MSC APPLIED ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

SEPTEMBER 2022

GABRIEL OSI BRAIMAH

BRIDGING THE COMMUNICATION GAP BETWEEN THE PUBLIC AND BSL SIGNERS WITH SEVERE HEARING LOSS USING ARTIFICIAL INTELLIGENCE

SOLENT UNIVERSITY

FACULTY OF BUSINESS, LAW AND DIGITAL TECHNOLOGIES

SOUTHAMPTON SOLENT UNIVERSITY

FACULTY OF BUSINESS, LAW AND DIGITAL TECHNOLOGIES

BRIDGING THE COMMUNICATION GAP BETWEEN THE PUBLIC AND BSL SIGNERS WITH SEVERE HEARING LOSS USING ARTIFICIAL INTELLIGENCE

BY

GABRIEL OSI BRAIMAH
STUDENT NUMBER: Q15695786

SUPERVISOR: Dr FEMI ISAAQ
SEPTEMBER 2022

PROJECT IN PART FULFILMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTERS

IN

APPLIED ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Abstract

Finding an effective artificial intelligence real time sign language recognition translator is still a barrier even with recent advancements in computer vision and machine learning more so for British sign language users as bulk of the research has been done on American and Indian sign language. There have been methods proposed to tackle the challenges of building an effective SLR translator using data gloves, arm sensors and computer vision. Every other method aside the computer vision method is intrusive, computationally intensive and can be inconvenient which makes them unscalable at least for now. This work aims to present a vision-based realistic approach to bridging the communication gap between the public and the deaf-dumb community of British sign language users using a real time BSL to English language translator capable of recognizing both static and dynamic BSL gestures. A webapp was built using the Streamlit framework which hosts the deep learning systems trained to recognize the BSL signs. The system uses a webcam to feed live images to the mediapipe holistic model which detects the pose and hand landmarks, the landmarks are then extracted, pre-processed, and fed to the deep learning models for classification with each model achieving over 90% accuracy on test data and the GRU model having 85% accuracy in real time test using just the hand landmarks to make prediction.

Acknowledgement

I wish to thank the almighty for giving me this privilege to be in a position where I can learn so much. I would also like to thank my supervisor Dr Femi Isiaq for pushing me to be the best and I can be and meticulously guiding me through the process. I would also like to thank all my friends that have been supportive and have made Solent university a pleasurable experience.

Finally, I would like to thank my family for giving me the strength needed to keep pushing. I would not have made it this far without their loving support.

Table of Contents

Table Of Figures

Chapter 1

1.1 Chapter overview

This chapter gives a brief history on sign language and the challenges faced by people with severe hearing loss. A brief history of artificial intelligence and its sub areas are also touched. The following areas were also outlined in the section.

- Research question
- Problem statement
- Value proposition
- Success metrics
- Aim
- Objectives

1.2 Introduction

Communication is a very essential part of the daily lives of all animals because it is the only way to express wants and needs, socialize, share knowledge and as humans being very expressive this is even more true. Communication can be verbal, textual, or done through visualizations. Majority of human-to-human communication is done verbally while a significant number of the deaf-dumb community communicate using sign language.

Sign language can be described as a language which combines hand gestures and body language to convey a message [Rustagi, Shaina and Singh, 2021]. As different verbal languages exist throughout the world so do different sign languages exist throughout different regions of the world which makes it difficult to have a standardized sign language which can then be used to build a universal sign language recognition system. The United Kingdom for example has both the British Sign language which is focused more on fingerspelling and Makaton which takes a more general approach by using signs and symbols in conjunction with spoken words. The price and relative availability of human sign language translators mean hearing impaired people cannot easily have access to these translators when one is needed.

British sign language (BSL) like Indian sign language (ISL) majorly consists of using two hands to form signs and fingerspell words except for the alphabet “C” which uses one hand while sign language like American sign language use one hand (ASL) to make signs. The British sign language is used by 151,000 people in the United Kingdom as stated in the united kingdom’s government’s website. There are only 1540 registered BSL interpreters according to the National Registers of Communication Professional as of 2019 which is just about 1% of the total users of BSL.

Artificial intelligence is a broad field encompassing several areas like machine learning, robotics, computer vision, and deep learning. Artificial intelligence has been referred to as the scientific approach adopted to make machines smart

[Buckley, Sherrett and Secco, 2021]. The influence Artificial intelligence has on our everyday lives cannot be overstated, it has become so prevalent in our society, dominated the decision-making process of businesses, and permeated all levels of our social structure, from the likes of Alexa and Siri to self-driving cars to fraud detection systems used by banks, the list continues to grow.

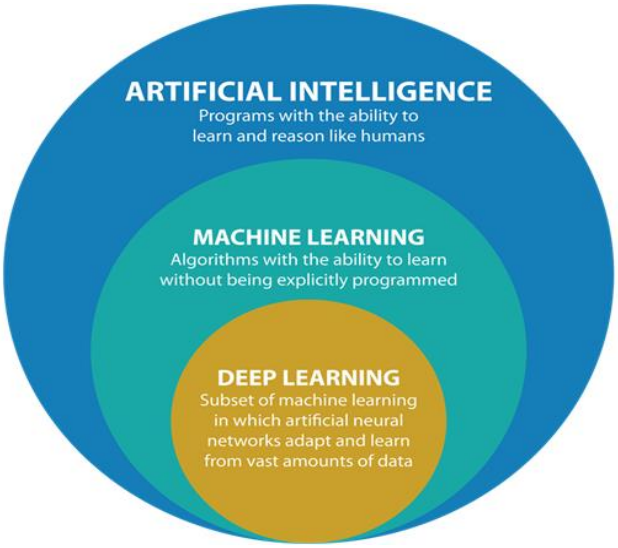


Figure 1: Artificial intelligence

Computer vision is a field of artificial intelligence that deals with the development of computers systems capable of extracting information from images, videos, any visual inputs, according to international business machines (IBM). The applications of computer visions systems in an ideal sense are limitless just as the human sight is to humans. Computer vision is already used in industries like medicine, manufacturing, self-driving cars, surveillance systems, and the service sector.

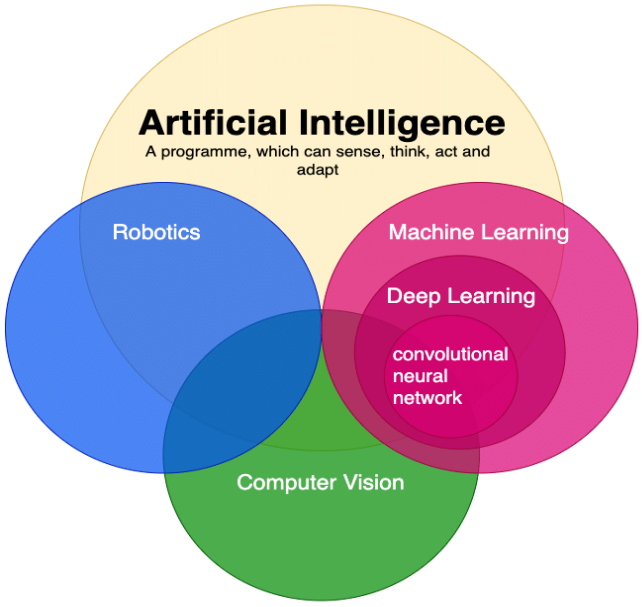


Figure 2:Relation between Artificial Intelligence, Machine Learning and Deep Learning, Computer Vision. [Mohammed 2022]

Sign language recognition systems brings together computer vision, machine learning or deep learning models used for classification and in some cases other areas of software development like webapp development using streamlit which was used in this work.

1.3 Background

Since the advancement in computer vision and artificial intelligence the possibility of building SLR systems became ever more realistic, and a lot of studies have been done on realising such possibility as shown by (ADEYANJU, I.A., O.O. BELLO and M.A. ADEGBOYE, 2021).

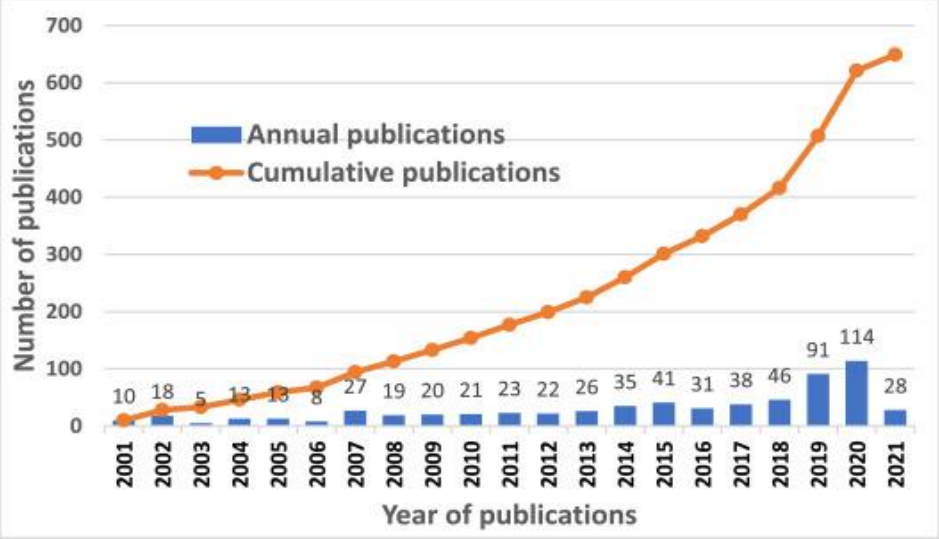


Figure 3: cumulative publications on intelligent SLR systems from the scopus database (Adeyanju 2021)

The bulk of the research on such intelligent systems have gone to other sign languages aside the BSL as a simple search as at the time of writing this paper on the ieeexplore.org website using the combination “British sign language” AND “machine learning” yields just two results while “British sign language” AND “deep learning” yields 3 results with one focusing on gestures which makes it semi-irrelevant to this study.

Two main methods have been adopted by the community in the development of these systems, and different techniques have been introduced to the data collection, data pre-processing and classification steps in combination with several machine learning and deep learning algorithms. One main method involves the wearing special gloves capable of tracking the movement of the fingers as done by (Y. MORI and M. TOYONAGA, 2018) and (H. S. ANUPAMA *et al.*, 2021) while the other main method is vison based and it involves using everyday multimedia cameras as done (KATOCH, S., V. SINGH and U.S. TIWARY, 2022), (KASAPBAŞI, A. *et al.*, 2022) and several others. For sign language systems to have its desired societal function it should be easily accessible, the best approach is to create a system

where the images are gotten from normal everyday cameras available in our electronic devices as it is easily accessible and cheap.

1.4 Research question

is it possible to bridge the communication gap between BSL signers with severe hearing loss and the public using a visual-based SLR artificial intelligence method.

1.5 Problem Statement

The importance of communication in our everyday lives cannot be overstated as it forms the backbone of all our social activities. 466 million people suffer from different levels of hearing loss as stated by the world health organization, this is 5% of the world's population with the number expected to increase in the coming years. 11 million people suffer from severe hearing loss and 151,000 of them communicate using British sign language according to the United Kingdom's government. The United Kingdom website on the statistics on people with severe hearing loss states that, Half of these vulnerable individuals are likely to have poor mental health and 35% are likely to be unemployed. The data shows that due to the communication gap these vulnerable individuals with severe hearing loss cannot fully access the services available to the public, are more likely to be poor and more like to suffer from mental issues.

1.6 Value Proposition

Having a BSL recognition system that truly bridges the communication gap between these vulnerable individuals and the public will have some benefits which includes but not limited to

- Improved Standard of living
- Easier Access to medical services
- Better integration into the society
- Technology pushes people to learn new things because of the ease of access and use so having such a system will make learning BSL easy and hereby further help bridge the communication gap.
- Carers don't have to understand BSL before providing care to vulnerable people using BSL recognition systems.

1.7 Success Metrics

The success metric for this work is set at 90% and above in a real-world scenario and not a controlled setting as this work is aimed at taking a realistic step toward building a BSL recognition system that can be commercially scalable in the future.

1.8 Aim

The aim of this project is to develop a BSL recognition system that takes a substantial first step towards bridging the communication gap between BSL users and the public.

1.9 Objectives

Meeting the aim of this project require the objectives stated below be met.

- Gather a suitable BSL dataset
- Pre-process the generated data
- Train and evaluate different models
- Form words with classified labels
- Build a webapp using streamlit to host the BSL recognition system.

Chapter 2

2.1 Related Studies

2.1.1 Chapter overview

A total of 15 papers were collected and reviewed from both ScienceDirect and ieeexplore.org. This chapter starts with a review of the studies done using the wearables approach followed by limitations of that approach. The chapter then moves on to the literature review about the visual methods focusing on the work done using CNN in the second paragraph as it is the dominant algorithm used when dealing with images. The third paragraph focuses on the work done in SLR using mediapipe. The fourth paragraph gives suggestions on merging both the wearables and vision approach for SLR is given having reviewed the work done in the papers. Finally, the chapter ends with a table of the reviewed works in mediapipe itemizing the important aspects as it relates to this work and giving my comments on each study.

2.1.2 Wearables method review

There are two approaches used in creating a sign language recognition system which are vision based and wearables-based method. The wearable methods make use of special gloves or arm sensors. Data-glove technology converts finger positions to electrical impulses to determine the posture of the hands.

(Y. MORI and M. TOYONAGA, 2018) proposed a method for developing SLR system for Japanese sign language that involved connecting data gloves with flex sensors and accelerometer and a gyro sensor to detect hand motion and shape of each finger this analogue signal is then converted to digital data by an Arduino data control unit which is then sent to a decision system, The study achieved an accuracy of 51%. The design of this approach makes it inconvenient to use coupled with the low accuracy makes the proposed method unscalable

(H. S. ANUPAMA *et al.,* 202) also used a similar approach to (Y. MORI and M. TOYONAGA, 2018) but used it with supervised learning process using the k nearest neighbour and other algorithms. The study also added an audio device to the gloves that converts the signs to spoken phrases. The k-nearest neighbour achieved an accuracy of 93% in this study. Like (Y. MORI and M. TOYONAGA, 2018) the design makes it unscalable. No real time results was reported for the study to check the accuracy.

(N. TUBAIZ, T. SHANABLEH and K. ASSALEH, 2015) used a modified K-Nearest neighbour as well as a DG5-V hand gloves embedded with flex sensor and a 3 axes accelerometer, the glove is also wireless and uses a battery as opposed to previous works, the proposed method had an accuracy of 98.9%. Data glove technologies for sign language recognition attempts to circumvent the environmental limitations of vision-based approaches and these proposed data glove designs can be considered inconvenient and restrictive by users although
(A. ABDULLAH, N. A. ABDUL-KADIR and F. K. CHE HARUN, 2020) proposed a less bulky design with fewer sensors, it is still not compact and convenient enough for everyday use.

2.1.3 visual-based method review

Vision-based approach uses a webcam or in some cases phone cameras to pass in the required data to the sign language recognition systems. Vision based approaches are non-invasive and do not require any dedicated hardware making it convenient and the most commercially scalable of the two methods of SLR systems. The vision-based approach like the data gloves approach has had a lot of research done using different machine learning and deep learning techniques. Convolutional neural network (CNN) is used extensively in vision-based approaches because of its dominance in computer vision. CNN is well suited to work on grid like data which is what images are. CNN also has an automatic feature extraction which implies that sometimes little work is done to extract the features from the images before they are being fed into the CNN model.

(KASAPBAŞI, A. *et al.,* 2022) developed an SLR system for ASL and achieved an accuracy of 99.38% with a 0.0250 loss. Like (KATOCH, S., V. SINGH and U.S.

TIWARY, 2022) and most vision-based approaches, image segmentation and sometimes background subtraction accompanied by specifying a region of interest (ROI) where the signs must be done before any classification can take place is a common practice when building vision based SLR systems. The study also involved training the CNN model using black and white images and preprocessing the signs to black and white during real time testing.

(KATOCH, S., V. SINGH and U.S. TIWARY, 2022) used several preprocessing steps like skin segmentation, gaussian filtering, SURF feature extraction and clustering to generate the data used in the ISL recognition system. The CNN model in this study performed marginally better than the SVM and achieved an accuracy of 99.64% on test data. The real time test was done in an environmental setting that gives contrast to the hands and therefore designed to maximize the accuracy rather than using the same method used to generate the data for the study.

(N. BUCKLEY, L. SHERRETT and E. LINDO SECCO, 2021) developed a BSL recognition system on 19 static gestures. The system is powered by a CNN model and performed the typical pre-processing steps including image binarization.

Using a combination of AlexNet which is a version of CNN, and LSTM (W. SULIMAN *et al.*, 2021) achieved an accuracy of 95.9% although it was signer dependent. The proposed system developed for Arabic sign language (ARSL) involved using AlexNet for feature extraction and LSTM for classification. The system in the pre-processing step used skin detection and face detection for image segmentation, further image processing was carried out using morphological operations to remove small regions, finally the two largest regions which are the hands were extracted and used to form the dataset. The problem with such an approach the user needs to always show the bare arms for the process to work which will not be the case most times

(M. QUINN and J. I. OLSZEWSKA, 2019) developed a BSL recognition system powered by SVM but used a different approach in the pre-processing step that involved downscaling then using several computer vision morphological operations such eroding and dilation combined with histogram of oriented gradients before performing feature extraction and finally upscaling to produce a HOG image. The study showed good performance regardless of the environment on random gestures and did not give any results on the real time tests using BSL signs.

