines inside of this class to enable it to perform differing tasks on images, whether it is for training or testing purposes. Such subroutines include ___init___(), retrieveHandsOverlay(), handsBackground() and handstoCSV(). The full program for this can be seen in Apeendix A.

___init___()

This subroutine is used to initialize the class with the required attributes for the processing of the images. There are three attributes in the 'hands' class:

mpHands Used to access a predetermined constant relating to how joints in the hand are joined together, ensuring that the hand skeletons are represented correctly.

hands Used to access the processing ability of the library.

mpDraw Used to access the ability to draw the landmarks and skeleton frame onto the image.

```
    def __init__(self):
        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands()
        self.mpDraw = mp.solutions.drawing_utils
```

Listing 1: Code to Initialize Hand Pretext Class

retrieveHandsOverlay()

This subroutine receives an image, and overlays the hand skeleton of up to two hands onto the image or any other background if specified. The subroutine processes this in the following manner:

1. Check if there is a specified background, if not, use the supplied image as the background.
2. Convert the image to RGB (Red, Green, Blue) from BGR (Blue, Green, Red).
3. Process the image, retrieving the results from the mediapipe function.
4. If one or more hands have been found in the image, loop through each hand
5. Place a large circle on the background at the base of the hand.
6. Draw the rest of the landmarks on the background along with the skeleton of the hand.
7. Return the background and handskeleton results (if asked for).

```
    def retrieveHandsOverlay(self, image, background=None,
    ↪  receiveResults=False):
        if background is None: # set background to the image if no
        ↪  background has been specified
            background = image
        imageRGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # turn the
        ↪  image to RGB
        results = self.hands.process(imageRGB) # process the image
        ↪  to receive hand information
        if results.multi_hand_landmarks:
            for handLms in results.multi_hand_landmarks: # working
            ↪  with each hand
                for id, lm in enumerate(handLms.landmark):
                    h, w, _ = image.shape
                    cx, cy = int(lm.x * w), int(lm.y * h)
                    if id == 0 :
                        cv2.circle(background, (cx, cy), 25, (255,
                        ↪  0, 255), cv2.FILLED)
                self.mpDraw.draw_landmarks(background, handLms,
                ↪  self.mpHands.HAND_CONNECTIONS)
        if (receiveResults):
            return (background, results)
        return background
```

Listing 2: Code to Retrieve the Hand Skeletons From an Image

handsBackground()

This subroutine is being used as a wrapper for the *retrieveHandsOverlay()* subroutine, described in section 5.1.1. A wrapper function can be defined as "a function (another word for a subroutine) in a software library or a computer program whose main purpose is to call a second subroutine or a system call with little or no additional computation" (Wikipedia, 2023). This subroutine's supplemental function is to provide the *retrieveHandsOverlay()* subroutine with a black background rather than the user needing to define it themselves. With this functionality being used in both the creation of the dataset and the live interaction with the model, the decision was made to implement a dedicated function for this, rather than having to perform this manually over multiple lines each time.

This subroutine provides this functionality using a 3D NumPy array filled with zeroes, the third dimension being used for the colour layers. Using a NumPy array

was crucial as that is the method in which the images are stored locally on the system and would allow for quick conversions without the need of extra code, furthermore, providing zeroes to the system would ensure that the background was black.

```
    def handsBackground(self, image):
        background = np.array(np.zeros(image.shape)) # create an
        ↪  empty background for the locations to be placed on
        img = self.retrieveHandsOverlay(image, background)
        return img
```

Listing 3: Code to Retrieve Hand Skeletons on a Black Background

handstoCSV()

This subroutine is used as a method to convert the hand skeletons into a CSV format, where each point is represented by two columns, of their 'X' and 'Y' values in relation to the base of the right hand. This was performed to allow one dimensional methods to attempt to classify a hand skeleton. Although implemented, no one dimensional models were implemented for reasons explained in section 4.

This method would use the calculation of $(x - RH0_x, y - RH0_y) = (new_x, new_y)$, where x and y are the co-ordinates of the current joint being investigated and $RH0$ is the point relating to the base of the right hand. Similar to the *retrieveHandsOverlay()* subroutine, this loops through each joint throughout each hand, but instead of plotting this upon an image, it adds the new co-ordinates to a CSV writer which will commit the change to a comma separated value ('.csv') file.

5.1.2 LoadData.py

This python program was developed to act as an intermediary between the *handPretext.py* module (5.1.1), to create the dataset. For this project the dataset was created in multiple stages. The program deals with everything after the data collection, namely the conversion to separate images from videos, converting these images into hand skeleton images and converting the hand skeletons into a CSV. One final method was implemented for the loading of the CSV file, but due to time constraints was not used.

loadTrainTest()

This subroutine was created to load the CSV file containing the hand skeleton information, as three separate dataframes, one for each of the training, testing and validation datasets.

Using pandas the CSV file was read, and the classification column was converted into coded categorical data. This was then converted into a tensorflow dataset, as it was said to be easier for training the tensorflow models, where the data was shuffled and split into a 70:20:10 split for testing, training and validation respectfully.

```
    def loadTrainTest(csv):
        dataframe = pd.read_csv(csv)

        dataframe['classification'] =
        ↪  pd.Categorical(dataframe['classification'])
        dataframe['classification'] =
        ↪  dataframe.classification.cat.codes

        targetVars = dataframe.pop('classification')
        tf_dataset =
        ↪  tf.data.Dataset.from_tensor_slices((dataframe.values,
        ↪  targetVars.values))

        trainSize = int(0.7*len(dataframe))
        testSize = int(0.2*len(dataframe))

        tf_dataset.shuffle()

        train = tf_dataset.take(trainSize)
        test = tf_dataset.skip(trainSize).take(testSize)
        val = tf_dataset.skip(trainSize+testSize)

        return train, test, val
```

Listing 4: Code to Retrieve Training/Testing datasets From a CSV file

This subroutine was not used by the end of the development phase of this project, but given more time would have been. This was due to the time constraints leading to there being no time for the development of other models than the one which was able to accept the images. This data could have been used to train and test these other models.

loadFromFolders()

This subroutine was developed to read the original images, process the hand skeletons, adding the information to the CSV file. This was developed as its own subroutine to enable the system to loop though each of the images through the folders in a single subroutine.

```
    def loadFromFolders(parentFolderLocation):
        handProcessor = hands.hands()
        for imageFolder in os.scandir(parentFolderLocation):
            for imagePath in os.scandir(imageFolder.path):
                image = cv2.imread(imagePath.path)
                _ = handProcessor.handstoCSV(image,
                ↪  imageFolder.path[-1])
```

Listing 5: Code to Save Hand Skeleton Data to CSV

createHandSkeletons()

This method was developed to allow the user to enter only two paths, which correlate to the location of the original pictures and the hand skeletons, where it would loop through this path and generate the hand skeletons for each image in the original picture dataset, saving it in a separate path to preserve the original data.

```
    def createHandSkeletons(parentFolderLocation, handImgLocation):
        handProcessor = hands.hands()
        for imageFolder in os.scandir(parentFolderLocation):
            currentLetter = os.path.split(imageFolder)[1]
            currentPath = handImgLocation + "\\" + currentLetter
            os.chdir(currentPath)
            for imagePath in os.scandir(imageFolder.path):
                image = cv2.imread(imagePath.path)
                newImg = handProcessor.handsBackground(image)
                imagename = os.path.split(imagePath)[1]
                cv2.imwrite(imagename,newImg)
```