Developing SLR systems using CNN requires lots of pre-processing and is easily affected by environmental factors coupled with the fact that CNN requires a lot of data to train that is why most researchers using this method specify a region of interest to reduce the number of features per image which helps the model train faster. CNN is not as fast as other neural networks like LSTM so using such a model in real time prediction might be a hassle coupled with the fact that the average signing rate is 2.3-2.5 signs per second (JEAN, B.G., 1979). Additionally, this approach is not suitable for everyday life as so many environmental factors can easily affect the useability of such systems. Another approach adopted in recent years is the use of mediapipe. Most of the hand detection is already done by the framework so researchers can focus on building the AI systems that take advantage

of the mediapipe framework. Mediapipe also come with several other advantages that make it suitable for real time prediction as will be discussed in later chapters.

2.1.4 Mediapipe method review

(M. MARAIS *et al.*, 2022) performed Argentine sign language recognition using hand landmarks extracted with mediapipe holistic model which was then used to train a 1D-CNN model used in classifying the signs. Also, another method involving the segmentation of hands from the entire image was done using image thresholding with the researchers wearing fluorescent gloves and final images fed to the Pruned VGG and ResNet models. Finally, raw images were fed to a Pruned VGG and ResNet models for training. The 1D-CNN plus mediapipe combination got an accuracy of 94.91% on test data, method 2 got accuracies of 47.83% and 43.24% on ResNet and Pruned VGG model respectively. Method 3 got an accuracy of 95.40% on the Prunned VGG model slightly edging out the ResNet model. The final 1D-CNN model was not tested in real time to check for real world accuracy and the study was not very detailed.

(D. BISHT *et al.*, 2022) uses a web interface that hosts an American sign language recognition system powered by a random forest classifier which achieved an accuracy of 94.69%. The study also implemented many features like speech to text, text to speech and autocorrect but the method used to predict in real time makes it difficult to commercially scale and improve as real time prediction is more of action recognition rather than static sign language recognition.

(V. H. IYER *et al.*, 2022.) implemented a system trained to recognize 3 classes using mediapipe for face, pose and hand landmarks extraction and trained an LSTM model, the study achieved an accuracy of 87.5% for the 3 labels which is not adequate compared to other studies like (M. MARAIS *et al.*, 2022),(M. H. ISMAIL, S. A. DAWWD and F. H. ALI, 2021.) and (D. BISHT *et al.*, 2022) which achieved higher accuracy on full datasets.

(M. H. ISMAIL, S. A. DAWWD and F. H. ALI, 2021) developed an SLR system classifying signs into static, dynamic and non-sign. The data comprised of 7500 videos with mediapipe used to extracts the pose, and hand landmarks. A Bi-GRU was then trained and achieved an accuracy of 99.05%.

(A. CHAIKAEW, K. SOMKUAN and T. YUYEN, 2021.) developed a system for Thai sign language recognition system for both desktop and smartphone picking 5 labels with 100 videos per label. The LSTM model used in this study achieved an accuracy of 97% on test data.

(S. ADHIKARY, A. K. TALUKDAR) developed a system recognizing 11 ISL gestures using mediapipe in combination with different models to perform SLR but failed to detail the clear method used to achieve the results. Also the work was not done on the full ISL dataset.

A hybrid method can be used to build, the wearables for collecting data and building the numeric dataset to ensure the accuracy of the data being collected while a vision-based approach using mediapipe holistic model can be used to extract the position of the hands from a live feed camera during data real-time testing.

Study	Method	Models implemented	Accuracy %	Real time Prediction	Background Dependency	User Interface	Comments
M. MARAIS *et al.*, 2022	Worked on the LSA64 dataset and experimented with 3 different methods with the raw image processing having the highest accuracy on test data	1D-CNN ResNet Pruned VGG	94.91 95.40 94.50	No	Yes	No	The study explored different methods for SLR but was not very detailed.
D. BISHT *et al.*, 2022	Trained different models on ASL dataset and few ISL alphabets using mediapipe to extract the hand features	RandomForestClassifier Naive Bayes Classifier Logistic regression KNN SVC LwP	94.58 79.74 89.95 89.15 90.91 58.45	yes	no	yes	A good study overall but the real time prediction using each frame rather than a combination of frames for the sign recognition is not suitable for commercial scaling and can't be used for action recognition which is what real time sign language prediction is geared towards.
V. H. IYER *et al.*, 2022.	Trained an LSTM model on 3 labels using mediapipe to extract features	LSTM	87.5	yes	no	yes	Poor accuracy on few labels, Face datapoints not needed to perform the described prediction
M. H. ISMAIL, S. A. DAWWD and F. H. ALI, 2021.	Implemented a system that recognizes Arabic sign language into static, dynamic and non-sign powered by a Bi-GRU model classifying the signs from video frames	Bi-GRU	99.05	no	yes	no	did not test the model in real time to confirm the accuracy
A. CHAIKAEW, K. SOMKUAN and T. YUYEN, 2021.	Designed a SLR system for Thai sign language	LSTM Bi-LSTM GRU	97 94 94	no	yes	no	No real time testing was done to validate the accuracy of the models
S. ADHIKARY, A. K. TALUKDAR	Implemented a system on ISL recognition with 11 gestures	Decision Tree Classifier Randomforestclassifier Gradient boosting classifier	83.8 97.4 95.7	yes	no	yes	Methodology on the data preprocessing and model training is unclear. The number of images per class wan not defined. Therefore, interpreting the type of confusion matrix presented without the number per class is not possible.

Chapter 3 Methodology

3.1 Chapter Overview

All libraries, algorithms and tools used in the building of the artefact are discussed in this chapter. The chapter begins with the proposed method for this study then proceeds to talk about the dataset and where it was gotten from followed by the discussion of libraries and tools

3.2 Proposed Method

Before this proposed method was reached 2 other methods were tried. A total of 260 images was collected for the 26 classes of the BSL dataset for both right and left hands from a secondary signer with each class having 10 images each, 8 for training and 2 for testing. The first method involved creating a directory structure having train and test folders as required by CNN. The created folders will have a folder for each of the alphabets with 8 images of each alphabet going to their respective sub folders in the train folder and the remaining 2 goes to their associated sub folders in the test folder. During training it became very clear a lot more images was needed to be generated to achieve any real accuracy as the validation accuracy did not get passed 6% because the pilot study conducted showed one of the major drawbacks of CNN is the amount of data it needs to train. Given the time constraints and potential challenges that comes with generating a robust data for such a project another approach was needed.

The next approach also used images collected in the first step but this time with a pretrained mobilenet SSD model in combination with TensorFlow object detection. All the image files with their associated xml files were split into training and test files and placed in the train and test folders with no sub folders for the alphabets needed like in the first method. In this approach a form of directed learning was done using image labelling. The images were labelled using an application called labelImg which generates an xml file for the image being labelled. The signs in the image were selected and the associated English alphabet was used to label the image. After testing the first time there was no real accuracy observed during real time test. After generating the TensorFlow record and changing the configuration file the model still did not learn. Given the time constraint the method using mediapipe needed to be explored.

This method uses the OpenCV library to access the web camera which takes images and passes these images frame by frame to the google mediapipe holistic model for pose, left and right-hand landmarks detection which are then extracted into a NumPy array and stored using the npy extension. These NumPy files are then loaded and preprocessed before been passed to the models for training. The models are then evaluated using accuracy, precision, F1 score and recall. The selected models are then deployed using the h5 format and a webapp using Streamlit is built to perform real time BSL recognition testing.

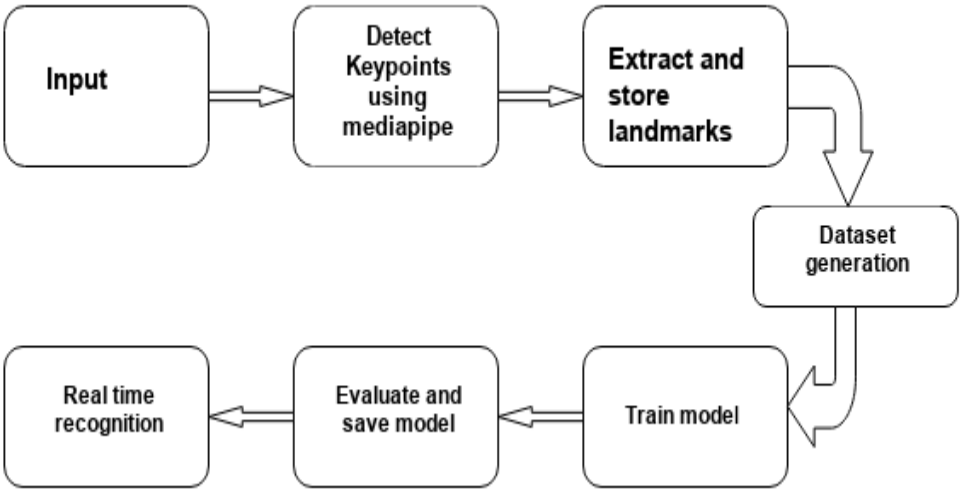


Figure 4: Block diagram of intended model

3.2.1 Dataset

The base line in which the data was generated was gotten from the university college London (UCL). It has the alphabets A-Z. The signs in this dataset were followed when generating the data needed to train the models. A function was utilized in the data collection process, each frame is read from the webcam then passed to the mediapipe holistic model for landmark extraction. Each alphabet had 60 folders with each folder having 30 files with the npy extension representing the extracted landmarks of each frame from 1-30, as 30 frames per second was used. It was ensured that the extracted key points covered just the arm pose landmark excluding other body landmarks, left and right landmarks. The data generated from this process was for left hand only. The 26 alphabets were used for the final model. finally, signs were generated for the "." and empty space character.

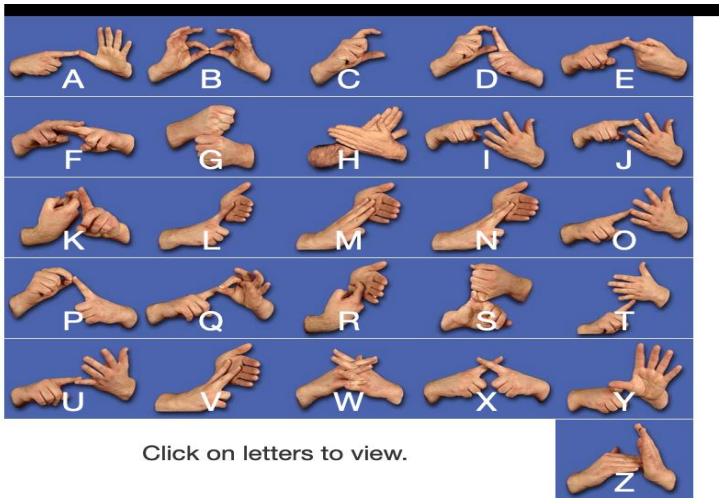


Figure 5:sample of dataset

3.2.2 OpenCV

Computer vision has it challenges and solving those challenges will bring about possibilities in self driving, engineering, entertainment, and other areas. Intel in 1999 officially launched OpenCV (open-source computer vision library) (I. CULJAK *et al.*, 2012) which is used mainly for image processing (NAVEENKUMAR, M. and A. VADIVEL, 2015.). OpenCV is built using C++ which makes it portable and highly optimized. OpenCV has lots of functions and algorithms used for edge detection, object tracking and face detection. For this work OpenCV will be used to access the webcam and read the frame-by-frame images used for further processing in both the data collection and testing phase.

3.2.3 Mediapipe

Mediapipe is an open-source, cross-platform, machine learning framework developed by google which is specifically used to build pipelines which is then used to perform inference over sensory data (LUGARESI, C. *et al.*, 2019.) Using mediapipe developers can balance usage resources, run operations in parallel and properly synchronize time series data such as audio and video frames. Mediapipe can optimize the detection process by synchronizing the GPU and CPU performance. Using Mediapipe, a perception pipeline can be built as a graph of components which might include media processing functions and trained models used for inference. Sensory data passes through the graph and object detection results and landmark annotations leave the graph.

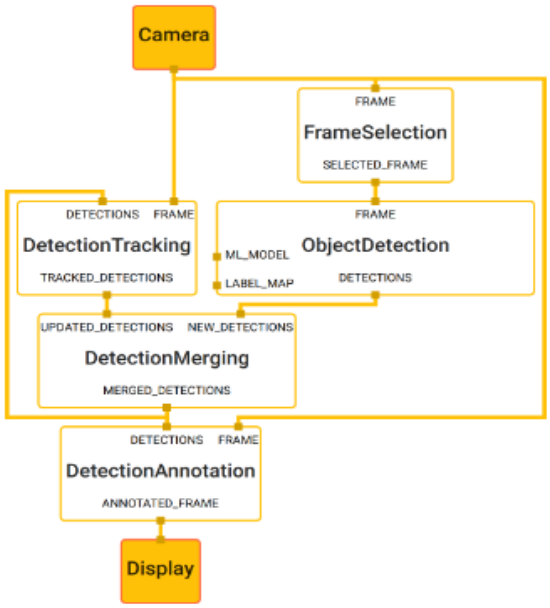


Figure 6: Object detection using mediapipe (LUGARESI, C. et al.*, 2019.)*

Mediapipe was written in C++ which makes it light and fast as C++ is compiled to machine code which does not need an interpreter to run in real time. This also means that

Mediapipe is truly cross platform and can be easily deployed to any platform, Android, Windows or MacOS. Using graphs, subgraphs, calculators, developers can tweak parameters of each calculator to fit different projects which makes it modular and reusable.

Mediapipe performs the 3D tracking of hand and postures using landmarks. The hand landmarks consist of 21 landmarks for each hand while the pose has 33 landmarks. Each hand is connected through a line passing through the shoulders. Using this approach to determine the position of fingers and postures makes mediapipe very effective in building sign language recognition systems. Using the variable detection confidence and tracking confidence parameters within the mediapipe holistic model the model can be tuned to ensure the percentage sureness of the hand detection. Mediapipe is easy to set up and use so the time of developers can be better spent focusing on building the models.

Figure 7: mediapipe key points of hands

The figure above shows the media pipe labelled key points for the hand.

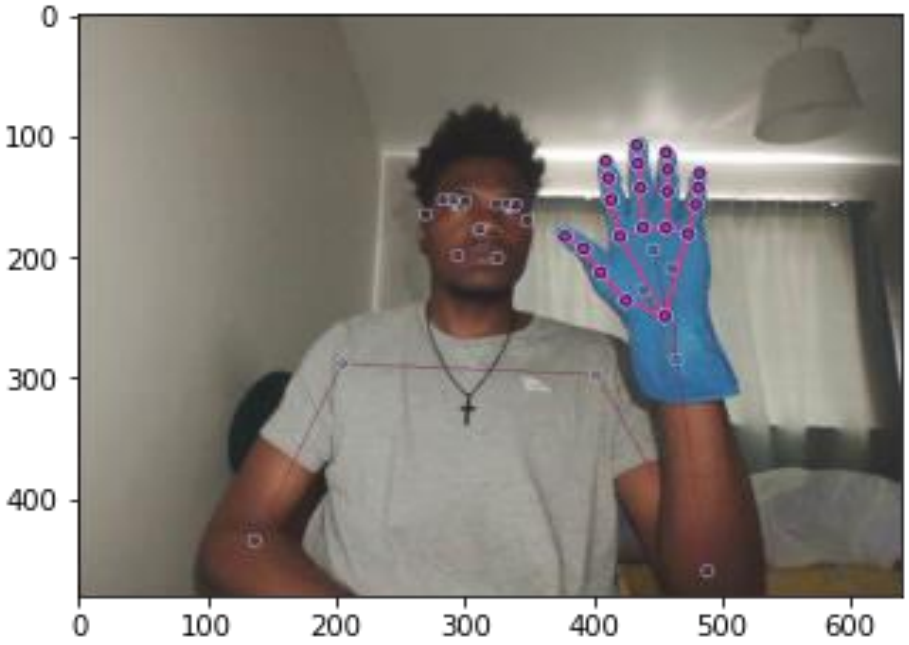


Figure 8: figure showing the mediapipe hand detection and posture keypoints

3.2.4 LSTM

LSTM (Long Short-Term Memory) is a sequential deep neural network model that was built as a replacement to deep recurrent neural network (RNN) models which had a vanishing gradient problem. LSTM was first proposed in 1997 by Sepp Horeiter and Jurgen Schmidhuber and has had iterations over the years. LSTM is very good at predicting time series data and have gotten a lot of attention over the years by researchers. It is used a lot in speech recognition, music composition, handwriting recognition, time series anomaly recognition, stock forecasting and so on. Like most neural network information in LSTM is passed through layer. LSTM uses the concept of gates in its architecture which have different uses.

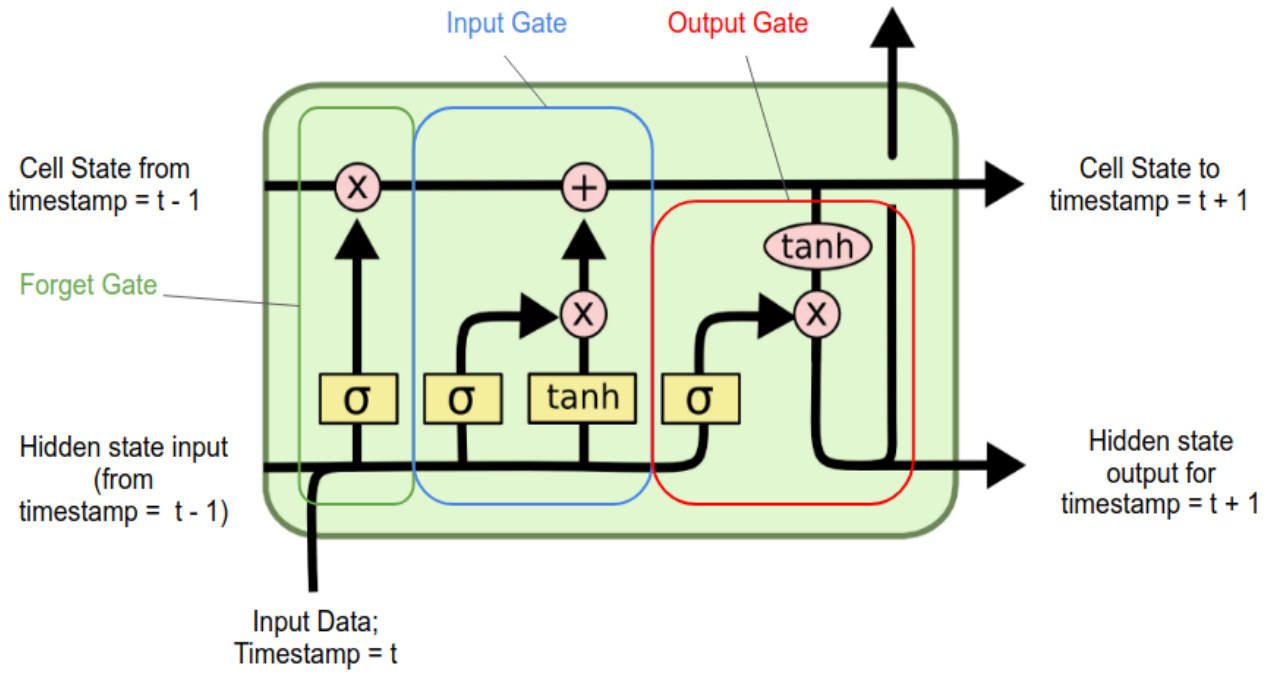


Figure 9: LSTM Cell (T, R., 2020.)

There are 3 gates and their functions are:

Forget gate: This gate within the LSTM cells manages the forgetting of information encoded into the previous state that was passed into the current cell.
Input gate: manages what information from the input in the current time step and the previous hidden state will be encoded into the new cell state.
Output gate: Controls the information from the previous hidden state and input in the current time step that will be encoded into the next hidden state in the next time step.

3.2.5 GRU

The gated recurrent unit (GRU) is a variation of LSTM introduced by Junyoung Chung, Caglar Gulcehre, KyungHyun Cho and Yoshua Bengio in 2014. It was also created to solve the vanishing gradient problem that plagued deep recurrent neural networks, but it does it with less parameters than LSTM in the unit cells. It achieves this feat by using 2 gates instead of 3 gates like in LSTM. GRU is used in every area LSTM is used.

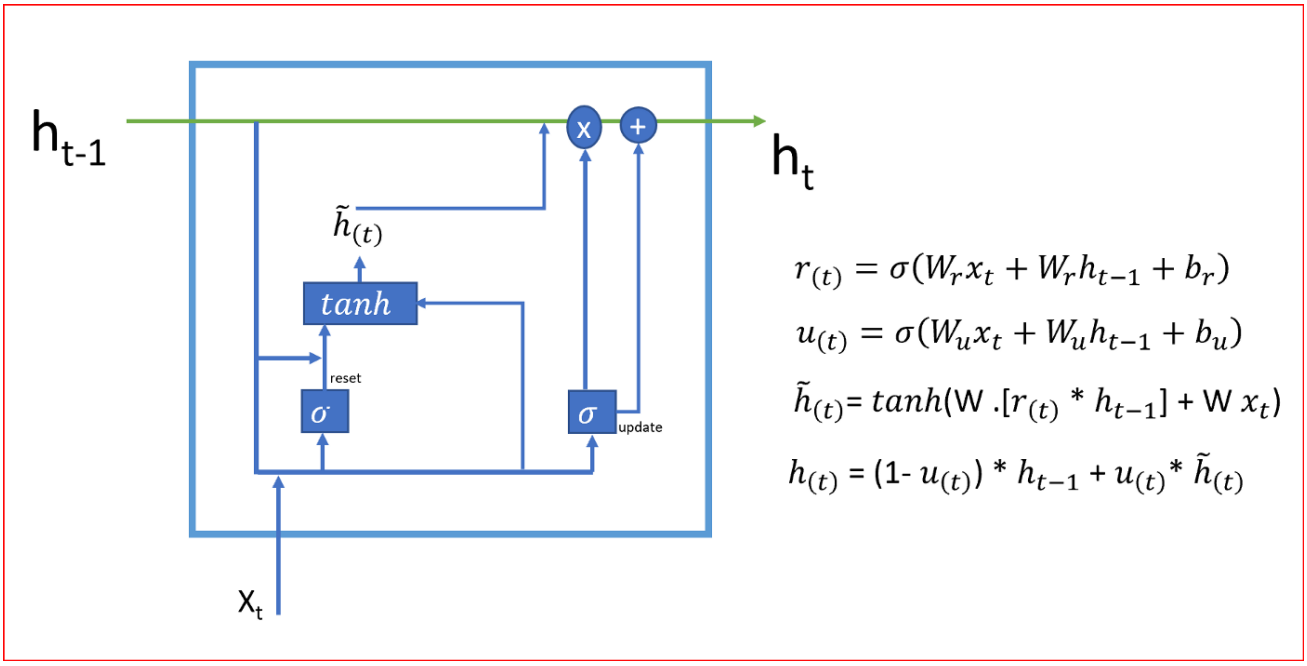


Figure 10: GRU cell (KHANDELWAL, R., 2019.)

There are 2 gates, and their functions are:

Reset Gate: The function of this gate is to determine how much information from previous time steps in the current time step should be forgotten.
Update Gate: the update determines how much information from the previous time steps is to be retained and what information from the input in the current time step will be encoded into the current hidden state that will be passed into the next time step.

3.2.6 Random Forest Classifier

According to IBM random forest is a popular machine learning algorithm trademarked by Adele Cutler and Leo Breiman. It combines the output of multiple decision trees to form a single result. Random forest can be used for both classification and regression problems. It has become popular among researchers because of its flexibility and ease of use. The algorithm used in the random forest which is the random forest algorithm extends the bagging method as it uses both bagging and feature randomness to create trees that are uncorrelated.

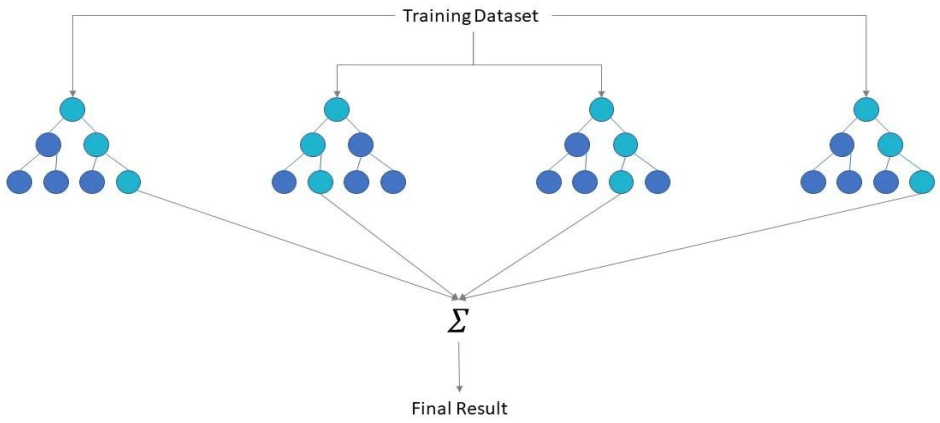


Figure 11: Random Forest Classifier diagram(IBM)

Each tree in the ensemble is made up of data taken from the training sample and 1/3 of the data is set aside to validate the model and it is called out-of-bag data. Random forest has several benefits as well as drawbacks. The randomness in which the training data shuffled prevents overfitting. Random forest is easy to use because it can handle both regression and classification tasks, lastly it can atomically determine features that are important to the prediction. Random forest has a few challenges, one is that it can be time consuming to train and computationally expensive, require more resources to store data and can be complex. Random forest is used in a lot of industries like finance, health care and e-commerce.

3.2.7 Streamlit

According to the Streamlit official website, Streamlit is an open source, light weight, multi-platform dashboarding framework adapted to python and used in building webapps. Streamlit was founded in 2018 by Adrien Treuille and Thiago Teixeria. It is primarily used in creating dashboard and building webapps with python. In terms of support Streamlit supports major libraries like TensorFlow, matplotlib, OpenCV, seaborn, pandas and NumPy to name a few according to the streamlit website. Streamlit is easy to use because it is integrated with python, supports a lot of the major libraries, easy to install and has python friendly syntax. With streamlit the development is fast and easy and source code can be changed and updated while the server is running making rapid development and testing possible, although this is a good thing it can hamper the speed as the entire code needs to be re-run each time the source code is updated especially when computationally intensive routines are within the sources code. Also, streamlit does not have a lot of widgets or design components like other well-established frameworks like Django.

3.2.7 1D convolutional neural network

Convolutional neural networks (CNN) are the neural networks that was built to process data in a matrix or grid form that is why they have become the de-facto neural networks used in computer vision. They are feed forward neural networks with sub sampling layers and convolutional layers that alternates. Three dimensional and two dimensional

convolutional neural networks have been great in dealing with data in 3D and 2D forms such as images and videos frames. The 3D and 2D CNN lose their viability when the data is one dimensional. CNN can achieve very high accuracy, but it requires large data to train as small or medium size databases were insufficient. Also, the training of a deep CNN model can be computationally expensive. 1D CNN has less computational complexities than 2D or 3D CNN (KIRANYAZ, S. *et al.*, 2021) because it requires less parameters to train. This makes the requirement of using dedicated hardware to train deep neural networks less needed. Also, it makes 1D CNN well suited for real-time, low-cost prediction. CNN great because they combine feature extraction and classification which has fuelled their widespread use. 1D CNN is preferred over other forms of CNN when real time prediction is needed, these include areas like speech recognition, real-time electrocardiogram monitoring vibration based structural damage detection, rotating machinery and so forth. All these factors make CNN a suitable model for sign language recognition where images are not passed through the model like this work proposes.

Chapter 4

4.1 Chapter overview

This chapter focus on the explanation of all the processes done during the development process, which includes but not limited to routines developed for the dataset generation, pre-processing, use of OpenCV, GUI and model evaluation. Code snippets are extracted from the main body of code and discussed in non-technical terms as much as possible except when it is necessary to justify the approach.

4.2 Implementation

To reduce a lot of code duplication a class was created to house the frequently used functions and variables. The class was named Braimah while the instance of the Braimah class was named vision_class. The instance of the mediapipe holistic model, mediapipe drawing model, number of sequence frame length, data path for the folders the system is going to operate from, the threshold and alphabets were all housed in the Braimah class as shown in the figure below.

```
class Braimah:
    mp_holistic = mp.solutions.holistic # Holistic model
    mp_drawing = mp.solutions.drawing_utils # Drawing utilities
    no_sequences = 60 # number of videos per action
    frame_length = 30 # length of frames
    DATA_PATH: os.path = os.path.join('data_with_gloves')
    threshold = 0.9
    actions = np.array(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
                        'O','P','Q','R','S','T','U','V','W','X','Y','Z','dot','space'])
    selected_actions = np.array(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L','O','P','Q','T','U',
                                 'W','X','Z','dot'])
```

Figure 12:code snippet showing the housed Braimah class variables

4.2.1 input

The input in the proposed method is the picture frame gotten from the webcam. OpenCV is used to access the webcam and leave the channel open. The webcam used can read up to 60 frames per second but just 30 frames per second was used in this case.

```
cap = cv2.VideoCapture(0)
# Set mediapipe model
with vision_class.mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():
        # Read feed
        _, frame = cap.read()
        # Make detections
        image, results = vision_class.mediapipe_detection(frame, holistic)
        # Draw landmarks
        vision_class.draw_styled_landmarks(image, results)
        # Show to screen
        cv2.imshow('OpenCV Feed', cv2.cvtColor(image,cv2.COLOR_BGR2RGB))
        # Break gracefully
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

Figure 13:OpenCV function to read images from camera

OpenCV is used to start the video capture and read the images frame by frame. OpenCV displays images in the RGB (Red Green Blue) format. Since the resulting frame is a product of mediapipe holistic model landmark detection, which processes images in the BGR (Blue Green Red) format, the conversion back to RGB is necessary for the image is to be displayed properly since RGB is the natural format in which humans see colours.

4.2.2 Detect key points using mediapipe

After the frame is collected from the webcam it undergoes two processes with the latter transforming the frame to include the visual displays of the key points and their connecting lines, before it is showed to the user. The first transformation involves passing the frame through the mediapipe holistic model to detect the key points in the frame.

```
def mediapipe_detection(self,image, model):
    img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    img.flags.writeable = False
    res = model.process(image)                #
    img.flags.writeable = True
    img = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    return img, res
```

Figure 14: mediapipe holistic key point detection code snippet

A function called mediapipe_detection was created within the vision_class to perform the operation. The holistic model and the frame are passed into the function. First a colour conversion is done because mediapipe uses the BGR colour format on the frame passed into the function. Secondly, the writeable property of the frame is set to false to ensure the frame is not altered as it passes through the mediapipe holistic model. After processing the writeable flags of the frame are then set back to True and returned along with the results of the detection.

4.2.3 Drawing of landmarks

A function called draw_styled_landmarks was created to perform the operation of drawing the key points and connecting lines to the frame.

```
def draw_styled_landmarks(self, image, results):
    self.mp_drawing.draw_landmarks(image, results.pose_landmarks, self.mp_holistic.POSE_CONNECTIONS,
                           self.mp_drawing.DrawingSpec(color=(80,22,10), thickness=1, circle_radius=4),
                           Braimah.mp_drawing.DrawingSpec(color=(80,44,121), thickness=1, circle_radius=2)
                           )
    # Draw left-hand connections
    self.mp_drawing.draw_landmarks(image, results.left_hand_landmarks, self.mp_holistic.HAND_CONNECTIONS,
                          self.mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                          self.mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                          )
    # Draw right-hand connections
    self.mp_drawing.draw_landmarks(image, results.right_hand_landmarks, self.mp_holistic.HAND_CONNECTIONS,
                          self.mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                          self.mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                          )
```

Figure 15: code snippet to draw the landmarks

As seen from the figure above the frame and the results from the mediapipe landmark detection function are passed into this function. Since the writeable flag of the frame is already set to true the mediapipe draw_landmarks function can write to the image transforming the image. Only the pose, left and right-hand landmarks are drawn to the image. Within the mediapipe draw_landmarks function the colour, thickness, circle radius can be modified to any preference. The drawing of the landmarks to the frame helps with not only understanding how mediapipe works but also helps in the data collection process because the hands can be positioned properly in real time when signing to make sure the data extracted is correct.

4.2.4 Dataset generation

```
cap = cv2.VideoCapture(0)
# Set mediapipe model
singular_action = ['space']
with vision_class.mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    # Loop through actions
    for action in singular_action:
        # Loop through sequences aka videos
        for sequence in range(vision_class.no_sequences):
            # Loop through video length aka sequence length
            try:
                if sequence == 0:
                    _, first_frame = cap.read()
                    cv2.putText(first_frame, 'Collecting frames for {} sequence Number {}'.format(action, sequence), (15,12)
                    cv2.imshow('OpenCV Feed', first_frame)
                    cv2.waitKey(5000)
                for frame_num in range(vision_class.frame_length):
                    # Read feed
                    _, frame = cap.read()
                    # Make detections
                    image, results = vision_class.mediapipe_detection(frame, holistic)

                    # Draw Landmarks
                    vision_class.draw_styled_landmarks(frame, results)
                    cv2.imshow('OpenCV Feed', frame)

                    # NEW Export keypoints
                    keypoints = vision_class.extract_datapoints(results)
                    Braimah.save_keypoints(vision_class.DATA_PATH,action,sequence,frame_num,keypoints)
                    # Break gracefully
            except RuntimeError as e :
                print(e)

            if cv2.waitKey(10) & 0xFF == ord('q'):
                break

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

Figure 16:code for dataset generation

Several loops were used, the first is for the number of alphabets to be collected, second for the number of sequences which is 60 in this case, last for the number of frames per sequence, each used to control the number of times the data collection process will run for the alphabet, sequence, and number of frames respectively. A collection was also used to restrict the number of alphabets that can be collected at a time to reduce the strain of the data collection process. The Braimah class function extract_datapoints was used to extract the landmarks for pose, left and right hands and concatenate them into a single NumPy array. The class function save_keypoints was implemented to write the extracted NumPy arrays to their corresponding frame number under the current sequence number of the sequence for loop. To make it clear during the data generation phase text showing the alphabet the data is being collected for and the corresponding number in the sequence is displayed using method OpenCV putText() is utilized. Persisting the storage of the dataset has several benefits aside the obvious storage for later use, another important reason is during model evaluation phase after evaluating the False negatives of the alphabets it is easy to regenerate the data of the alphabets with a lot of False negatives without having to regenerate the data for all alphabets

4.2.5 Extract datapoints

This step is a very important step as it is responsible extracting the pose, left and right-hand landmarks and converting them to a NumPy array which is used to trin the model.

```
def extract_datapoints(self, frame_results):
    pose = np.array([[rs.x, rs.y, rs.z, rs.visibility] for rs in frame_results.pose_landmarks.landmark]).flatten()
    if frame_results.pose_landmarks else np.zeros(132)
    lh = np.array([[rs.x, rs.y, rs.z] for rs in frame_results.left_hand_landmarks.landmark]).flatten()
    if frame_results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[rs.x, rs.y, rs.z] for rs in frame_results.right_hand_landmarks.landmark]).flatten()
    if frame_results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([pose, lh, rh])
```

Figure 17: code snippet for mediapipe key point extraction

The result from the mediapipe detection model is passed into this function with the variable name frame_results. The pose landmarks unlike the left and right-hand landmarks have 4 variables x, y, z and visibility property. The mediapipe pose landmark has 33 key points, the multiplication of these 4 variables with these key points makes 132 as seen in the figure above. The flatten property is used to convert the 2D array to 1D array. The if else is used to check if the landmark is captured if not zeros are generated in the place of the data. This is necessary during training because whatever key points captured by the mediapipe holistic model can be inputted in the spaces the zeros occupy. The same process is done for the landmarks of both hands. In total each hand landmark has 21 key points as earlier stated and those 21 key points are multiplied by the x, y, z variables of each key point to make 63 points which is then multiplied by 2 to give 126. After concatenation of all landmarks into a single array it totals at 258. Each frame will have a NumPy array with 258 values. The resulting NumPy array is then returned.