Listing 6: Code to Process Images to Hand Skeletons

videoToImages()

For the initial collection of the data, videos of a single signer were taken and cropped to only contain the signs. This meant that they needed to be separated into indi-

vidual images as to be processed by the neural network. This is what this subroutine was created to perform. It would allow for two paths to be entered, one containing the path of the video and one where the images should belong.

```
    def videoToImages(parentFolderLocation, imagePath):
        for videoPath in os.scandir(parentFolderLocation):
            currentLetter = os.path.split(videoPath)[1][0]
            imagePathTemp = imagePath + "\\" + currentLetter
            os.chdir(imagePathTemp)
            video = cv2.VideoCapture(videoPath.path)
            success,image = video.read()
            count = 0
            while success:
                cv2.imwrite(currentLetter + "%d.jpg" % count, image)
                success,image = video.read()
                count += 1
```

Listing 7: Code to Process Videos to Separate Images

5.2 Dataset Creation

For this section the programs developed to create the two datasets will be discussed. These programs use the functions in the pretext section to develop a full dataset with information extracted to help the model classify BSL fingerspelling.

5.2.1 DataSetCreation.py

This piece of code purely interacts with the *LoadData.py* (section 5.1.2) file and was used to create both datasets, not needing to use the videoToImages for one of the datasets. This program simply imports the LoadData module and uses saved locations to call the LoadData functions to create the images from the videos, the hand skeletons from the images or the CSV from the handskeletons. This was completed through manually commenting and un-commenting the program. The full code for this can be seen in Appendix C.

5.2.2 DataSetCreateLive.py

This program was developed to create the second dataset used for the training of the Convolutional Neural Network model (CNN). This program accesses the user's webcam and asks for the letter that is being signed. Once confirmed, the system takes a picture every four seconds, and saves these images in a predetermined folder specified in the code. Figure 5.1 shows an example of what the user sees. You can see two numbers clearly, the top being the number of seconds until a picture is taken whilst the lower is the number of pictures already taken for this class, this was included so that when the dataset becomes larger, it doesn't become an issue of remembering the number of images taken for each class. The full code for this module can be seen in Appendix D.

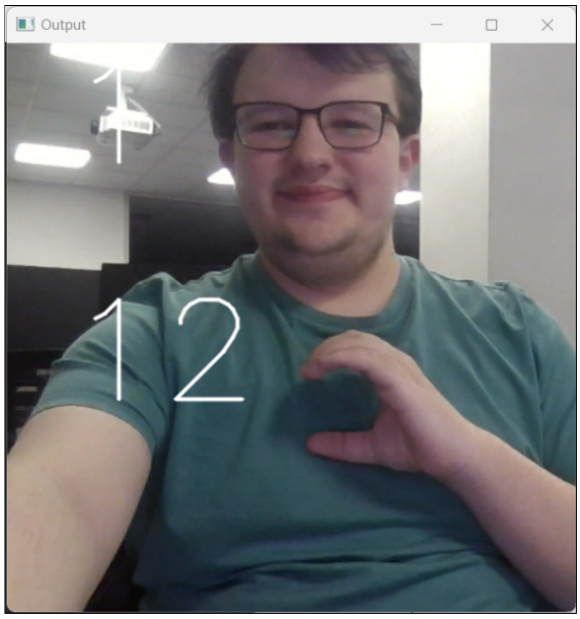

Figure 5.1: Example of the User's View

5.3 Models

This section details the programs used to train/create the machine-learning model for the system.

5.3.1 Convolutional Neural Network (CNN.py)

This section describes the program used to train and test the convolutional neural network.

Data Loading

To start the settings for the model are set, along with the location of the training and validation datasets. TensorFlow datasets are then defined to create an image dataset from the specified directory. To check that the correct number and names of classes have been detected, it outputs these to the console. To end, the program prefetches the datasets to enable them to be used in training the model.

```
batch_size = 32
img_height = 480
img_width = 480

fileLoc = r"HAND SKELETON LOCATION HERE"

train_ds = tf.keras.utils.image_dataset_from_directory(
  fileLoc,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
  fileLoc,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

class_names = train_ds.class_names
print(class_names)
```

Listing 8: Code to Define Model Settings and Load Data

Model Creation & Training

The model is then defined as a sequential model in TensorFlow, allowing each layer to be added one after the other. There are nine layers in this model, with the reasoning behind each layer detailed in section 4.3.3.

```
model = models.Sequential()
model.add(layers.Rescaling(1./255, input_shape=(img_height,
↪  img_width, 3))),
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(5))
```

Listing 9: Code to Add Layers To the Model

A path is then defined for the checkpoint to be saved into. This checkpoint serves as a method to allow for the final artefact to load the final settings from the model. Additionally, it acts as a safety mechanism, where if the machine that is training the model crashes, the state of the model is saved after each epoch, enabling us to load the model at its previous state and continue training. The checkpoint is defined as a callback and is only used to save the weights from the training. After this is completed, the model is compiled, and finally trained, where the checkpoint callback is applied.

```
checkpoint_path = r"CHECKPOINT PATH HERE"
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create a callback that saves the model's weights
cp_callback =
↪  tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
↪  save_weights_only=True, verbose=1)
```

Listing 10: Code to Define/Create the Checkpoint

All the settings as stated in section 4.3.3 are defined and used when the model is

trained. For the training process the model is first compiled and then the model 'fit' to the data using the built in subroutine.

Model Testing

To begin the model testing, the model first needs to predict the classes from the testing dataset. The format which is received is a two-dimensional array of confidence values. This requires the index of the highest confidence to be extracted and this will directly relate to the letter being used. After this the labels in the test dataset are extracted to allow for direct comparison to the predictions. Finally, the accuracy and confusion matrix are produced to allow for direct comparison.

```
predicted = model.predict(test_ds)
predictions = argmax(predicted, axis=-1)
temp = []
for _ , labels in test_ds:
  for label in labels:
    temp.append(int(label))
print(model.evaluate(test_ds))
confusionmtrix = tf.math.confusion_matrix(temp, predictions)
confusionmtrix = confusionmtrix.numpy()
df_cm = pd.DataFrame(confusionmtrix[:13], index = [i for i in
↪  "ABCDEFGHIJKLM"],
                  columns = [i for i in
                   ↪  "ABCDEFGHIJKLMNOPQRSTUVWXYZ"])
plt.figure(figsize = (10,7))
sn.heatmap(df_cm, annot=True)
```

Listing 11: Code to Test the Model

5.4 Final Artefact (MainUI.py)

This section details the program used for the final artefact. There are two programs used for the final artefact, 'MainUI.py' and 'HandPretext.py' (described in section 5.1.1). The MainUI program is used to test the model live, and could be used in a manner to translate the BSL letters. Although only five letters are currently supported. The code for this can be seen in appendix E.

To start, the program first produces the model, this model is the same as what is used

in the training section. The layers are defined and produced in the 'create-model' function defined in the program. This model is then compiled and loaded, from the trained model checkpoint. This moves the weights across to our new empty model and allows the new model to be an exact copy of the model from the development aspect of the project.

Secondly, the live video from the device's webcam is accessed. Which when called, allows for a system to read the current webcam view and save as an image. After the webcam has been initialized, the hands object is initialized. This allows the current program to access the functions and methods from the 'handsPretext.py' (section 5.1.1) program.