4.2.6 Create folders

To store the data generated from the video frames during the data collection process a file system needed to be created. A folder was created for each Alphabet and each alphabet had several folders according to the number of sequences. Each sequence has 30 NumPy files named 0 – 29 with each file representing the 30 frames captured for that sequence

```
def create_folders(self):
    for action in self.actions:
        for sequence in range(self.no_sequences):
            try:
                os.makedirs(os.path.join(self.DATA_PATH, action, str(sequence)))
            except:
                pass
```

Figure 18: code snippet for creating folders

The figure above shows how the folders were created. The python os library in combination with for loops were used in the creation of the folders.

4.1.7 save key points

The function named save_keypoints was created to save the extracted NumPy files from their corresponding video frames to the dataset directories. Due to the frequency of use the @staticmethod keyword was used to mark the function which makes the function class independent and saves it to the Random Access Memory (RAM) rather than the heap memory of the computer for faster access.

```
@staticmethod
def save_keypoints(data_path,action,sequence_no,frame_num,keypoints):
     npy_path = os.path.join(data_path, action, str(sequence_no), str(frame_num))
     np.save(npy_path, keypoints)
```

Figure 19:code snippet for the save_keypoints function

The data path, alphabet, sequence number, frame number and the NumPy array developed from the extract_keypoints were passed into the function. Again the python library OS with its path.join function was used to generate a path that generated path is then passed to the NumPy.save function together with the key points array, the function writes the key points to the directory path.

4.2.8 Data pre-processing

After the data for all the alphabets had been generated and saved to their corresponding files in the dataset directories, they needed to be loaded back into memory and arranged

in a structured format before they could be used for training. The figures below show the routines created to perform this task.

```
label_map = {label:num for num, label in enumerate(vision_class.selected_actions)}
```

Figure 20: code snippet for alphabet dictionary creation

The figure above shows the creation of a dictionary named label_map which uses the selected alphabets as keys and their respective positions in the selected_actions list as values.

```
sequences, labels = [], []
for action in vision_class.selected_actions:
    for sequence in range(vision_class.no_sequences):
        window = []
        for frame_num in range(vision_class.frame_length):
            res = np.load(os.path.join(vision_class.DATA_PATH, action, str(sequence),
                                       "{}.npy".format(frame_num)))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])
```

Figure 21: code snippet or loading files

The sequences and labels list created was used to store the data read from the dataset directories. 3 for loops were implemented, the first to loop through the list of the selected alphabets, the second for looping through the length of the sequences while the last from frame length. The same process used for storing was also used reading back into memory. The NumPy library with the NumPy.Load function was used to load the files corresponding to the 30 frames for the 60 sequences of their respective alphabets. The result of the loaded NumPy file is assigned to variable res and then appended to the list labelled window which is then appended the list variable called sequences. At the end of the second for loop the integer value of the corresponding action from the label dictionary is also appended to a separate list named labels which is used to create the y training data.

The next step in the data pre-processing was to convert the list named sequences which contains all the data into a NumPy array as shown in the figure below. The shape of the new array was also checked.

```
X = np.array(sequences)
X.shape
```

```
(1260, 30, 258)
```

Figure 22: code snippet for X values creation

The X data had 1260 records which is 60 sequences by 20 alphabets plus the “.” Label making 21 labels. Each record has 30 arrays and each of those 30 has 258 numbers.

To create the y data the list named labels was used to generate an array of categorical data of ones and zeros. This is necessary not to create bias during the training process as integers can be interpreted as having an ordered significance. In this case it also suitable because the predictions returned by the neural networks are probability distributions enabled by the SoftMax activation function in the last dense layer of the neural network's configuration.

```
y = to_categorical(labels).astype(int)
y
```

```
array([[1, 0, 0, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 1],
       [0, 0, 0, ..., 0, 0, 1],
       [0, 0, 0, ..., 0, 0, 1]])
```

Figure 23: code snippet for Y data creation

This was done with the help of the to_categorical functions from the keras.utils library.

The next step was to split the data into training and test data. This was implemented using the train_test_split function from the sklearn.model_selction library.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10)
```

Figure 24: code snippet for dataset splitting

The dataset was split with 90% for training and 10% for test data this is done because another 10% of the training data was also used for validation accuracy during training making a total of 19% of the data used for testing.

4.2.9 model building and training

Several models were trained using the generated data to see which one has the best performance in terms of accuracy in real time. These models included LSTM, GRU, 1DConvo from the Keras library and random forest classifier from the scikit-learn library

4.2.10 GRU

```
model = Sequential()
model.add(GRU(200, return_sequences=True, activation='tanh', input_shape=(30,datapoints),dropout=0.2))
model.add(GRU(400, return_sequences=True, activation='tanh',dropout=0.2))
model.add(GRU(200, return_sequences=False, activation='tanh',dropout=0.2))
#model.add(GRU(1000, return_sequences=True, activation='tanh',dropout=0.1))
#model.add(GRU(400, return_sequences=False, activation='tanh',dropout=0.1))
model.add(Dense(125, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(self.selected_actions.shape[0], activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
return model
```

Figure 25: GRU Model instantiation

After several iterations of the layers the current configuration in the figure above was used which struck a balance between model complexity and speed. This was needed to make the GRU capable of coping with real time prediction and to make sure the neural network does not become too deep because the activation function used in the LSTM outputs a value between 0 and 1 which means the model can stop learning if too deep which can lead to vanishing gradient. The figure above also shows the format in which the GRU model was built. The sequential model had 3 GRU model layers and 3 dense layers. The first and third layers had 200 units while the second layer had 400. The preceding layers are made up of dense layers with the first having 125 units, the second having 64 units while the last dense layer has a number according to the number of categories to be classified. The activation functions used for the GRU layers was tanh because tanh can take advantage of the presence of the Nvidia GPU CUDA cores present in the system used for training which reduces the time needed for training by over 80-90 percent. The dense layers except the third one uses a Rectified linear unit (ReLU) activation function which outputs a value between -1 and 1. It is less computationally intensive than tanh and all the drawbacks of the vanishing gradient problem in deep neural network is completely removed. The last layer uses a Softmax activation function which converts a vector of numbers into a probability distribution which means the certainty in which a model classifies a label can be checked in real time. During training in deep learning the network parameters like weights and learning rate needs to be modified to minimize the loss function therefore an optimization function is needed to make this happen. The adam optimizer is used because it requires less computational time and less parameter to tune, it also gives a good probability of getting the best results. The loss function categorical cross entropy is used because of the multiple classes present in the data. The metric used to evaluate the model is categorical accuracy. To make sure the model did not overfit a 20% dropout was introduced in each GRU layer and a 10% dropout was introduced in the dense layers. This randomly freezes the specified percentage of nodes in the layers.

4.2.11 LSTM

For the LSTM model when set with the same configuration with the GRU model failed to learn during training so a less shallow configuration was used but it still gave a good performance.

```
if model_type == 'LSTM':
    model = Sequential()
    model.add(LSTM(150, return_sequences=True, activation='tanh', dropout= 0.1, input_shape=(30,datapoints)))
    #model.add(LSTM(250, return_sequences=True, activation='tanh', dropout=0.1))
    #model.add(LSTM(250, return_sequences=True, activation='tanh',dropout=0.1))
    model.add(LSTM(150, return_sequences=False, activation='tanh',dropout=0.1))
    model.add(Dense(100, activation='relu'))
    model.add(Dropout(0.1))
    model.add(Dense(62, activation='relu'))
    model.add(Dropout(0.1))
    model.add(Dense(self.selected_actions.shape[0], activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
    return model
```

The first 2 layers of the sequential model are LSTM layers with 150 units each. A dropout rate of 10% was used because of the small number of units used in the layers. The next 3 layers are dense layers with the first dense layer having 100 units, the second with 62 units while the last has the same number of units as the labels to be classified. All other configurations for the dense layer are the same with the configuration in the GRU model.

The LSTM had lower training parameters than the GRU because of the lower number units per layer

4.2.12 CONV1D

```
elif model_type == 'CONV1D':
    model = Sequential()
    model.add(Conv1D(filters=300, kernel_size=3, activation='relu', input_shape=(30,258)))
    model.add(Conv1D(filters=600, kernel_size=3, activation='relu'))
    model.add(Dropout(0.2))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(self.selected_actions.shape[0], activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

Figure 26: code snippet for CONV1D build

The 1D CNN when built with similar configuration to the LSTM and GRU performed very poorly in the real time test. 2 layers of the 1D CNN followed by a dropout layer is used. The 2 layers are then flattened after a pool_size of 2 is added. The pool layer added is responsible for down sampling the features in the data because the model is very sensitive to the positions of the features.

4.2.13 Graphical user interface

The graphical user interface is a simple interface built using the streamlit frame it consists

```
mp_holistic = mp.solutions.holistic #Holistic model
mp_drawing = mp.solutions.drawing_utils
threshold = 0.9

selected_actions = np.array(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L','M','N','O','P','Q','R',
                             'S','T','U','V','W','X','Y','Z','dot','space'])
st.set_page_config(
    page_title='BSL Detection App',
    layout='wide'
)

model_lstm = keras.models.load_model('data_gloves_LSTM_pose.h5')
model_lstm_nopose = keras.models.load_model('data_gloves_LSTM_nopose.h5')
model_gru = keras.models.load_model('data_gloves_GRU_pose.h5')
model_gru_nopose = keras.models.load_model('data_gloves_GRU_nopose.h5')
model_conv1d = keras.models.load_model('data_gloves_CONV1D.h5')
model_conv1d_nopose = keras.models.load_model('data_gloves_CONV1D.h5')
```

Figure 27: code snippets for loading models into the GUI environment

The mediapipe holistic model is loaded into memory and the threshold for prediction is set. The specified actions to be predicted are also created. The page configuration for streamlit is instantiated with its title and layout. All models for both pose and non-pose are loaded into memory using the Keras library.

4.2.13.1 Description

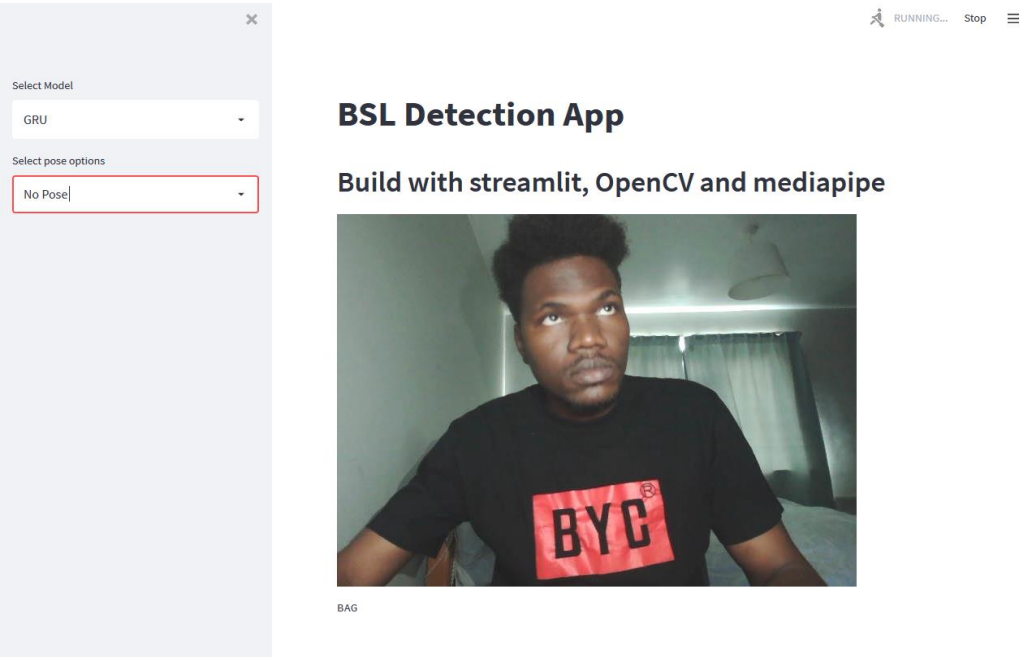


Figure 28: Gui description

1. The side bar has 2 options select model and select pose positions so the user can pick whatever combinations of the 2 configurations to use.
2. The area with the picture shows the frame in real time and it is the region in which all BSL signs will be performed.
3. The area just below the image is where the recognized signs will be outputted and where sentence formation will occur

4.2.13.2 Detection

A different approach was taken with the detection using the GUI because it had to accommodate multiple models for the real time pose and non-pose BSL recognition detection.

```
def detect(sequence, sentence, choice, pose):

    if ['LSTM','GRU','CONV1D'].__contains__(choice):
        if choice == 'LSTM' and pose == 'Pose':
            prediction = model_lstm.predict(np.expand_dims(sequence[:30], axis=0))[0]
        elif choice == 'LSTM' and pose == 'No Pose':
            prediction = model_lstm_nopose.predict(np.expand_dims(sequence[:30], axis=0))[0]
        elif choice == 'GRU' and pose == 'Pose':
            prediction = model_gru.predict(np.expand_dims(sequence[:30], axis=0))[0]
        elif choice == 'GRU' and pose == 'No Pose':
            prediction = model_gru_nopose.predict(np.expand_dims(sequence[:30], axis=0))[0]
        elif choice == 'CONV1D' and pose == 'Pose':
            prediction = model_conv1d.predict(np.expand_dims(sequence[:30], axis=0))[0]
        elif choice == 'CONV1D' and pose == ' No Pose':
            prediction = model_conv1d_nopose.predict(np.expand_dims(sequence[:30], axis=0))[0]
        else:
            pass
        if prediction[np.argmax(prediction)] >= threshold:
            if selected_actions[np.argmax(prediction)] == 'dot':
                sentence.append('.')
                # st.write(''.join(sentence))
            elif selected_actions[np.argmax(prediction)] == 'space':
                sentence.append(' ')
            else:
                sentence.append(selected_actions[np.argmax(prediction)])
                # st.write(''.join(sentence))
        else:
            pass
    else:
        pass
```

Figure 29: detection logic for pose and non-pose

If and else approach is used to determine the right model to use and if the pose or non-pose method had been selected by the user. The prediction is then checked to see if it

reaches the specified threshold if it does it is appended to the sentence list passed into the function. If the prediction is “dot” or “space” the characters “.” and empty space are appended to the sentence list respectively.

4.2.13.3 Program flow

The figure below shows the how the program flow takes place and how the components within the GUI were initialized and used.

```
with mp_holistic.Holistic(min_detection_confidence=0.6, min_tracking_confidence=0.6) as holistic:
    while cam.isOpened():
        _, frame = cam.read()
        image, result = mediapipe_detection(frame, holistic)
        if result.right_hand_landmarks is None and result.left_hand_landmarks is None:
            frame_sequence.clear()
            count = 0
            draw_styled_landmarks(image, result, pose_choice)
            frame_window.image(image)
            if cv2.waitKey(10) & 0xFF == ord('q'):
                break
            continue
        else:
            draw_styled_landmarks(image, result, pose_choice)
            key_points = extract_datapoints(result, pose_choice)
            frame_sequence.append(key_points)
            frame_window.image(image)
            count += 1
            if cv2.waitKey(10) & 0xFF == ord('q'):
                break
        if count == 30:
            detect(frame_sequence, sentence, choices, pose_choice)
            text_area.text('')
            text = ''.join(sentence)
            text_area.subheader(text)
            if len(sentence) == 0:
                pass
            else:
                if sentence[len(sentence) - 1] == '.':
                    sentence.pop()
                    audio.speak(''.join(sentence))
                    sentence.clear()
                    text_area.text('')
                else:
                    pass

            count = 0
```

Figure 30: code snippet for program flow

In the main function of the python script hosing the GUI code OpenCV is used to start the capturing of the video and each frame is passed into mediapipe holistic model so landmark detection can be made. The result is then used to check if hands are in the frame. If no hands are in the frame the list used to house the frames that will used for prediction is cleared and the count variable returned to zero. There are two benefits to this, one is the data passed into the model for prediction will be a sign that has been formed by the user and not something random. Secondly, it helps with the jittering because if a sign jitters to the point where no hand is detected it starts the data collection process all over again which ensures the right data has been passed to the model increasing the chances of having the right prediction. After the detection process is done, the frame is then used to update the image property of the Streamlit image class that was instantiated. If there is no heavy interruption to the point where the hands aren't detected in the signing process the frames are constantly processed and the count property updated until the number reaches 30 where it will be passed into the detection function. The detection function returns the prediction and if the prediction is the full stop character. The Sentence collection is popped, and the list is then converted to a string and passed to the instance of the pyttsx3 library used for text to speech translation. The sentence list is then cleared along with the output text so the process can start all over again. Since the program was built in a way where the detection can use both pose or just the right- and left-hand landmarks the pose_choice property is passed to all the functions that are used in the process.

4.2.14 Model evaluation

The LSTM and GRU models can both be used for classification and regression but in this case, they were used for classification therefore they were evaluated using classification evaluation methods. A simple but effective approach is using the classification report which includes accuracy, precision, recall, F1 score and support.

```
print(classification_report(ytrue,y_predict))
```

Figure 31: code snippet for classification report

The function multiple_confusion_matrix was used to display the confusion matrix of each label.

```
cf = multilabel_confusion_matrix(ytrue, y_predict)
cf
```

Figure 32: code snippet for multiple confusion matrix

The accuracy was calculated using the accuracy_score library from the sklearn.metrics library.

```
accuracy_score(ytrue, y_predict)
```

```
0.9761904761904762
```

Figure 33: code snippet for accuracy score

Having the confusion matrix output in the format shown in the figure below might seem hard to read so the seaborn library was used to plot the multiple confusion matrix for the individual classes.

```
def print_confusion_matrix(confusion_matrix, axes, class_label, class_names, fontsize=14):

    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
    )

    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d", cbar=False, ax=axes)
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right', fontsize=fontsize)
    axes.set_ylabel('True label')
    axes.set_xlabel('Predicted label')
    axes.set_title("class - " + class_label)
```

Figure 34:code snippet to print multiple confusion matrix

The function above was used to generate the heatmap using the seaborn library after the multiple confusion matrix was converted to a Pandas data frame with the alphabets as index and class names. The title with y and x labels were then set accordingly.

Chapter 5

5.1 Chapter overview

This chapter has 3 tables showing results. The first table shows the models and their respective accuracy on both test data and in real time testing using pose and hand landmarks for prediction. The Second Table like first table shows the accuracy data but without the use of pose landmarks. The third table shows the optimal result from this work in comparison with previous works using mediapipe indicating if real testing was done or not. The tables are followed by training visualizations and classification reports screen shots for the 6 models with each model in their own subheading. The confusion matrix plots was not added to this section but rather to the section B of the appendix section because of their very large sizes.

5.2 Result

Two methods were used to develop the models so there are total of 6 models. 3 using pose estimation and 3 without. The results are grouped accordingly and a table comparing result of this work to other related works using mediapipe is also done.

Model	Accuracy %	Real time Accuracy %
LSTM	97.6	67.8
GRU	99.3	82.1
CONV1D	98.2	67.8

Figure 35: Result using pose estimation

The table above shows results for the models when pose landmarks in combination with the left- and right-hand landmarks were used for training. LSTM having an accuracy on 97.6% on test data and 67.8% in real time testing. GRU has 99.3% on test data and 82.1% in real time testing.

Model	Accuracy %	Real time Accuracy %
LSTM	98.8	82.1
GRU	99.4	85.7
CONV1D	97.2	75

Figure 36: Result without pose estimation

The table above shows the results reached by the model when just left- and right-hand landmarks were used for training. LSTM with had an accuracy of 98.8% on the test data and 82.1% in real time testing. GRU reached an accuracy of 99.4 on the test data and a real time accuracy of 85.7% While the CONV1D had a 97.2% accuracy on test data and a real time of 78.5%.

Study	Real time testing	language	Accuracy %
D. BISHT *et al.*, 2022	yes	ASL and few ISL words	94.58(real time)
V. H. IYER *et al.*, 2022.	yes	N/A	87.5

M. H. ISMAIL, S. A. DAWWD and F. H. ALI, 2021.	no	ARSL	99.03
A. CHAIKAEW, K. SOMKUAN and T. YUYEN, 2021.	no	Thai sign language	97
S. ADHIKARY, A. K. TALUKDAR	yes	ISL	97.4
This work	yes	BSL	92.6(real time)

Figure 37: Results of comparative studies

The table above itemizes the performance of the different models used by different researchers using the media pipe holistic model to extract the data used for classification.

5.3 Results for the Model Evaluation

5.3.1 GRU

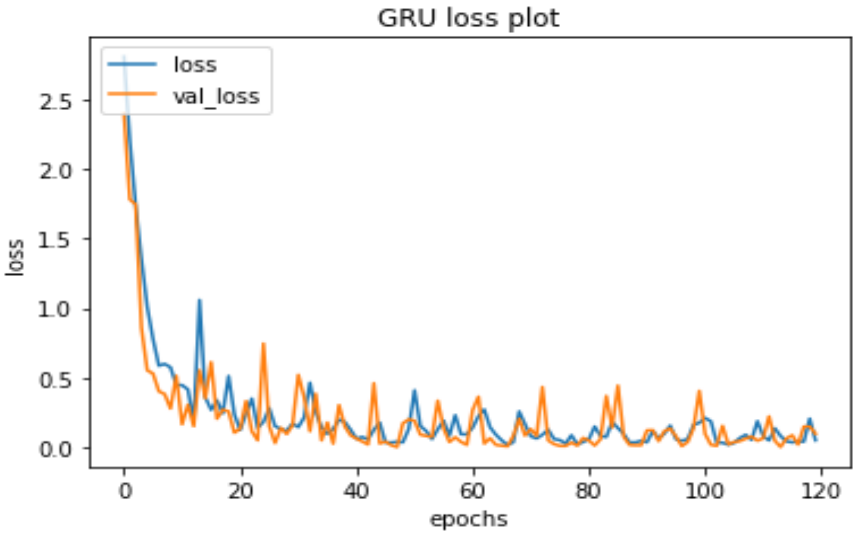


Figure 38: GRU pose loss plot

The figure above shows the GRU loss versus validation loss plot for the pose in combination with the hand landmark training.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6
1	1.00	1.00	1.00	6
2	1.00	1.00	1.00	15
3	1.00	1.00	1.00	8
4	1.00	1.00	1.00	5
5	1.00	1.00	1.00	5
6	0.80	1.00	0.89	4
7	1.00	1.00	1.00	5
8	1.00	1.00	1.00	6
9	1.00	1.00	1.00	7
10	1.00	1.00	1.00	4
11	1.00	1.00	1.00	1
12	1.00	0.88	0.93	8
13	1.00	1.00	1.00	7
14	1.00	1.00	1.00	3
15	1.00	1.00	1.00	8
16	1.00	1.00	1.00	3
17	1.00	1.00	1.00	3
18	1.00	1.00	1.00	6
19	1.00	1.00	1.00	6
20	1.00	1.00	1.00	8
21	1.00	1.00	1.00	7
22	1.00	1.00	1.00	6
23	1.00	1.00	1.00	8
24	1.00	1.00	1.00	8
25	1.00	1.00	1.00	6
26	1.00	1.00	1.00	3
27	1.00	1.00	1.00	6
accuracy			0.99	168
macro avg	0.99	1.00	0.99	168
weighted avg	1.00	0.99	0.99	168

Figure 39: GRU pose classification report

The figure above shows the classification report for the GRU model using both pose and hand landmarks for detection

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6
1	1.00	1.00	1.00	3
2	1.00	1.00	1.00	9
3	1.00	1.00	1.00	4
4	1.00	1.00	1.00	2
5	1.00	1.00	1.00	8
6	1.00	1.00	1.00	8
7	1.00	1.00	1.00	9
8	1.00	1.00	1.00	8
9	1.00	1.00	1.00	5
10	1.00	1.00	1.00	5
11	1.00	0.80	0.89	5
12	0.86	1.00	0.92	6
13	1.00	1.00	1.00	7
14	1.00	1.00	1.00	13
15	1.00	1.00	1.00	3
16	1.00	1.00	1.00	5
17	1.00	1.00	1.00	3
18	1.00	1.00	1.00	2
19	1.00	1.00	1.00	1
20	1.00	1.00	1.00	12
21	1.00	1.00	1.00	7
22	1.00	1.00	1.00	4
23	1.00	1.00	1.00	7
24	1.00	1.00	1.00	7
25	1.00	1.00	1.00	7
26	1.00	1.00	1.00	8
27	1.00	1.00	1.00	4
accuracy			0.99	168
macro avg	0.99	0.99	0.99	168
weighted avg	0.99	0.99	0.99	168

Figure 40: GRU no pose classification report

The figure above shows the classification report for the GRU model using just hand landmarks for the detection process.

5.3.2 LSTM

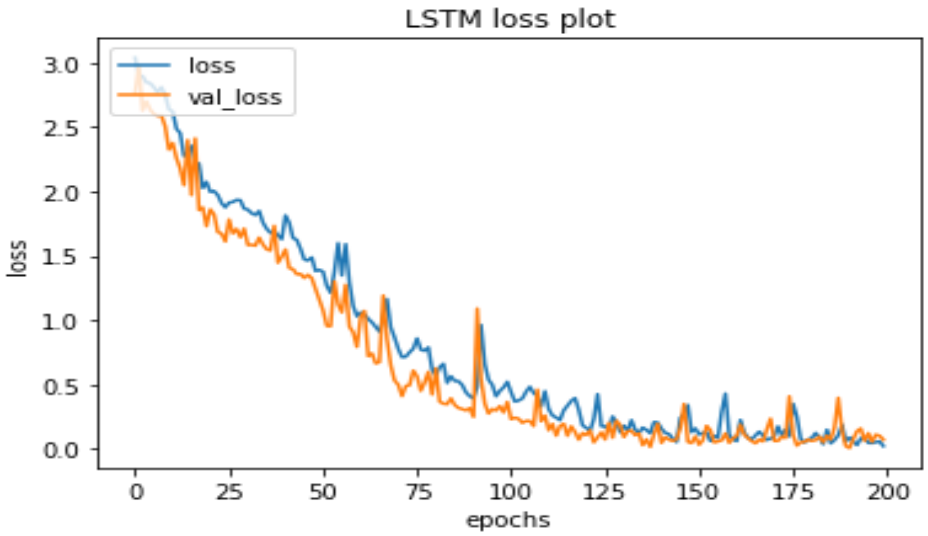


Figure 41: LSTM pose loss plot

The figure above shows the loss versus validation loss training graph for the LSTM model using both pose and hand landmarks for detection.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	5
1	1.00	1.00	1.00	5
2	1.00	1.00	1.00	8
3	1.00	0.83	0.91	6
4	1.00	1.00	1.00	5
5	0.89	0.89	0.89	9
6	1.00	1.00	1.00	6
7	1.00	0.80	0.89	5
8	1.00	1.00	1.00	6
9	1.00	1.00	1.00	4
10	1.00	1.00	1.00	2
11	1.00	1.00	1.00	4
12	1.00	1.00	1.00	7
13	1.00	1.00	1.00	4
14	0.75	1.00	0.86	6
15	1.00	1.00	1.00	6
16	0.67	1.00	0.80	2
17	1.00	1.00	1.00	5
18	1.00	1.00	1.00	8
19	0.83	1.00	0.91	5
20	1.00	0.88	0.93	8
21	1.00	1.00	1.00	6
22	1.00	1.00	1.00	6
23	1.00	0.90	0.95	10
24	1.00	1.00	1.00	9
25	1.00	1.00	1.00	8
26	1.00	1.00	1.00	9
27	1.00	1.00	1.00	4
accuracy			0.97	168
macro avg	0.97	0.97	0.97	168
weighted avg	0.98	0.97	0.97	168

Figure 42:LSTM pose classification report

The figure above shows the classification report for the LSTM model using both pose and hand landmarks for detection.

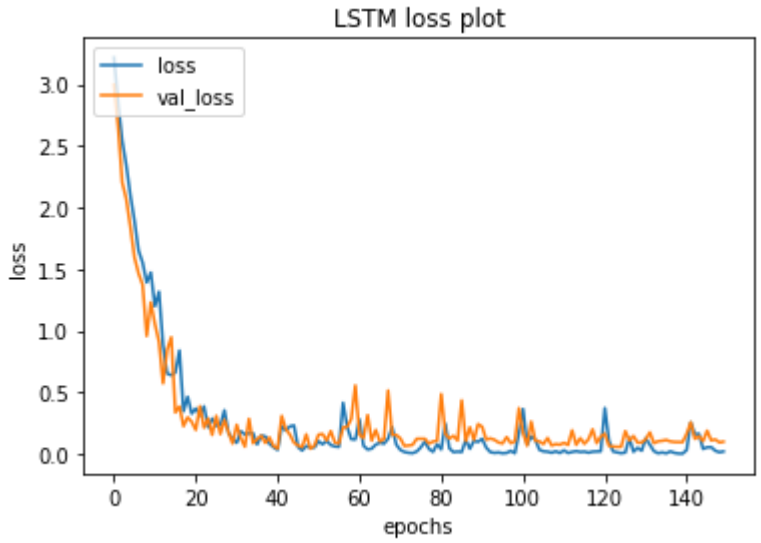


Figure 43: LSTM loss plot no pose

The figure above shows the loss versus validation loss training graph for the LSTM model using just hand landmarks for detection.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6
1	1.00	1.00	1.00	3
2	1.00	1.00	1.00	9
3	1.00	1.00	1.00	4
4	1.00	1.00	1.00	2
5	1.00	1.00	1.00	8
6	1.00	1.00	1.00	8
7	1.00	1.00	1.00	9
8	1.00	1.00	1.00	8
9	1.00	1.00	1.00	5
10	1.00	1.00	1.00	5
11	1.00	0.80	0.89	5
12	0.86	1.00	0.92	6
13	1.00	1.00	1.00	7
14	1.00	1.00	1.00	13
15	1.00	1.00	1.00	3
16	1.00	1.00	1.00	5
17	1.00	1.00	1.00	3
18	1.00	1.00	1.00	2
19	1.00	1.00	1.00	1
20	1.00	1.00	1.00	12
21	1.00	1.00	1.00	7
22	1.00	1.00	1.00	4
23	1.00	1.00	1.00	7
24	1.00	1.00	1.00	7
25	1.00	1.00	1.00	7
26	1.00	1.00	1.00	8
27	1.00	1.00	1.00	4
accuracy			0.99	168
macro avg	0.99	0.99	0.99	168
weighted avg	0.99	0.99	0.99	168

Figure 44: LSTM no pose classification report

The figure above shows the classification report for the LSTM model using just hand landmarks for detection.

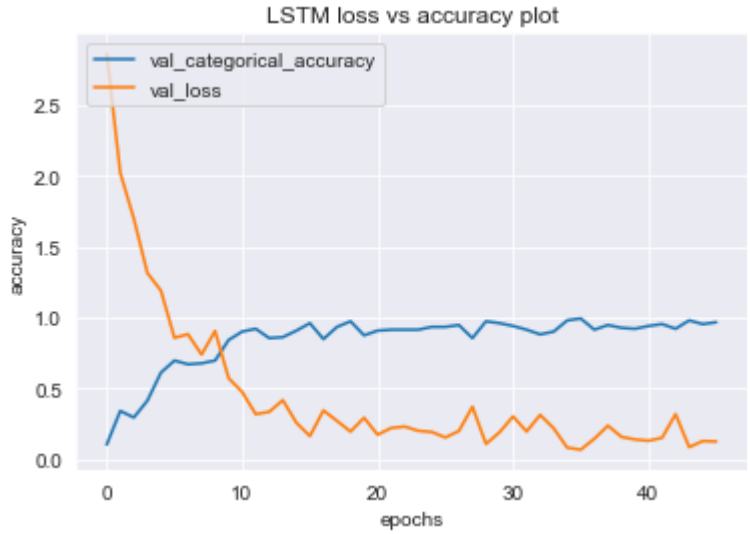


Figure 45: LSTM validation loss vs validation accuracy graph

The figure above shows the validation loss versus validation accuracy for the LSTM model using just hand landmarks for detection.

5.3.3 CONV1D

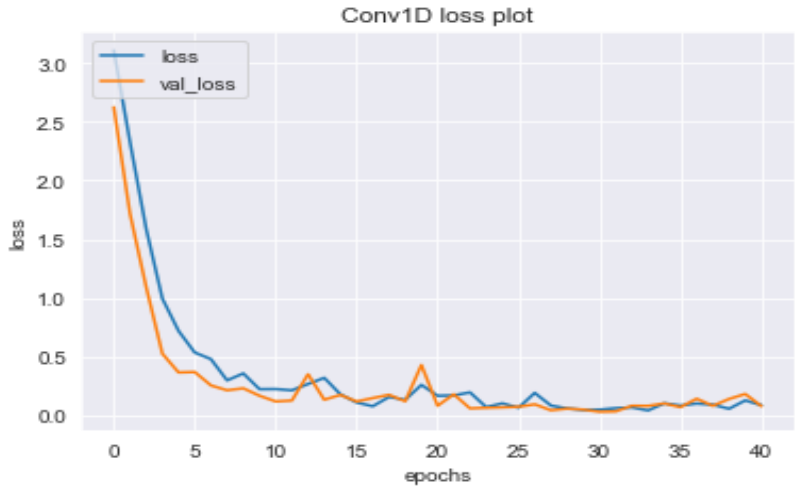


Figure 46: CONV1D loss plot no pose

The training plot of CONV1D with using just the hand landmarks for detection

	precision	recall	f1-score	support
0	1.00	1.00	1.00	8
1	1.00	1.00	1.00	7
2	0.88	1.00	0.93	7
3	1.00	1.00	1.00	4
4	1.00	0.90	0.95	10
5	1.00	1.00	1.00	4
6	1.00	1.00	1.00	8
7	1.00	1.00	1.00	11
8	1.00	1.00	1.00	6
9	1.00	1.00	1.00	5
10	1.00	1.00	1.00	4
11	1.00	1.00	1.00	4
12	1.00	0.89	0.94	9
13	1.00	0.88	0.93	8
14	1.00	1.00	1.00	6
15	1.00	1.00	1.00	8
16	0.67	0.67	0.67	3
17	1.00	1.00	1.00	9
18	1.00	1.00	1.00	6
19	1.00	1.00	1.00	3
20	1.00	1.00	1.00	5
21	0.83	1.00	0.91	5
22	1.00	1.00	1.00	3
23	1.00	1.00	1.00	9
24	0.80	1.00	0.89	4
25	1.00	1.00	1.00	9
26	1.00	1.00	1.00	2
27	1.00	1.00	1.00	1
accuracy			0.98	168
macro avg	0.97	0.98	0.97	168
weighted avg	0.98	0.98	0.98	168

Figure 47: classification report for CONV1D using the hand landmarks

The figure above shows the classification report for the CONV1D model using just the hand landmarks for detection.