Once the initial objects have been instantiated, the program enters an infinite loop. This is to enable the user to perpetually use the system. Inside the loop, an image is read from the video capture object. The image is then flipped and reshaped into a square, this new shape on the current device will be 480 by 480 by 3 (Three for the number of colour channels).

Now that the image has been processed the system adds one to a counter. This counter system is implemented to reduce latency on the system, only fully processing one image out of every ten. This reduces the amount of processing required by the system by 90%, and also reduces the amount of latency visible on the system. If the image is one to be processed, it is sent to the hands object to return an image with the hand skeleton and a black background.

```
def makeGuess(model, currentImg):
    img = np.array(currentImg)
    predicted = model.predict(img[None,:,:], verbose = 0)

    #predicted = model.predict(currentImg)

    prediction = argmax(predicted, axis=-1)[0]
    prediction = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"[prediction]
    print(prediction)
```

Listing 12: Code to Make Prediction

After which the guess is made in the system, through the 'makeGuess()' subroutine.

This subroutine converts the current image to a NumPy array, and then sends this to the model to predict which letter it is. Verbose is set to zero on this line, to stop the program from outputting this information to the command line interface. As the prediction from a single image will be an array of confidences of each potential class, the index of the maximum confidence in each row is extracted, and then the first value out of this is extracted due to there being only one image passed to the model for prediction. This number extracted directly relates to the position in the alphabet that the letter the model predicted was in the image.

Chapter 6

Results

6.1 Preliminary Results

This section discusses the results achieved from the project before the artefact was finalized, and how it affected the following development of the artefact.

6.1.1 Pretext

The pretext provided to the system can change the accuracy of the model during training. For this reason, it was imperative to choose the best method of pretext to feed to the model. In this case, the pretext for the system was provided through the use of hand skeletons.

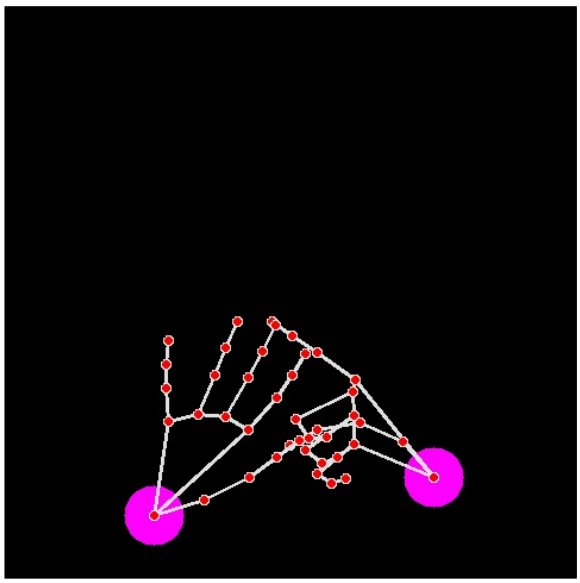

Figure 6.1: The Original Hand Skeleton Output

The original hand skeleton developed, as shown above in figure 6.1, shows each finger as white lines and the joints in red circles. When testing this on the machine learning model, using new data, the model scored very low, with low accuracy and high loss. This contradicts the values which were provided during training. To improve this, it seemed logical to add more pretext to the hand skeleton images, to enable the model to properly learn. In order to perform this addition to the pretext, each finger had different colours from one another, to allow for the machine to properly discern between fingers, shown in figure 6.2.

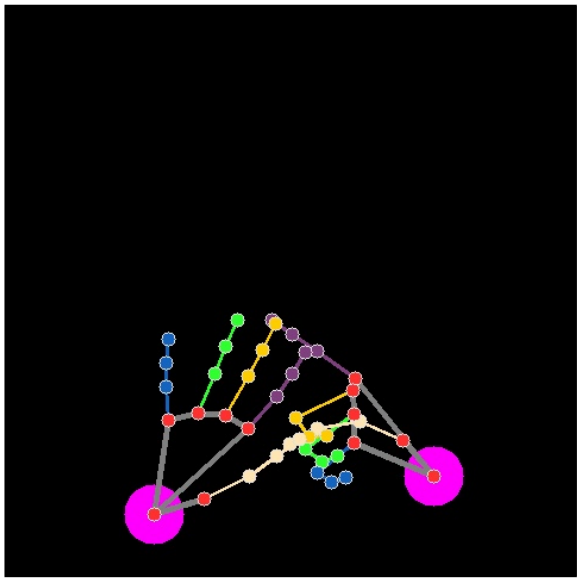

Figure 6.2: The Second Hand Skeleton Output (With Colours)

The effect of this seemed to actually decrease the training accuracy in the model, as shown in the accuracy table below. Because of this, the change was reverted and the fingers were reverted into having the same colours.

6.1.2 Class Numbers

This section will discuss the results relating to changing the number of classes and the effect it has on the accuracy of the training model.

For the class numbers tested, they were tested at differing stages, namely at: two, three, five, ten, twenty, and twenty-six classes. The reasoning for this was to speed

up the time taken to test, whilst still being able to give an idea of the class numbers that would be possible to train, the results from which are shown in figure 6.3.

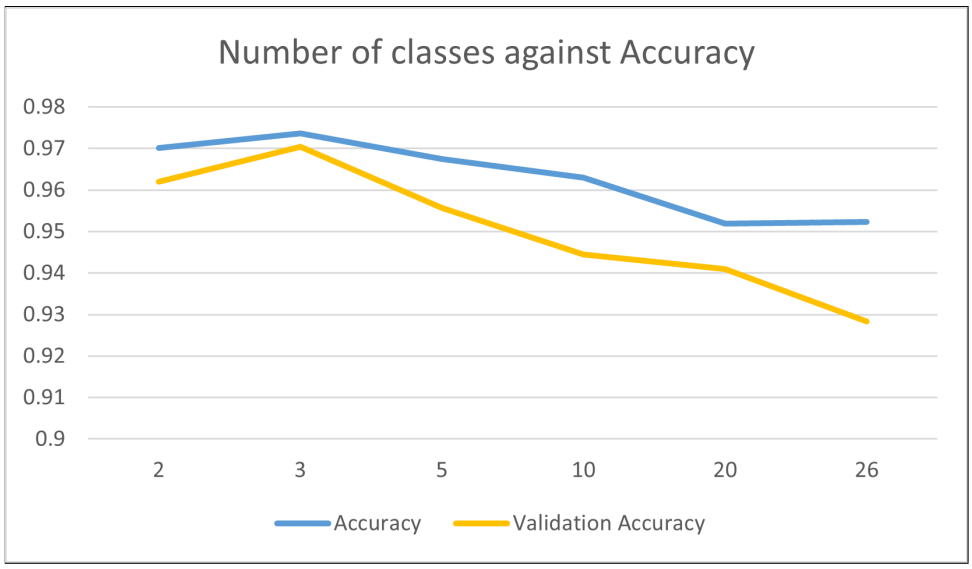

Figure 6.3: The Difference in Accuracy When Increasing The Class Numbers

When looking at the data above, the accuracy in the system clearly lowers when there are more classes for the system to classify. For this, it seems obvious that a lower number of classes would be better suited for accuracy. This was shown by the end result, where the project only managed to correctly classify five of the letters, out of twenty-six.

6.1.3 Testing Data

When testing the model, a portion of the original data was dedicated as testing data to ensure that the model was being trained correctly. This testing was evaluated through the use of confusion matrices and accuracy calculations. The confusion matrix at the end of the first test is shown in figure 6.4.

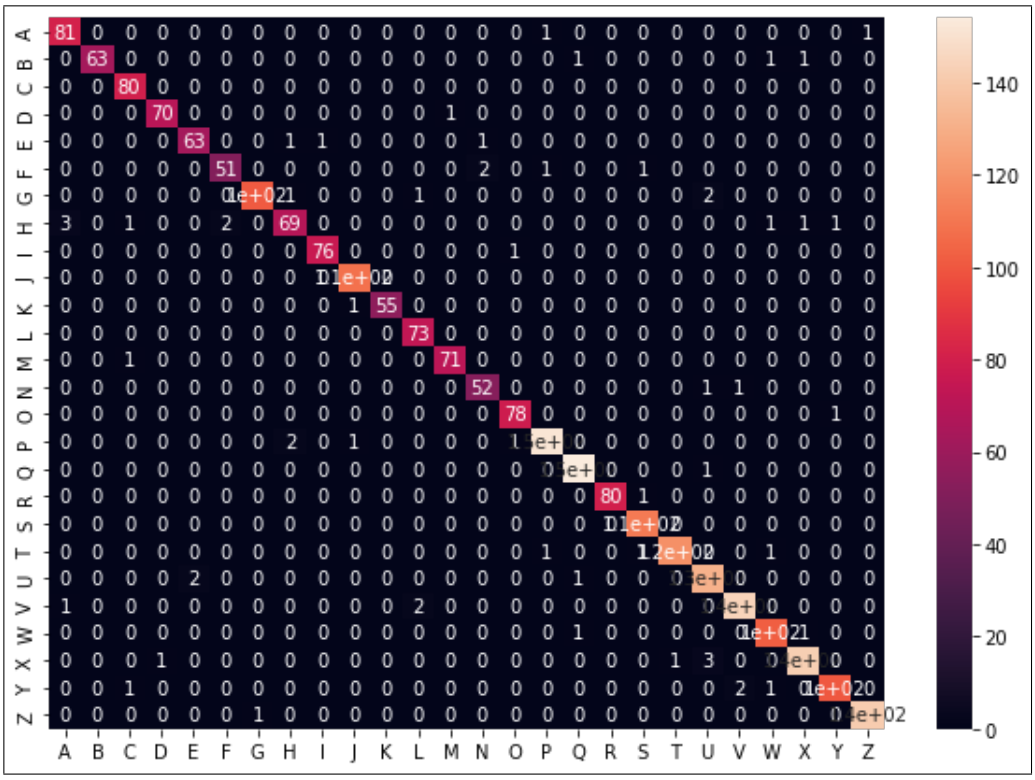

Figure 6.4: Confusion Matrix Using the Original Testing Data

The data shown in the original confusion matrix has a high correlation between the expected outcome and the actual outcome. Although this seemed like good progress, a check needed to be made to ensure that the original testing data was not too similar to the training. For this reason, a new dataset was provided, from the letters 'A' to 'M' (to allow for quicker testing whilst still being able to provide insight into the model's performance). Such confusion matrices can be seen below in figures 6.5 and 6.6.

As shown by these confusion matrices, the models are unlikely to classify the new testing data correctly. This means that the model actually has low accuracy and therefore needed modifications in order to increase accuracy. This led to the models being re-developed, and a new method for collecting the training data being developed.

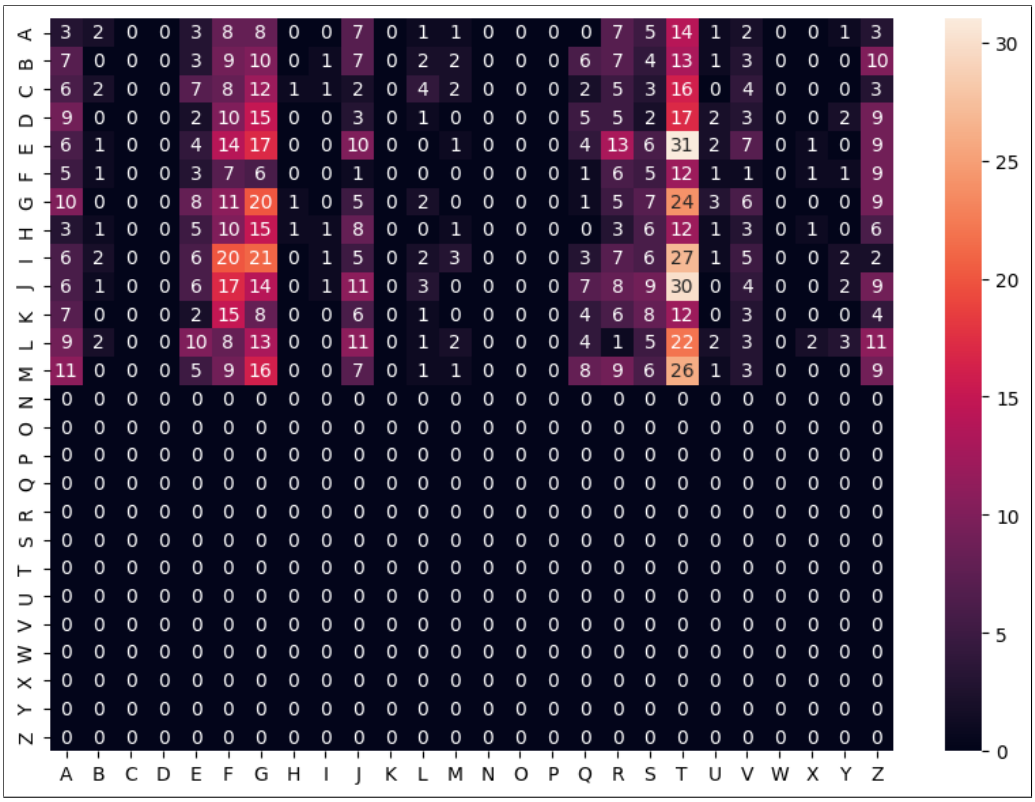

Figure 6.5: Confusion Matrix Using the New Testing Data (Original Images)