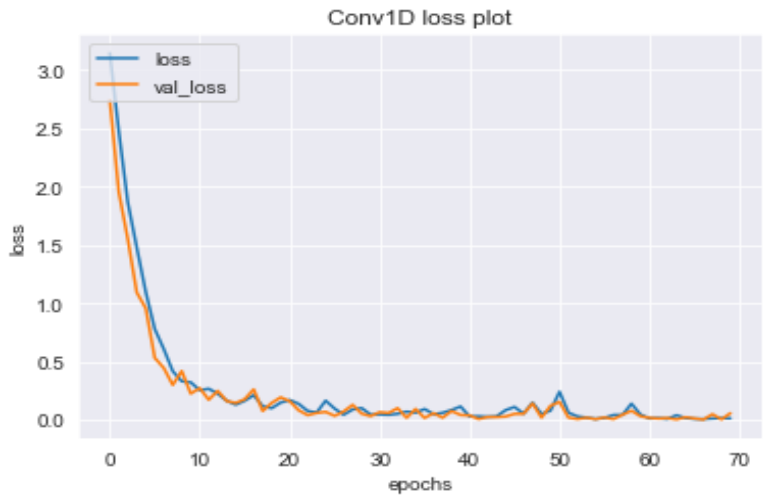


Figure 48: Training plot for CONV1D using pose

The figure above shows the training plot for the CONV1D using pose and hand landmarks for detection.

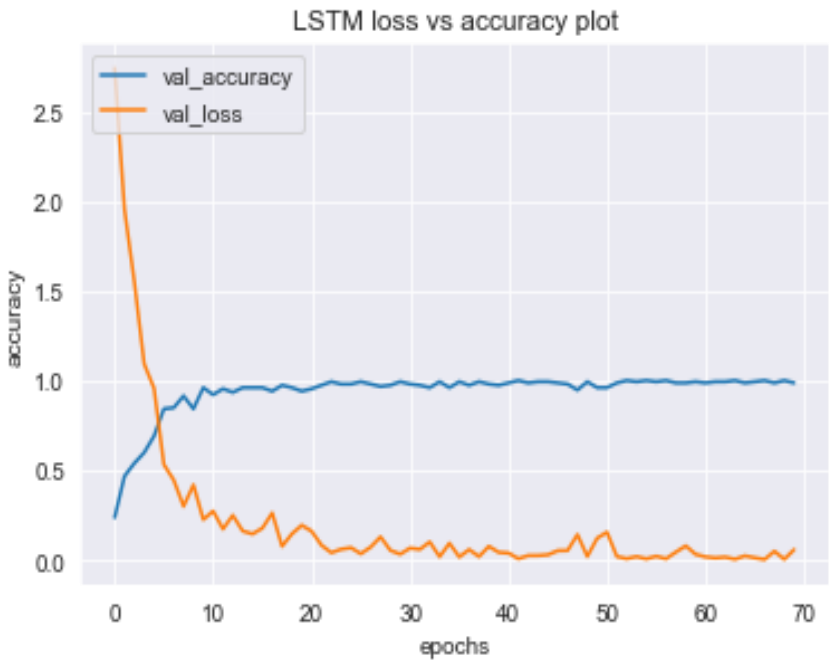


Figure 49: validation loss versus validation accuracy for CONV1D

The figure above shows the plot for validation loss versus validation accuracy for CONV1D using pose and hand landmarks for detection.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	7
1	0.75	1.00	0.86	6
2	1.00	1.00	1.00	3
3	1.00	1.00	1.00	5
4	1.00	0.71	0.83	7
5	1.00	1.00	1.00	6
6	1.00	1.00	1.00	3
7	1.00	1.00	1.00	9
8	1.00	1.00	1.00	6
9	1.00	1.00	1.00	6
10	1.00	1.00	1.00	5
11	1.00	1.00	1.00	4
12	1.00	1.00	1.00	8
13	1.00	1.00	1.00	5
14	1.00	1.00	1.00	9
15	1.00	1.00	1.00	6
16	1.00	1.00	1.00	4
17	1.00	1.00	1.00	1
18	1.00	1.00	1.00	6
19	1.00	1.00	1.00	10
20	1.00	1.00	1.00	4
21	1.00	1.00	1.00	13
22	1.00	1.00	1.00	7
23	1.00	1.00	1.00	4
24	1.00	1.00	1.00	5
25	0.89	1.00	0.94	8
26	1.00	1.00	1.00	5
27	1.00	0.83	0.91	6
accuracy			0.98	168
macro avg	0.99	0.98	0.98	168
weighted avg	0.99	0.98	0.98	168

Figure 50:classification report for CONV1D using pose and hand landmarks for prediction

Table 1: Real time label recognition Table

Labels	GRU + Pose	GRU + No Pose	LSTM + Pose	LSTM + No Pose	CONV1D + Pose	CONV1D + No Pose	Average likelihood of model detection %
A	Yes	Yes	Yes	Yes	Yes	Yes	100
B	Yes	Yes	Yes	Yes	Yes	Yes	100
C	Yes	Yes	Yes	Yes	Yes	Yes	100
D	Yes	Yes	Yes	Yes	Yes	Yes	100
E	Yes	Yes	Yes	Yes	Yes	Yes	100
F	Yes	Yes	Yes	Yes	Yes	Yes	100
G	Yes	Yes	No	Yes	Yes	Yes	83.5
H	No	Yes	No	Yes	No	Yes	50.1
I	No	Yes	No	Yes	No	No	33.4
J	Yes	No	Yes	No	No	No	33.4
K	Yes	Yes	No	Yes	Yes	Yes	83.5
L	Yes	Yes	Yes	Yes	Yes	Yes	100
M	No	Yes	No	No	No	Yes	16.7
N	No	Yes	No	No	No	No	16.7
O	Yes	Yes	No	No	No	Yes	50.1
P	Yes	Yes	No	Yes	No	Yes	66.8
Q	Yes	Yes	Yes	Yes	Yes	Yes	100
R	Yes	Yes	Yes	Yes	Yes	Yes	100
S	Yes	Yes	Yes	Yes	Yes	Yes	100
T	Yes	Yes	Yes	Yes	Yes	Yes	100
U	Yes	Yes	Yes	Yes	Yes	Yes	100
V	Yes	No	No	Yes	Yes	No	50.1
W	No	Yes	Yes	Yes	Yes	Yes	83.5
X	Yes	No	Yes	No	No	Yes	50.1
Y	Yes	Yes	Yes	Yes	Yes	No	83.5
Z	Yes	No	Yes	Yes	Yes	No	66.8
.	Yes	Yes	Yes	Yes	No	No	66.8
space	Yes	Yes	Yes	Yes	Yes	Yes	100
Total = 28	23	24	19	23	19	21	

Real Time Alphabet Recognition (Yes/No)

5.4 Discussion

The results from sub section 5.1 shows that GRU performed the best in real time testing achieving an accuracy of 85.7% when only the hand landmarks were used for detection. The real time detection with pose was not far behind with 82.1%.

(M. H. ISMAIL, S. A. DAWWD and F. H. ALI, 2021.) achieved a 99.03% on the ARSL dataset but did not test in real time. Given the challenges with mediapipe jittering it will be

beneficial to see some real time test with the generated data and not just on test data. This study achieved a maximum accuracy of 99.3% with the GRU on test data using pose in combination with the hand landmarks but had a real time accuracy of 82.1%.

(D. BISHT *et al.,* 2022) achieved the best result using two hands in real time test, but this was for just 6 gestures introduced into the study from ISL and not on a full two-hand sign language dataset like the one used in this study. The study also implemented some important features like speech to text and text to speech with the auto correct feature for words which is vital to commercial useability.

(A. CHAIKAEW, K. SOMKUAN and T. YUYEN, 2021) achieved a 97% on the test data for 6 ISL gestures in real time. This study does better in the aspect of two hand signing because using similar approach achieved a 92.6 % accuracy in real time on the full BSL dataset with 2 extra labels making a total of 28.

(S. ADHIKARY, A. K. TALUKDAR) achieved a 97.4% accuracy in ISL recognition using just 11 gestures. The result from the confusion matrix was hard to interpret because of the type of confusion matrix plot used and the amount of data per class was not specified.

(V. H. IYER *et al.,* 2022.) used mediapipe to build a gesture recognition system on three distinct gestures and achieved a real time accuracy of 87.5% which is inadequate given how distinct the signs are and the number of classes used.

This work achieved a better result on full datasets than any of the study done so far. The BSL system developed is capable of both static and dynamic sign language recognition. This work also gives a good real time accuracy on both the pose and non-pose methods and shows that both have their strengths and weaknesses as can be seen from the Table showing the real time accuracy. The average of each label recognized across the 6 models is taken to give the percentage likelihood of that label being recognized in real time. Any label having over 80% have a very good likelihood of being detected in real time. M and N performed the worst with just a 16.7% chance of being recognized because of how similar the signs are. J is a dynamic language and did not seem to be recognized when the model trained to detect with just the hand landmarks were used, but dynamic gestures created for “.” and space character had no such problems.

Chapter 5

5.1 Summary

A suitable dataset was gotten from the university college London and used as a baseline to generate the data used to train the deep learning models. The generated data was stored in a structured file system as explained in sub section 3.1.1. The data was then pre-processed as explained in sub section 4.1.8. The various models were trained and evaluated after each training using test data and real time testing. Two methods were used to develop models using pose in combination with hand landmarks and just hand landmarks. A graphical user interface was built and used to perform the model inference as that gives practicality to the work. After extensive testing the results showed that GRU slightly outperformed LSTM and CONV1D in real time testing and had a better generalization capability in this instance. The result also showed that sign language recognition without pose landmarks performed better overall although not without its limitations. The proposed method was able to detect both static and dynamic sign languages in the BSL dataset.

5.2 Conclusion

This study born out of the need to bridge the communication gap between the public and BSL signers with severe hearing loss using artificial intelligence, achieved its aim. All objectives set out at the beginning of the study were achieved and more. The study also proved that it is possible to develop a truly scalable BSL recognition system by merging Google mediapipe framework and deep learning. During the study two viable ways using the mediapipe framework was used and they can both be used to accomplish BSL sign language recognition given more data and time. The study also showed some of the challenges with the mediapipe landmark detection system and how it might affect the data collection process and real time testing. This work answered the research question and proved that using a Vision based AI system the communication gap between the deaf-mute community of BSL signers and the public can be bridged.

Limitations

The are several limitations in the system and model developed and they have been itemized below

1. Generating the data using media pipe can create a lot of noise in the data because of the flickering and there is not any known way to clean the data like in normal machine learning process.

2. Because of the flickering the real time testing trails behind the test data accuracy due to the type of data generated.

3. Letters like L, M, N, V, O, I, U having similar signs cannot be predicted with certainty because of the flickering during real time testing.

4. Not enough data is used in the training process.

5. A sign was not developed for “delete” meaning a label cannot be deleted from the sentence during sentence formation even if it is wrong.

6. Sign for C in the BSL dataset and the sign developed for “.” are often confused during real time testing when models using both pose and hand landmarks were used.

7. BSL recognition using just hand landmarks for recognition will work well in fingerspelling but have real limitations when the pose plays a key factor which is more real-world scenario. A simple example of that is the sign for brown that requires the palm be rubbed on the arms. Without pose estimation it will be impossible to detect that.

Recommendations/Future work

Looking back several steps could have been done differently, new methods and algorithms tried. This would have helped improve the quality of the data and models generated.

1. Restructure the data during the pre-processing step to make it digestible by machine learning algorithms like SVM and random forest classifier for static sign language recognition as the study is structured towards both static and dynamic SLR as the latter is a time series analysis which non-linear models cannot handle.

2. The method of data generation process by taking pictures that are of high resolution and passing them through the mediapipe landmark detection model can be tried to help reduce the flickering problem.

3. Try to classify signs into static dynamic and no sign before any classification process as done by (M. H. ISMAIL, S. A. DAWWD and F. H. ALI, 2021.

4. Generate more data that can be used for training to make the model more robust and increase the generalization ability.

5. Generate data that includes more of the landmark key points as this study takes only the arms and shoulders. This will help the model more adaptable to real world situations.

6. Build a smartphone app that can be used to test the model in a real-world scenario and check scalability

7. Try an increase in the number of frames used in prediction in real time and train the model accordingly which might increase the accuracy in real time as shown by (D. BISHT *et al.*, 2022), this approach might take a longer time for the model to predict but with advances in the processing power of devices that challenge can be easily overcome in the future.

8. From the results in the real time result table the alphabets that have below 80% in the average prediction across all model can benefit from better data collection.

References

1. A. RUSTAGI, SHAINA and N. SINGH, 2021. American and Indian Sign Language Translation using Convolutional Neural Networks. - *2021 8th International Conference on Signal Processing and Integrated Networks (SPIN)*. pp.646-651
2. N. BUCKLEY, L. SHERRETT and E. LINDO SECCO, 2021. A CNN sign language recognition system with single & double-handed gestures. - *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. pp.1250-1253
3. BAHARETH MOHAMMED, 2022. ARTIFICIAL INTELLIGENCE IN GOVERNMENTS SECTORS.
4. ADEYANJU, I.A., O.O. BELLO and M.A. ADEGBOYE, 2021. Machine learning methods for sign language recognition: A critical review and analysis. *Intelligent Systems with Applications,* 12, 200056
5. M. MARAIS *et al.*, 2022. An Evaluation of Hand-Based Algorithms for Sign Language Recognition. - *2022 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*. pp.1-6
6. D. BISHT *et al.*, 2022. Smart Communication System Using Sign Language Interpretation. - *2022 31st Conference of Open Innovations Association (FRUCT)*. pp.12-20
7. V. H. IYER *et al.*, 2022. Sign Language Detection using Action Recognition. - *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*. pp.1682-1685
8. M. H. ISMAIL, S. A. DAWWD and F. H. ALI, 2021. Arabic Sign Language Detection Using Deep Learning Based Pose Estimation. - *2021 2nd Information Technology To Enhance e-learning and Other Application (IT-ELA)*. pp.161-166
9. A. CHAIKAEW, K. SOMKUAN and T. YUYEN, 2021. Thai Sign Language Recognition: an Application of Deep Neural Network. - *2021 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunication Engineering*. pp.128-131
10. S. ADHIKARY, A. K. TALUKDAR and K. KUMAR SARMA, 2021. A Vision-based System for Recognition of Words used in Indian Sign Language Using MediaPipe. - *2021 Sixth International Conference on Image Information Processing (ICIIP)*. pp.390-394
11. NAVEENKUMAR, M. and A. VADIVEL, 2015. OpenCV for computer vision applications. *Proceedings of national conference on big data and cloud computing (NCBDC'15)*. pp.52-56
12. I. CULJAK *et al.*, 2012. A brief introduction to OpenCV. - *2012 Proceedings of the 35th International Convention MIPRO*. pp.1725-1730
13. LUGARESI, C. *et al.*, 2019. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172,*
14. T, R., 2020. LSTMs Explained: A Complete, Technically Accurate, Conceptual Guide with Keras.
15. KHANDELWAL, R., 2019. Multivariate Time Series using Gated Recurrent Unit -GRU.
16. KIRANYAZ, S. *et al.*, 2021. 1D convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing,* 151, 107398
17. KASAPBAŞI, A. *et al.*, 2022. DeepASLR: A CNN based human computer interface for American Sign Language recognition for hearing-impaired individuals. *Computer Methods and Programs in Biomedicine Update,* 2, 100048
18. KATOCH, S., V. SINGH and U.S. TIWARY, 2022. Indian Sign Language recognition system using SURF with SVM and CNN. *Array,* 14, 100141

19. N. BUCKLEY, L. SHERRETT and E. LINDO SECCO, 2021. A CNN sign language recognition system with single & double-handed gestures. - *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. pp.1250-1253
20. W. SULIMAN *et al.*, 2021. Arabic Sign Language Recognition Using Deep Machine Learning. - *2021 4th International Symposium on Advanced Electrical and Communication Technologies (ISAECT)*. pp.1-4
21. ADEYANJU, I.A., O.O. BELLO and M.A. ADEGBOYE, 2021. Machine learning methods for sign language recognition: A critical review and analysis. *Intelligent Systems with Applications,* 12, 200056
22. JEAN, B.G., 1979. Gestural Linguistics: *The Signs of Language*. Edward S. Klima and Ursula Bellugi with ten others. Harvard University Press, Cambridge, Mass., 1979. xiv, 418 pp., illus. $25. *Science,* 205(4412), 1253-1254
23. M. QUINN and J. I. OLSZEWSKA, 2019. British Sign Language Recognition In The Wild Based On Multi-Class SVM. - *2019 Federated Conference on Computer Science and Information Systems (FedCSIS)*. pp.81-86
24. Y. MORI and M. TOYONAGA, 2018. Data-Glove for Japanese Sign Language Training System with Gyro-Sensor. - *2018 Joint 10th International Conference on Soft Computing and Intelligent Systems (SCIS) and 19th International Symposium on Advanced Intelligent Systems (ISIS)*. pp.1354-1357
25. H. S. ANUPAMA *et al.*, 2021. Automated Sign Language Interpreter Using Data Gloves. - *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*. pp.472-476
26. N. TUBAIZ, T. SHANABLEH and K. ASSALEH, 2015. *Glove-Based Continuous Arabic Sign Language Recognition in User-Dependent Mode*. , pp.526-533
27. A. ABDULLAH, N. A. ABDUL-KADIR and F. K. CHE HARUN, 2020. An Optimization of IMU Sensors-Based Approach for Malaysian Sign Language Recognition. - *2020 6th International Conference on Computing Engineering and Design (ICCED)*. pp.
28. WILLIAMS ORENDA, Understanding Matrices to Perform Basic Image Processing on Digital Images.
29. SOFFER, S. *et al.*, 2019. Convolutional neural networks for radiologic images: a radiologist's guide. *Radiology,* 290(3), 590-606
30. M., S., P. M.V.D. and K. P.V.V., 2021. Multi-view motion modelled deep attention networks (M2DA-Net) for video based sign language recognition. *Journal of Visual Communication and Image Representation,* 78, 103161
31. KATILMIŞ, Z. and C. KARAKUZU, 2021. ELM based two-handed dynamic Turkish Sign Language (TSL) word recognition. *Expert Systems with Applications,* 182, 115213

Appendix

Section A Ethics Application form

Ethical clearance for research and innovation projects

Project status

Status

Approved

Actions

Date	Who	Action	Comments
11:00:00 15 July 2022	Femi Isiaq	Supervisor approved	
10:40:00 15 July 2022	Gabriel Braimah	Principal investigator submitted	full title has been inputted
06:49:00 11 July 2022	Femi Isiaq	Supervisor declined	You need to input the full title of your project and resubmit your application.
12:55:00 09 July 2022	Gabriel Braimah	Principal investigator submitted	

Get Help

Ethics release checklist (ERC)

Project details

Project name: Bridging the communication gap between deaf-dumb individuals and the public using sign language recognition with convolutional neural network.