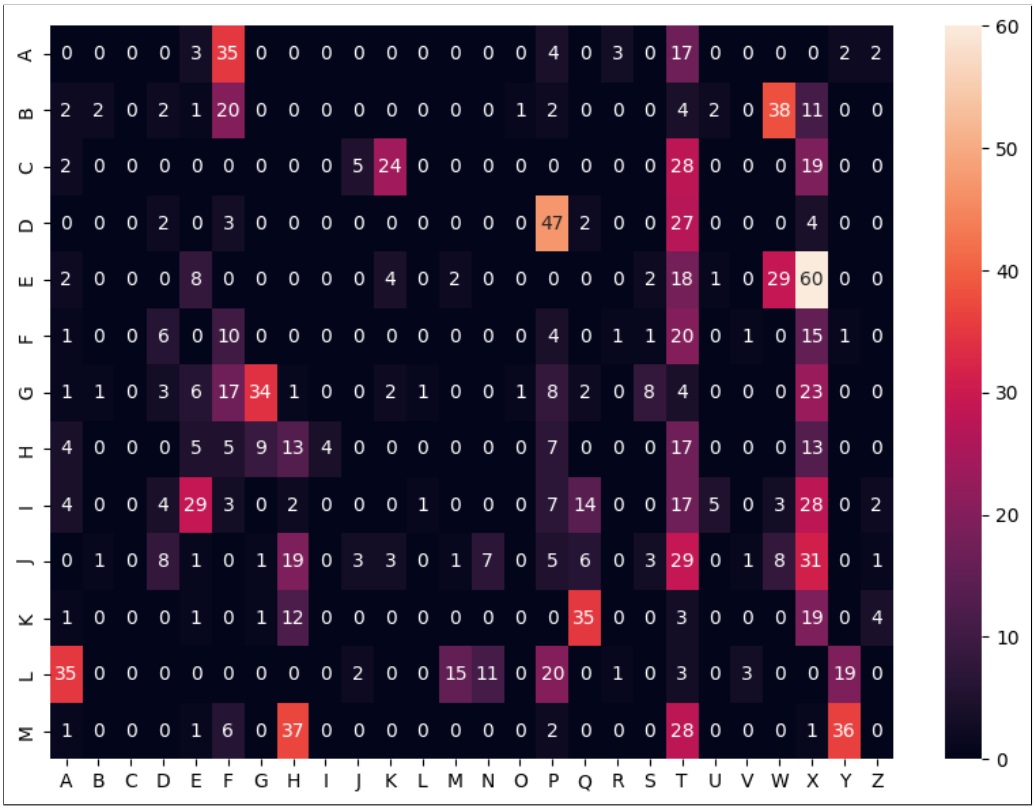

Figure 6.6: Confusion Matrix Using the New Testing Data (Hand Skeletons)

This new training data for the model was collected in a different manner from the original method. The original method called for the use of collecting videos, which then extracted each frame to collect multiple images from each video and therefore class. This led to the data being very similar to each other, leading to the models being over-trained on similar images, such that when the new testing images were shown it was unable to correctly classify a majority of the images. For this new method, a dedicated program was created for the collection (detailed in section 5.2.2), where the program took images every four seconds to enable the subject to move places and change the data provided to the system, saving the images to their correct directory. This enables more variability between the images without overtraining the system. The table provided below details the difference between the sizes of the different datasets.

Table 6.1: Table Showing the Amounts of Data in the two Datasets

	Dataset Name	
Variable	Original	New
Images Per Class	100	40
Image Count	2600	1040
Training Amount	2040	832
Validation Amount	560	208

The Testing data used for both datasets was extracted through the use of a brand new dataset, rather than using data in the above datasets. This was performed to enable the models to be equally tested against one another in an objective manner, this testing set had the letters from A to M, and an average of 97 images per letter.

6.2 Final Results

This section details the final results of the project. Including the testing and evaluation analysis as specified in section 3.3.

To start the analysis for the final model, the first check to be made was the accuracy value and loss using the testing dataset. This testing dataset, used in the evaluation

function of the model, provided an accuracy of 16% along with a loss of 27. These values would tell us that the model is unable to properly classify the first five letters in the alphabet. To ensure this is the case the next check was the use of a confusion matrix.

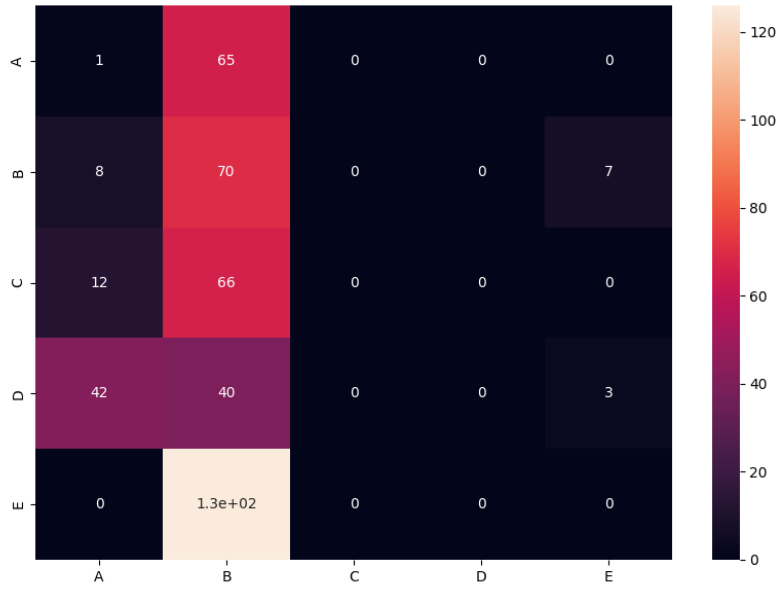

Figure 6.7: Confusion Matrix of the Final Model

As shown by the confusion matrix above, the testing data is unable to be correctly classified by the model. Most of the letters being entered into the model are being classified as the letter B, this leads to most of the classification being incorrect. When checking the accuracy within this confusion matrix, you receive roughly 16% as also shown using the evaluation function. With next to no letters being correctly classified in the model, it would be highly unlikely that any other checks would show that the model would be able to correctly classify these letters. However, the most important check for this model would be for it to be used in a live testing environment.

The final check for the model is to be used within a live testing environment. This environment proved to allow the final model to classify letters correctly. This result goes directly against the achieved metrics from the previous two stages of final test-

ing. This being said, the user needed to be a specific distance away from the camera, with sometimes the model fluctuating between some of the letters. Solutions to this could be to use an averaging algorithm, which finds the most common letter over a given number of frames, providing this answer as the most likely letter being signed by the user.

To finalize these results, the model, although theoretically should not perform well, manages to perform well when used in a live application. This could infer that the testing data is not of a quality that could be used properly by the model, however due to timeframes this could not be tested and therefore rectified.

Chapter 7

Conclusions

To conclude this project, it is useful to compare the results to the aims and objectives set out in section 1.3.

To start, the first aim of the project was to create a machine-learning model which is capable of classifying the BSL fingerspelling alphabet. Although not complete, only enabling the use of five out of twenty-six letters, in a live setting, the model was able to classify the first five letters in the BSL fingerspelling alphabet.

A secondary aim of the project was also defined, which was to use the model created in the first aim in a user-friendly interface to enable others to use the application. This specific aim was not fulfilled in the project, only through the use of a live developmental interface was a partial form of the aim completed. This aim was however not required for the purposes of the project, but rather was ideal if time permitted.

One interesting conclusion from this project was that the statistics from the model, during testing or training, such as confusion matrices or through accuracy calculations, may not be entirely representative of the usefulness of the model when compared to live testing. In this project specifically, the aforementioned statistics implied that the model would be completely unusable for its intended purpose, however live testing proved that it was the opposite case.

One positive for the final result of the project is that the use of the model does not cause latency issues when using the live testing application. This would further be indicative of the latency if the model was to be built into a fully developed applica-

tion, although not completed as part of this project. During the initial testing phases with the model, there were concerns with the latency, leading to extra development for the live testing framework, reducing the number of times that the model is used to predict the letter currently being signed. This method could be built into the final application as a method to reduce the latency and ensure that usability was prioritized.

When looking more in depth about the aims, towards the objectives, the project only partially fulfills these. Specifically, for aim one objective two, it specified the use of five differing pretext methods, however only four out of these five were developed. These included the use of no skeleton, the skeleton being laid over the original image along with the differing colours for the fingers of the skeletons. Even though a few of these were developed only three made it to the testing phase, ignoring the overlay of the skeleton.

Another objective which was overlooked was the use of three differing models for comparison purposes. This was not completed, rather only the use of a single model the CNN. This led to the inability for comparison with the possibility that the CNN was not the most effective classification model for this task.

Due to the above reasons, this project was a partial success. The model created enabled the first few letters of the BSL alphabet to be transcribed into text, however, there was no interface for this model to be fully integrated into. This project was therefore able to prove that it is possible to classify some letters and provided essential documentation for the development of a convolutional neural network for sign language. Others may be able to use this project to inform their own project in terms of the development of their networks or be able to improve directly upon the work set out in this project.

For future opportunities in this project, there would be the capability of modifying the model to enable the use of more letters, along with the full development of a user interface to encapsulate the model for public use. This interface for usability and access purposes would be best built into a mobile-based application, to ensure

that everyone who used it would have access to a camera. This would fully round out the project and enable the original aims and objectives to be fully met.

Chapter 8

Reflective Analysis

The model developed, took longer than what was originally planned and also did not fully meet the expectations set out in the aims and objectives in section 1.3. This led to the application being only partially successful in its purpose, allowing only the use of a select few amount of letters in the BSL alphabet to be correctly classified. Although time management techniques were laid out at the beginning of the project, there were some issues with the collection of data which led to time-frames being manipulated and the project taking longer than originally planned. These time management techniques allowed for this change in time frames, enabling a product to be produced at the end of the project, partially fulfilling the original aims.

With the changes made at the beginning of the project, moving the responsibility of data collection to only one person rather than looking for external signers, significantly slowed down the development of the product, as development was unable to occur whilst the data was being collected. This, in retrospect, would have been better if not changed but rather continued as planned to enable variability in the dataset along with enabling a larger dataset for training. This would have likely enabled the model to learn more letters in the alphabet whilst increasing the accuracy for the ones currently working.

In addition to this, the final solution provided by this project lacks the refinement of a fully produced product, and would likely not allow for a wide array of people to use it. With the model currently only allowing five differing letters along with the need for the signers to sign in a very similar manner to the data collected, there

are too many specific needs for the model to perform effectively to allow for a larger manner of people to use it. This also could possibly be improved with the collection of data in the originally planned manner rather than the revised manner, leading to more variability in the data and a larger set for the model to learn from.

References

Albanie, S., Varol, G., Momeni, L., Afouras, T., Chung, J.S., Fox, N. and Zisserman, A. (2007). *BSL-1k: Scaling up co-articulated sign language recognition using mouthing cues.* Available from: https://arxiv.org/pdf/2007.12131.pdf [Accessed on 01 November 2022].

Barnett, S. (2002). *Communication with Deaf and Hard-of-Hearing People: A Guide for Medical Education.* Academic Medicine 77(7). Available from: https://journals.lww.com/academicmedicine/Fulltext/2002/07000/Communication_with_Deaf_and_Hard_of_hearing.00009.aspx [Accessed on 06 February 2023].

Ben Jaab (2021). *International Week of the Deaf People 2021.* [blog]. 22 September. Available from: https://civilservice.blog.gov.uk/2021/09/22/international-week-of-the-deaf-people-2021/ [Accessed 28 October 2022].

Bierly, M. (2022). *12 Python Data Visualization Libraries.* Available from https://mode.com/blog/python-data-visualization-libraries/ [Accessed on 07/02/2023].

Central Digital & Data Office (2017). *Saleem: profoundly deaf user.* London: CDDO. Available from: https://www.gov.uk/government/publications/understanding-disabilities-and-impairments-user-profiles/saleem-profoundly-deaf-user [Accessed 02 November 2022].

Costa, C. (2020). *Best Python Libraries for Machine Learning and Deep Learning.* Available from: https://towardsdatascience.com/best-python-libraries-for-machine-learning-and-deep-learning-b0bd40c7e8c [Accessed on 11/12/2022].

Cox, S., Lincoln, M., Tryggvason, J., Nakisa, M., Wells, M., Tutt, M. and Abbot, S. (2002). *TESSA, a system to aid communication with deaf people.* Available from: https://dl.acm.org/doi/pdf/10.1145/638249.638287 [Accessed on 06 February 2023].

Elngar, A., Arafa, M., Fathy, A., Moustafa, B., O., Mahmoud, Shaban, M. and Fawzy, N. (2014). *Image Classification Based on CNN: A Survey. Journal of Cybersecurity and Information Management (JCIM)*, 6(1) 18-50. Available from: https://www.researchgate.net/profile/Ahmed-Elngar/publication/352192341_vol_6_1_2/links/60be3c0192851cb13d889f29/vol-6-1-2.pdf [Accessed on 18/01/2023].

Emond, A., Ridd, M., Sutherland, H., Allsop, L., Alexander, A. and Kyle, J. (2015). *Access to primary care affects the health of Deaf people. British Journal of General*

Practice, 65(631) 95-96. Available from: https://bjgp.org/content/65/631/95 [Accessed on 01 November 2022].

Gallard-Bridger, B. (2022). *An Electronic Reminder for Taking Medications.* Available from: https://bengallardbridger.co.uk/resources/Final_Dissertation.pdf [Accessed on 15/12/2022].

Google (2022). *Descending into ML: Training and Loss.* Available from: https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss [Accessed on 03/01/2023].

Gupta, A. (2021). *A Comprehensive Guide on Optimizers in Deep Learning.* blog. Available from: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers [Accessed on 07/02/2023].

Gupta, S. (2021). *What Is the Best Language for Machine Learning?* Available from: https://www.springboard.com/blog/data-science/best-language-for-machine-learning/ [Accessed on 10/12/2022].

Kuenburg, A., Fellinger, P. and Fellinger, J. (2015). *Health Care Access Among Deaf People.* Journal of Deaf Studies and Deaf Education, 21(1) 1-10. Available from: https://academic.oup.com/jdsde/article/21/1/1/2404217 [Accessed on 06 February 2023].

Kumar, K. (2022). *DEAF-BSL: Deep Learning Framework for British Sign Language Recognition.* ACM Transactions on Asian and Low-Resource Language Information Processing, 21(5) 1-14. Available from: https://doi.org/10.1145/3513004 [Accessed on 07 November 2022].

Kyle, J., Reilly, A.M., Allsop, L., Clark, M. and Dury, A. (2005). *Investigation of Access to Public Services in Scotland Using British Sign Language.* Available from: https://webarchive.nrscotland.gov.uk/3000/https://www.gov.scot/Publications/2005/05/23131410/14269 [Accessed on 06 February 2023].

Liwicki, S. and Everingham, M. (2009). *Automatic Recognition of Finger-spelled Words in British Sign Language.* In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 20-25 June. Leeds, UK: IEEE, 1-8. Available from: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5204291 [Accessed on 01 November 2022].

Mahendra, S. (2023). *7 Best Programming Languages for Machine Learning.* Available from: https://www.aiplusinfo.com/blog/7-best-programming-languages-for-machine-learning/ [Accessed on 11/12/2022].

MathWorks (n.d.). *What Is a Convolutional Neural Network?* Available from: https://uk.mathworks.com/discovery/convolutional-neural-network-matlab.html [Accessed on 04/05/2023].

McEwen, E. and Anton-Colver, H. (1988). *The Medical Communication of Deaf Patients.* The Journal of Family Practice, 26(3) 289-291. Available from: https://pubmed.ncbi.nlm.nih.gov/3346631/ [Accessed on 01 November 2022].

National Deaf Children's Society (n.d.). *Lip-reading.* Available from: https://www.ndcs.org.uk/information-and-support/language-and-communication/spoken-language/supporting-speaking-and-listening/lip-reading/ [Accessed on 15/01/2023].