Principal investigator: Gabriel Braimah

Faculty: Faculty of Business, Law and Digital Technologies

Level: Postgraduate

Course: MAIDS

Unit code: COM726

Supervisor name: Femi Isiaq

Supervisor search:

Checklist

Question	Yes	No
Q1. Will the project involve human participants other than the investigator(s)?	○	◉
Q1a. Will the project involve **vulnerable participants** such as children, young people, disabled people, the elderly, people with declared mental health issues, prisoners, people in health or social care settings, addicts, or those with learning difficulties or cognitive impairment either contacted directly or via a **gatekeeper** (for example a professional who runs an organisation through which participants are accessed; a service provider; a care-giver; a relative or a guardian)?	○	◉
Q1b. Will the project involve the use of **control groups** or the **use of deception**?	○	◉
Q1c. Will the project involve any **risk to the participants' health** (e.g. intrusive intervention such as the administration of drugs or other substances, or vigorous physical exercise), or involve psychological stress, anxiety, humiliation, physical pain or discomfort to the investigator(s) and/or the participants?	○	◉
Q1d. Will the project involve **financial inducement** offered to participants other than reasonable expenses and compensation for time?	○	◉
Q1e. Will the project be carried out by individuals unconnected with the University but who wish to use staff and/or students of the University as participants?	○	◉
Q2. Will the project involve sensitive materials or topics that might be considered offensive, distressing, politically or socially sensitive, deeply personal or in breach of the law (for example criminal activities, sexual behaviour, ethnic status, personal appearance, experience of violence, addiction, religion, or financial circumstances)?	○	◉
Q3. Will the project have detrimental impact on the environment, habitat or species?	○	◉
Q4. Will the project involve living animal subjects?	○	◉
Q5. Will the project involve the development for export of 'controlled' goods regulated by the Export Control Organisation (ECO)? (This specifically means military goods, so called dual-use goods (which are civilian goods but with a potential military use or application), products used for torture and repression, radioactive sources.) Further information from the Export Control Organisation	○	◉
Q6. Does your research involve: the storage of records on a computer, electronic transmissions, or visits to websites, which are associated with terrorist or extreme groups or other security sensitive material? Further information from the Information Commissioners Office	○	◉

Declarations

I/we, the investigator(s), confirm that:

- [x] The information contained in this checklist is correct.

- [x] I/we have assessed the ethical considerations in relation to the project in line with the University Ethics Policy.

- [x] I/we understand that the ethical considerations of the project will need to be re-assessed if there are any changes to it.

- [x] I/we will endeavour to preserve the reputation of the University and protect the health and safety of all those involved when conducting this research/enterprise project.

- [x] If personal data is to be collected as part of my project, I confirm that my project and I, as Principal Investigator, will adhere to the General Data Protection Regulation (GDPR) and the Data Protection Act 2018. I also confirm that I will seek advice on the DPA, as necessary, by referring to the Information Commissioner's Office further guidance on DPA and/or by contacting information.rights@solent.ac.uk. By Personal data, I understand any data that I will collect as part of my project that can identify an individual, whether in personal or family life, business or profession.

- [x] I/we have read the prevent agenda.

Section B: Evaluation figures

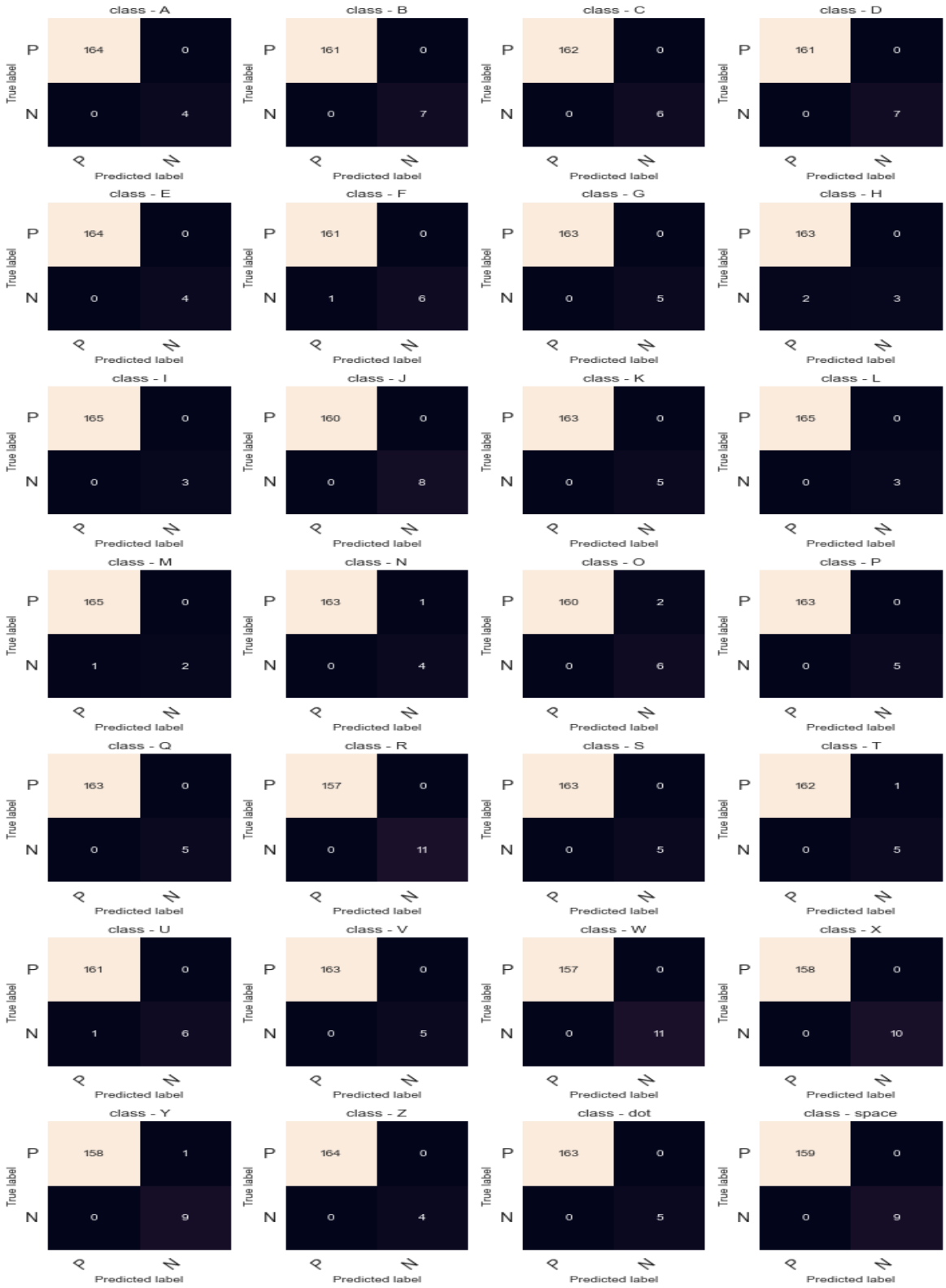


Figure 51:confusion matrix for pose LSTM

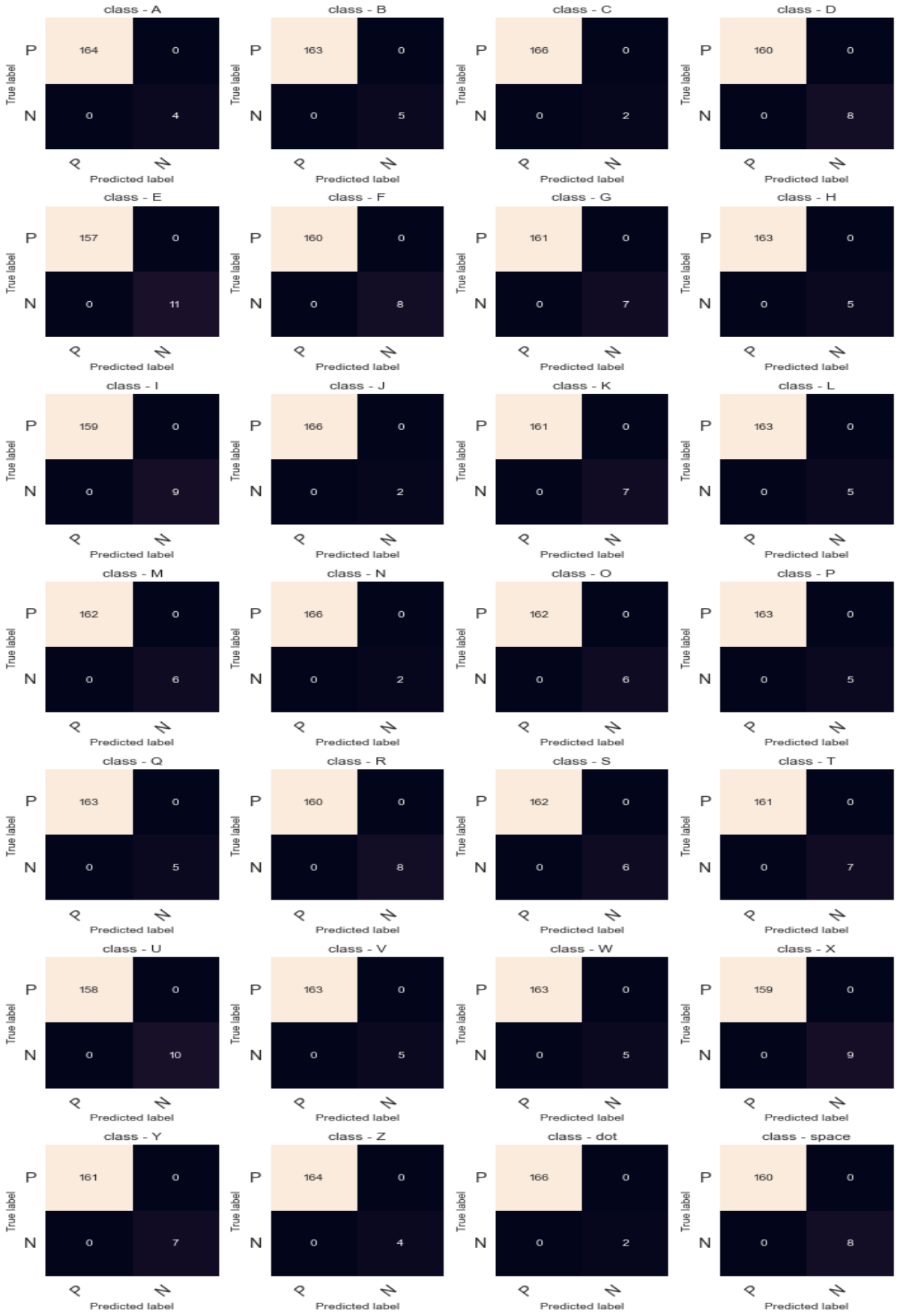


Figure 52: LSTM confusion matrix without pose

Figure 53: confusion matrix for pose GRU

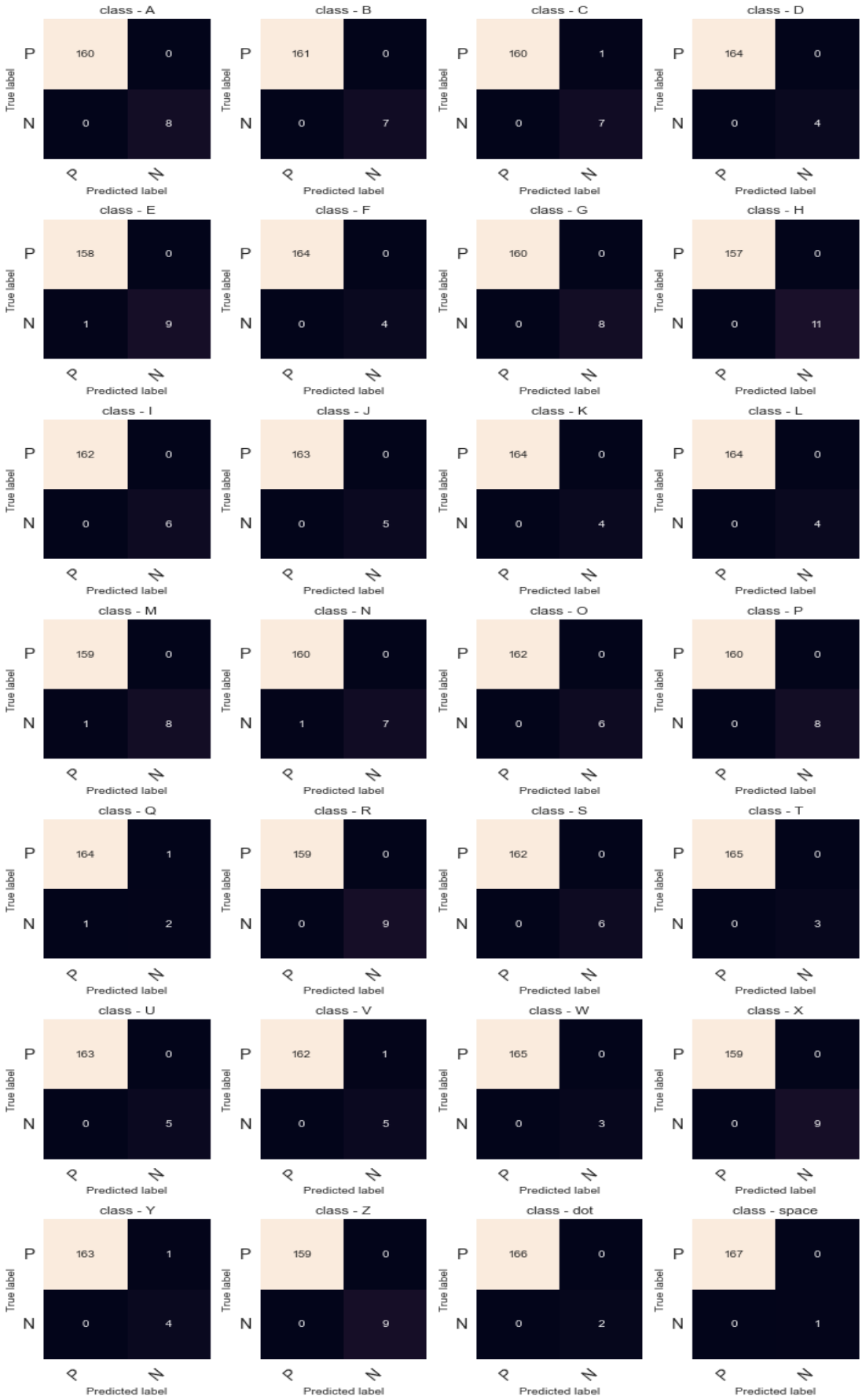


Figure 54: confusion matrix for non-pose GRU

Figure 55: confusion matrix for CONV1D no pose

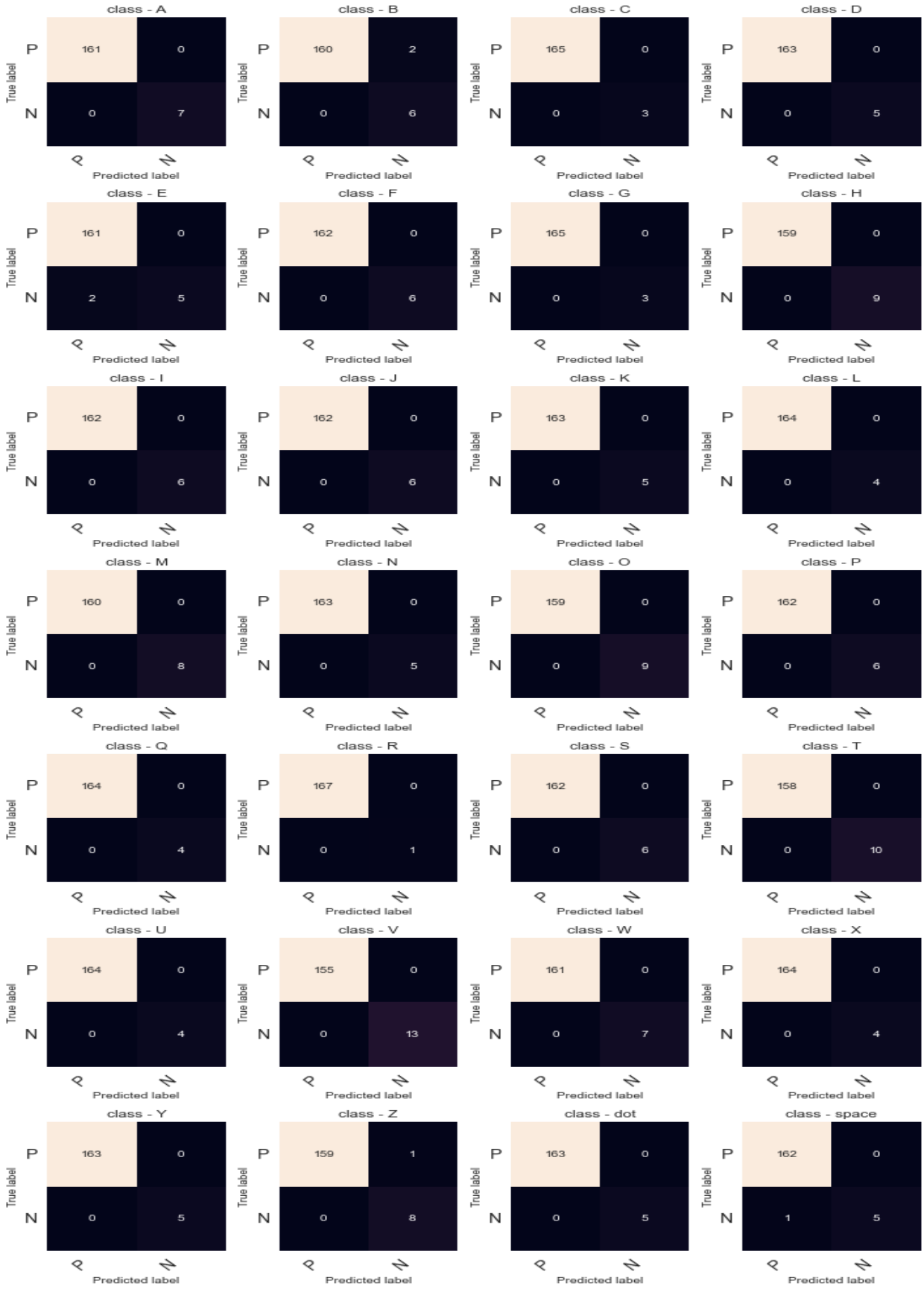


Figure 56:confusion matrix for CONV1D using pose and hand landmarks for prediction