National Institute on Deafness and Other Communication Disorders (NIDCD) (2019). *Assistive Devices for People with Hearing, Voice, Speech, or Language Disorders.* Available from: https://www.nidcd.nih.gov/health/assistive-devices-people-hearing-voice-speech-or-language-disorders [Accessed on 02 November 2022].

Njazi, S. and Ng, S. (2021). *Veritas: A Sign Language-To-Text Translator Using Machine Learning and Computer Vision.* In: 2021 The 4th International Conference on Computational Intelligence and Intelligent Systems, Tokyo Japan, 20-22 November. New York USA: ACM. Available from: https://doi.org/10.1145/3507623.3507633 [Accessed on 02 November 2022].

Office For National Statistics (2020). *Coronavirus and key workers in the UK.* Available from: https://www.ons.gov.uk/employmentandlabourmarket/peopleinwork/earningsandworkinghours/articles/coronavirusandkeyworkersintheuk/2020-05-15 [Accessed on 05/02/2023].

Rogers, K., Ferguson-Coleman, E. and Young, A. (2018). *Challenges of Realising Patient-Centered Outcomes for Deaf Patients.* The patient – Patient-Centered Outcomes Research, 11(1) 9-16. Available from: https://doi.org/10.1007/s40271-017-0260-x [Accessed on 01 November 2022].

Signhealth (2014). *A Report Into the Health of Deaf People in the UK.* Available from: https://signhealth.org.uk/wp-content/uploads/2019/12/THE-HEALTH-OF-DEAF-PEOPLE-IN-THE-UK-.pdf [Accessed on 03 November 2022].

TensorFlow (2023). *tf.keras.losses.SparseCategoricalCrossentropy.* Available from: https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentr [Accessed on 04/01/2023].

Voskoglou, C. (2017). *What is the best programming language for Machine Learning?* Available from: https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7 [Accessed on 11/12/2022].

Wikipedia (2023). *Wrapper Function.* Available from: https://en.wikipedia.org/wiki/Wrapper_function [Accessed on 24/04/2023].

Word Count 17043

Appendix A
handPretext.py

```
import mediapipe as mp
import cv2
import numpy as np
from os.path import exists
import csv

class hands:

    def __init__(self):
        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands()
        self.mpDraw = mp.solutions.drawing_utils

    def retrieveHandsOverlay(self, image, background=None,
    ↪  receiveResults=False):
        if background is None: # set background to the image if no
        ↪  background has been specified
            background = image
        imageRGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # turn the
        ↪  image to RGB
        results = self.hands.process(imageRGB) # process the image
        ↪  to receive hand information
        if results.multi_hand_landmarks:
            for handLms in results.multi_hand_landmarks: # working
            ↪  with each hand
                for id, lm in enumerate(handLms.landmark):
                    h, w, _ = image.shape
                    cx, cy = int(lm.x * w), int(lm.y * h)
                    if id == 0 :
                        cv2.circle(background, (cx, cy), 25, (255,
                        ↪  0, 255), cv2.FILLED)
                self.mpDraw.draw_landmarks(background, handLms,
                ↪  self.mpHands.HAND_CONNECTIONS)
        if (receiveResults):
            return (background, results)
        return background
```

```
    def handsBackground(self, image):
        background = np.array(np.zeros(image.shape)) # create an
        ↪  empty background for the locations to be placed on
        img = self.retrieveHandsOverlay(image, background)
        return img

    def handstoCSV(self, image, classification=None):
        headers = ["LH0x","LH0y","LH1x","LH1y","LH2x","LH2y","LH3x"
        ↪  ,"LH3y","LH4x","LH4y","LH5x","LH5y","LH6x","LH6y"
        ↪  ,"LH7x","LH7y","LH8x","LH8y","LH9x","LH9y","LH10x"
        ↪  ,"LH10y","LH11x","LH11y","LH12x","LH12y","LH13x"
        ↪  ,"LH13y","LH14x","LH14y","LH15x","LH15y","LH16x"
        ↪  ,"LH16y","LH17x","LH17y","LH18x","LH18y","LH19x"
        ↪  ,"LH19y", "LH20x",
        ↪  "LH20y","RH0x","RH0y","RH1x","RH1y","RH2x","RH2y"
        ↪  ,"RH3x","RH3y","RH4x","RH4y","RH5x","RH5y","RH6x"
        ↪  ,"RH6y","RH7x","RH7y","RH8x","RH8y","RH9x","RH9y"
        ↪  ,"RH10x","RH10y","RH11x","RH11y","RH12x","RH12y"
        ↪  ,"RH13x","RH13y","RH14x","RH14y","RH15x","RH15y"
        ↪  ,"RH16x","RH16y","RH17x","RH17y","RH18x","RH18y"
        ↪  ,"RH19x","RH19y", "RH20x",
        ↪  "RH20y"]
        if (classification is not None):
            headers.append('classification')
        if (not exists("handPos.csv")): # add the headers for the
        ↪  file if it doesnt exist
            with open("handPos.csv", "a", newline='') as csvFile:
                csvWriter = csv.writer(csvFile, delimiter=',')
                csvWriter.writerow(headers)
        outputImg, handsResult = self.retrieveHandsOverlay(image,
        ↪  None, True)

        with open("handPos.csv", "a",newline='') as csvFile: # open
        ↪  the CSV
            csvWriter = csv.writer(csvFile, delimiter=',')
            if handsResult.multi_hand_landmarks:
                if (len(handsResult.multi_hand_landmarks) <= 2 and
                ↪  len(handsResult.multi_hand_landmarks) >= 0): #
                ↪  check there are either 1 or 2 hands
                    row = [0] * 84
                    if (classification is not None): # if a
                    ↪  classification is provided, add it to the
                    ↪  end
                        row.append(classification)
                    handCounter = 0 # counter to check which number
                    ↪  hand is currently being processed
```

```
                rightHandInd = -1
                for i in range(0,
                ↪  len(handsResult.multi_hand_landmarks)):
                    value = handsResult.multi_handedness[i]
                    ↪  .classification[0].label
                    if (value == "Right"):
                        rightHandInd = i
                if (rightHandInd == -1):
                    rightHandInd = 0
                origin = handsResult
                ↪  .multi_hand_landmarks[rightHandInd]
                ↪  .landmark[0]
                h, w, c = image.shape
                origin = (int(origin.x*w) , int(origin.y*h))
                for handLms in handsResult.multi_hand_landmarks:
                ↪  # working with each hand
                    for id, lm in enumerate(handLms.landmark):
                        h, w, c = image.shape
                        cx, cy = int(lm.x * w), int(lm.y * h) #
                        ↪  get the position of the hands
                        handName = handsResult
                        ↪  .multi_handedness[handCounter]
                        ↪  .classification[0].label # get the
                        ↪  name of which hand is currently
                        ↪  being processed
                        if (len(handsResult
                        ↪  .multi_hand_landmarks)==1): #assume
                        ↪  that if only one hand is on the
                        ↪  screen it is the right hand
                            offset=1
                        else: #if there is more than one hand -
                        ↪  check the hands' type to put it in
                        ↪  the right category
                            if handName == 'Left':
                                offset = 0
                            else:
                                offset = 1
                        cx = origin[0]-cx
                        cy = origin[1]-cy
                        row[(id*2)+(42*offset)] = str(cx)
                        row[(id*2) +(42*offset)+1] = str(cy)
                    handCounter+=1
                csvWriter.writerow(row)
    return outputImg #remove when in use just for testing
```

Appendix B
LoadData.py

```
import pandas as pd
import tensorflow as tf
import numpy as np
import os
import cv2
import Modules.handPretext as hands
class loadData:
    def loadTrainTest(csv):
        dataframe = pd.read_csv(csv)

        dataframe['classification'] =
        ↪  pd.Categorical(dataframe['classification'])
        dataframe['classification'] =
        ↪  dataframe.classification.cat.codes

        targetVars = dataframe.pop('classification')
        tf_dataset =
        ↪  tf.data.Dataset.from_tensor_slices((dataframe.values,
        ↪  targetVars.values))

        trainSize = int(0.7*len(dataframe))
        testSize = int(0.2*len(dataframe))

        tf_dataset.shuffle()

        train = tf_dataset.take(trainSize)
        test = tf_dataset.skip(trainSize).take(testSize)
        val = tf_dataset.skip(trainSize+testSize)

        return train, test, val

    def loadFromFolders(parentFolderLocation):
        handProcessor = hands.hands()
        for imageFolder in os.scandir(parentFolderLocation):
            for imagePath in os.scandir(imageFolder.path):
                image = cv2.imread(imagePath.path)
```

```
            _ = handProcessor.handstoCSV(image,
            ↪  imageFolder.path[-1])

def createHandSkeletons(parentFolderLocation, handImgLocation):
    handProcessor = hands.hands()
    for imageFolder in os.scandir(parentFolderLocation):
        currentLetter = os.path.split(imageFolder)[1]
        currentPath = handImgLocation + "\\" + currentLetter
        os.chdir(currentPath)
        for imagePath in os.scandir(imageFolder.path):
            image = cv2.imread(imagePath.path)
            newImg = handProcessor.handsBackground(image)
            imagename = os.path.split(imagePath)[1]
            cv2.imwrite(imagename,newImg)

def videoToImages(parentFolderLocation, imagePath):
    for videoPath in os.scandir(parentFolderLocation):
        currentLetter = os.path.split(videoPath)[1][0]
        imagePathTemp = imagePath + "\\" + currentLetter
        os.chdir(imagePathTemp)
        video = cv2.VideoCapture(videoPath.path)
        success,image = video.read()
        count = 0
        while success:
            cv2.imwrite(currentLetter + "%d.jpg" % count, image)
            success,image = video.read()
            count += 1
```

Appendix C
DataSetCreation.py

```
from LoadData import loadData

imageFolderLocation = r"NORMAL IMAGE PATH HERE"
handSkeletonFolderLocation = r"HAND SKELETON PATH HERE"
videoFolderLocation = r"VIDEO PATH HERE"
loadData.loadFromFolders(imageFolderLocation)
loadData.videoToImages(videoFolderLocation, imageFolderLocation)
loadData.createHandSkeletons(imageFolderLocation,
↪  handSkeletonFolderLocation)
```

Appendix D
DataSetCreateLive.py

```
import cv2
import os
import numpy as np
import datetime

cap = cv2.VideoCapture(0) # activate the video capture
letter = input("Enter the letter you are signing: ").upper()
path = r"PATH OF ORIGINAL PICTURES HERE"
path = os.path.join(path, letter)
if (not os.path.exists(path)):
    os.makedirs(path)
previousTime = datetime.datetime.now()
count= len(os.listdir(path));
while True:
    success, image = cap.read()
    image = cv2.flip(image, flipCode=1)
    image = image[0:image.shape[0], 0:image.shape[0]]
    currentTime = datetime.datetime.now()
    if ((currentTime - previousTime).seconds >= 4):
        count +=1
        previousTime = currentTime
        #save image
        filename = str(count) + ".jpg"
        newPath = os.path.join(path,filename)
        cv2.imwrite(newPath, image)
        image = np.ones(image.shape)
    cv2.putText(image,(str)(4 - (currentTime -
    ↪  previousTime).seconds),(50,100), cv2.FONT_HERSHEY_SIMPLEX,
    ↪  4, (255, 255, 255), 2, cv2.LINE_AA)
    cv2.putText(image,(str)(count),(50,300),
    ↪  cv2.FONT_HERSHEY_SIMPLEX, 4, (255, 255, 255), 2,
    ↪  cv2.LINE_AA)
    cv2.imshow("Output",image)
    cv2.setWindowProperty("Output", cv2.WND_PROP_TOPMOST, 1)
    cv2.waitKey(1)
```

Appendix E
MainUI.py

```
import tensorflow as tf
from tensorflow import keras
from keras import datasets, layers, models
import matplotlib.pyplot as plt
import os
from keras.backend import argmax
from Modules import handPretext
import cv2
import numpy as np

def create_model():
    batch_size = 32
    img_height = 480
    img_width = 480
    model = models.Sequential()
    model.add(layers.Rescaling(1./255, input_shape=(img_height,
    ↪  img_width, 3))),
    model.add(layers.Conv2D(32, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(6))

    model.compile(optimizer='adam',
              loss=tf.keras.losses
              ↪  .SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

    checkpoint_path =
    ↪  r'C:\Users\benGa\Documents\University\Masters\MComp Research
    ↪  Project\Programming
    ↪  Repo\MComp-Research-Project-2022-23\Checkpoints\cp6.ckpt'
```

```
    model.load_weights(checkpoint_path)

    return model

def makeGuess(model, currentImg):
    img = np.array(currentImg)
    predicted = model.predict(img[None,:,:], verbose = 0)

    #predicted = model.predict(currentImg)

    prediction = argmax(predicted, axis=-1)[0]
    prediction = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"[prediction]
    print(prediction)

currentModel = create_model()

cap = cv2.VideoCapture(0) # activate the video capture
hand = handPretext.hands() # create an object to process the hand
↪  tracking pretext tasks
counter = 0
while True:
    success, originalImg = cap.read()
    originalImg = cv2.flip(originalImg, flipCode=1)
    originalImg = originalImg[0:originalImg.shape[0],
    ↪  0:originalImg.shape[0]]
    processedImg = hand.handsBackground(originalImg)
    counter = counter + 1
    if (counter >4):
        makeGuess(currentModel, processedImg)
        counter = 0
    cv2.imshow("Output",originalImg)
    cv2.waitKey(1)
```

Appendix F
CNNModel.py

```
import tensorflow as tf
from tensorflow import keras
from keras import datasets, layers, models
import matplotlib.pyplot as plt
import os

batch_size = 32
img_height = 480
img_width = 480

fileLoc = r"HAND SKELETON LOCATION HERE"

train_ds = tf.keras.utils.image_dataset_from_directory(
  fileLoc,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
  fileLoc,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

class_names = train_ds.class_names
print(class_names)

AUTOTUNE = tf.data.AUTOTUNE

train_ds =
 ↪  train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
#Model taking in only the images

model = models.Sequential()
model.add(layers.Rescaling(1./255, input_shape=(img_height,
↪  img_width, 3))),
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(5))

checkpoint_path = r"CHECKPOINT PATH HERE"
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create a callback that saves the model's weights
cp_callback =
↪  tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
↪  save_weights_only=True, verbose=1)

model.compile(optimizer='adam',
              loss=tf.keras.losses
               ↪  .SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs,
  callbacks=[cp_callback]
)
```

Appendix G
Application Code

The full program can be downloaded and viewed at this GitHub Repository: `https://github.com/BenGallardBridger/MComp-Research-Project-2022-23`