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 28, 150)           56850

 max_pooling1d (MaxPooling1D  (None, 14, 150)          0
 )

 conv1d_1 (Conv1D)           (None, 12, 150)           67650

 max_pooling1d_1 (MaxPooling  (None, 6, 150)           0
 1D)

 conv1d_2 (Conv1D)           (None, 4, 150)            67650

 max_pooling1d_2 (MaxPooling  (None, 2, 150)           0
 1D)

 flatten (Flatten)           (None, 300)               0

 dense (Dense)               (None, 100)               30100

 dropout (Dropout)           (None, 100)               0

 dense_1 (Dense)             (None, 28)                2828

=================================================================
Total params: 225,078
Trainable params: 225,078
Non-trainable params: 0
```

Figure 57: training parameters for conv1D no pose

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 30, 200)           276000

 gru_1 (GRU)                 (None, 30, 400)           722400

 gru_2 (GRU)                 (None, 200)               361200

 dense (Dense)               (None, 125)               25125

 dropout (Dropout)           (None, 125)               0

 dense_1 (Dense)             (None, 64)                8064

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 25)                1625

=================================================================
Total params: 1,394,414
Trainable params: 1,394,414
Non-trainable params: 0
_________________________________________________________________
```

Figure 58: GRU parameters

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 lstm (LSTM)                  (None, 30, 150)           245400

 lstm_1 (LSTM)                (None, 150)               180600

 dense_3 (Dense)              (None, 100)               15100

 dropout_2 (Dropout)          (None, 100)               0

 dense_4 (Dense)              (None, 62)                6262

 dropout_3 (Dropout)          (None, 62)                0

 dense_5 (Dense)              (None, 25)                1575

=================================================================
Total params: 448,937
Trainable params: 448,937
Non-trainable params: 0
_________________________________________________________________
```

Figure 59:LSTM training parameters

Section C: Entire code for implementation

```
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import mediapipe as mp
from keras.models import Sequential,load_model
from keras.layers import LSTM, Dense, Dropout, GRU, Conv1D, MaxPooling1D, Flatten
import pyttsx3 as audio
```

```
class Braimah:
    mp_holistic = mp.solutions.holistic # Holistic model
    mp_drawing = mp.solutions.drawing_utils # Drawing utilities
    no_sequences = 60 # number of videos per action
    frame_length = 30 # length of frames
    DATA_PATH: os.path = os.path.join('data_with_gloves')
    threshold = 0.9
    selected_actions = np.array(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L','M','N','O','P','Q','R','S','T',

    def mediapipe_detection(self,image, model):
        img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # COLOR CONVERSION BGR 2 RGB
        img.flags.writeable = False                  # Image is no longer writeable
        res = model.process(image)                # Make prediction
        img.flags.writeable = True                   # Image is now writeable
        img = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) # COLOR CONVERSION RGB 2 BGR
        return img, res

    def draw_styled_landmarks(self, image, results, pose):
        if pose:
            self.mp_drawing.draw_landmarks(image, results.pose_landmarks, self.mp_holistic.POSE_CONNECTIONS,
                                self.mp_drawing.DrawingSpec(color=(80,22,10), thickness=1, circle_radius=4),
                                Braimah.mp_drawing.DrawingSpec(color=(80,44,121), thickness=1, circle_radius=2)
                                )
        else:
            pass
        # Draw left-hand connections
        self.mp_drawing.draw_landmarks(image, results.left_hand_landmarks, self.mp_holistic.HAND_CONNECTIONS,
                            self.mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                            self.mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                            )
        # Draw right-hand connections
        self.mp_drawing.draw_landmarks(image, results.right_hand_landmarks, self.mp_holistic.HAND_CONNECTIONS,
                            self.mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                            self.mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                            )
    def show_image(self, image):
        cv2.imshow('Openfeed CV',image)

    def extract_datapoints(self, frame_results, pose):
        ps = np.array([[rs.x, rs.y, rs.z, rs.visibility] for rs in frame_results.pose_landmarks.landmark]).flatten() if fram
        lh = np.array([[rs.x, rs.y, rs.z] for rs in frame_results.left_hand_landmarks.landmark]).flatten() if frame_results.
        rh = np.array([[rs.x, rs.y, rs.z] for rs in frame_results.right_hand_landmarks.landmark]).flatten() if frame_results
        if pose:
            return np.concatenate([ps, lh, rh])
        else:
            return np.concatenate([lh,rh])

    def create_folders(self):
        for action in self.selected_actions:
            for sequence in range(self.no_sequences):
                try:
                    os.makedirs(os.path.join(self.DATA_PATH, action, str(sequence)))
                except:
                    pass

    def create_model(self,model_type, pose):
        if pose:
            datapoints = 258
        else:
```

```
        else:
            datapoints = 126
        if model_type == 'LSTM':
            model = Sequential()
            model.add(LSTM(180, return_sequences=True, activation='tanh', dropout= 0.3, input_shape=(30,datapoints)))
            #model.add(LSTM(250, return_sequences=True, activation='tanh', dropout=0.1))
            #model.add(LSTM(250, return_sequences=True, activation='tanh',dropout=0.1))
            model.add(LSTM(180, return_sequences=False, activation='tanh',dropout=0.3))
            model.add(Dense(100, activation='relu'))
            model.add(Dropout(0.2))
            model.add(Dense(62, activation='relu'))
            model.add(Dropout(0.2))
            model.add(Dense(self.selected_actions.shape[0], activation='softmax'))
            model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
            return model
        elif model_type == 'CONV1D':
            model = Sequential()
            model.add(Conv1D(filters=160, kernel_size=3, activation='relu', input_shape=(30,datapoints)))
            model.add(MaxPooling1D(pool_size=2))
            model.add(Conv1D(filters=160, kernel_size=3, activation='relu'))
            model.add(MaxPooling1D(pool_size=2))
            model.add(Flatten())
            model.add(Dense(100, activation='relu'))
            model.add(Dropout(0.2))
            model.add(Dense(self.selected_actions.shape[0], activation='softmax'))
            model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
            return model
        elif model_type == 'GRU':
            model = Sequential()
            model.add(GRU(200, return_sequences=True, activation='tanh', input_shape=(30,datapoints),dropout=0.2))
            model.add(GRU(400, return_sequences=True, activation='tanh',dropout=0.2))
            model.add(GRU(200, return_sequences=False, activation='tanh',dropout=0.2))
            #model.add(GRU(1000, return_sequences=True, activation='tanh',dropout=0.1))
            #model.add(GRU(400, return_sequences=False, activation='tanh',dropout=0.1))
            model.add(Dense(125, activation='relu'))
            model.add(Dropout(0.1))
            model.add(Dense(64, activation='relu'))
            model.add(Dropout(0.1))
            model.add(Dense(self.selected_actions.shape[0], activation='softmax'))
            model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
            return model
        else:
            pass


    @staticmethod
    def save_keypoints(data_path,action,sequence_no,frame_num,keypoints):
         npy_path = os.path.join(data_path, action, str(sequence_no), str(frame_num))
         np.save(npy_path, keypoints)

    def model_predict(self, sequence, model, sentence, rf=False):
    #   sequence.insert(0,keypoints)
        if rf is True:
            prediction = model.predict(np.array(sequence[:30]).reshape(-1,30 * 258))
            prediction = np.argmax(prediction, axis=1)
            print(self.selected_actions[prediction])
            if self.selected_actions[prediction] == 'dot':
                sentence.append('.')
                print(' '.join(sentence))
            elif self.selected_actions[np.argmax(prediction)] == 'space':
                sentence.append(' ')
                print(' '.join(sentence))
            else:
                sentence.append(self.selected_actions[self.selected_actions[prediction]])
                print(' '.join(sentence))
        else:
            prediction = model.predict(np.expand_dims(sequence[:30], axis=0))[0]
            if prediction[np.argmax(prediction)] >= self.threshold :
                if self.selected_actions[np.argmax(prediction)] == 'dot':
```

```
                elif self.selected_actions[np.argmax(prediction)] == 'space':
                    sentence.append(' ')
                    print(' '.join(sentence))
                else:
                    sentence.append(vision_class.selected_actions[np.argmax(prediction)])
                    print(' '.join(sentence))
            else:
                pass
```

2. Keypoints using MP Holistic

Type *Markdown* and LaTeX: α^2

In [3]:
```
vision_class = Braimah()
is_pose = True
```

In [4]:
```
cap = cv2.VideoCapture(0)
# Set mediapipe model
with vision_class.mp_holistic.Holistic(static_image_mode=False, min_detection_confidence=0.7,min_tracking_confidence=0.7) as
    while cap.isOpened():
        # Read feed
        _, frame = cap.read()
        # Make detections
        image, results = vision_class.mediapipe_detection(frame, holistic)
        # Draw landmarks
        vision_class.draw_styled_landmarks(image, results, is_pose)
        # Show to screen
        cv2.imshow('OpenCV Feed', cv2.cvtColor(image,cv2.COLOR_BGR2RGB))
        # Break gracefully
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

In [ ]:
```
vision_class.draw_styled_landmarks(frame, results)
```

In [ ]:
```
plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
```

3. Extract Keypoint Values

In [ ]:
```
result_test = vision_class.extract_datapoints(results)
```

In [ ]:
```
#result_test
```

In [ ]:
```
np.save('0', result_test)
```

In [ ]:
```
np.load('0.npy').shape
```

4. Setup Folders for Collection

In [6]:
```
vision_class.create_folders()
```

In [6]:
```
vision_class.create_folders()
```

5. Collect Keypoint Values for Training and Testing

n [231]:
```
cap = cv2.VideoCapture(0)
# Set mediapipe model
singular_action = ['M']
frame_num = 0
with vision_class.mp_holistic.Holistic(static_image_mode=True, min_detection_confidence=0.6) as holistic:
    # Loop through actions
    for action in singular_action:
        # Loop through sequences aka videos
        for sequence in range(vision_class.no_sequences):
            # Loop through video length aka sequence length
            try:
                if sequence == 0:
                    _, first_frame = cap.read()
                    cv2.putText(first_frame, 'Collecting frames for {} sequence Number {}'.format(action, sequence), (15,12)
                    cv2.imshow('OpenCV Feed', first_frame)
                    cv2.waitKey(5000)
                while frame_num < 30:
                    # Read feed
                    _, frame = cap.read()
                    # Make detections
                    image, results = vision_class.mediapipe_detection(frame, holistic)

                    # Draw landmarks
                    vision_class.draw_styled_landmarks(frame, results, is_pose)
                    cv2.putText(frame, 'Collecting frames for {} sequence Number {}'.format(action, sequence), (15,12),cv2.F
                    cv2.imshow('OpenCV Feed', frame)

                    # NEW Export keypoints
                    keypoints = vision_class.extract_datapoints(results,is_pose)
                    Braimah.save_keypoints(vision_class.DATA_PATH,action,sequence,frame_num,keypoints)
                    frame_num += 1
                    # Break gracefully
                frame_num = 0
            except RuntimeError as e :
                print(e)

            if cv2.waitKey(10) & 0xFF == ord('q'):
                break

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

6. Preprocess Data and Create Labels and Features

In [4]:
```
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
```

In [5]:
```
label_map = {label:num for num, label in enumerate(vision_class.selected_actions)}
```

Type *Markdown* and LaTeX: α^2

In [6]:
```
label_map
```

Out[6]:
```
{'A': 0,
 'B': 1,
 'C': 2,
 'D': 3,
 'E': 4,
 'F': 5,
 'G': 6,
 'H': 7,
 'I': 8,
 'J': 9,
 'K': 10,
 'L': 11,
 'M': 12,
 'N': 13,
 'O': 14,
 'P': 15,
 'Q': 16,
 'R': 17,
 'S': 18,
 'T': 19,
 'U': 20,
 'V': 21,
 'W': 22,
 'X': 23,
 'Y': 24,
 'Z': 25,
 'dot': 26,
 'space': 27}
```

In [7]:
```
sequences, labels = [], []
for action in vision_class.selected_actions:
    for sequence in range(vision_class.no_sequences):
        window = []
        for frame_num in range(vision_class.frame_length):
            res = np.load(os.path.join(vision_class.DATA_PATH, action, str(sequence),
                                       "{}.npy".format(frame_num)))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])
```

In [8]:
```
X = np.array(sequences)
X.shape
```

Out[8]:
```
(1680, 30, 258)
```

In [9]:
```
y = to_categorical(labels).astype(int)
y
```

Out[9]:
```
array([[1, 0, 0, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 1],
       [0, 0, 0, ..., 0, 0, 1],
       [0, 0, 0, ..., 0, 0, 1]])
```

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10)
```

In [11]:
```
X_train.shape
```

Out[11]:
```
(1512, 30, 258)
```

In [12]:
```
y_test.shape
```

Out[12]:
```
(168, 28)
```

In [13]:
```
y_train.shape
```

Out[13]:
```
(1512, 28)
```

7. Build and Train Neural Networks

Type *Markdown* and LaTeX: α^2

In [22]:
```
model = vision_class.create_model(model_type='CONV1D',pose=is_pose)
```

In [23]:
```
model.summary()
```

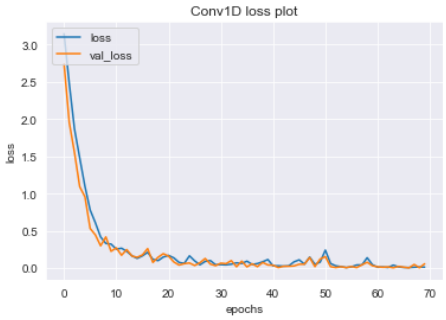
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_4 (Conv1D)           (None, 28, 160)           124000

 max_pooling1d_4 (MaxPooling  (None, 14, 160)          0
 1D)

 conv1d_5 (Conv1D)           (None, 12, 160)           76960

 max_pooling1d_5 (MaxPooling  (None, 6, 160)           0
 1D)

 flatten_2 (Flatten)         (None, 960)               0

 dense_4 (Dense)             (None, 100)               96100

 dropout_2 (Dropout)         (None, 100)               0

 dense_5 (Dense)             (None, 28)                2828

=================================================================
Total params: 299,888
Trainable params: 299,888
Non-trainable params: 0
_________________________________________________________________
```

In [24]:
```
from keras.callbacks import EarlyStopping
#early_stopping = EarlyStopping(monitor='val_loss',patience=10,mode='min',restore_best_weights=True, baseline=0.99)
history = model.fit(X_train, y_train,validation_split= 0.1,epochs=70)
```

```
Epoch 1/70
43/43 [==============================] - 1s 11ms/step - loss: 3.1446 - accuracy: 0.0831 - val_loss: 2.7406 - val_accuracy:
0.2368
Epoch 2/70
43/43 [==============================] - 0s 3ms/step - loss: 2.4764 - accuracy: 0.2044 - val_loss: 1.9534 - val_accuracy:
0.4671
Epoch 3/70
43/43 [==============================] - 0s 3ms/step - loss: 1.8669 - accuracy: 0.3721 - val_loss: 1.5507 - val_accuracy:
0.5395
Epoch 4/70
43/43 [==============================] - 0s 3ms/step - loss: 1.4786 - accuracy: 0.4875 - val_loss: 1.0926 - val_accuracy:
0.5987
Epoch 5/70
43/43 [==============================] - 0s 4ms/step - loss: 1.0973 - accuracy: 0.6360 - val_loss: 0.9567 - val_accuracy:
0.6908
Epoch 6/70
43/43 [==============================] - 0s 3ms/step - loss: 0.7794 - accuracy: 0.7360 - val_loss: 0.5319 - val_accuracy:
0.8421
```

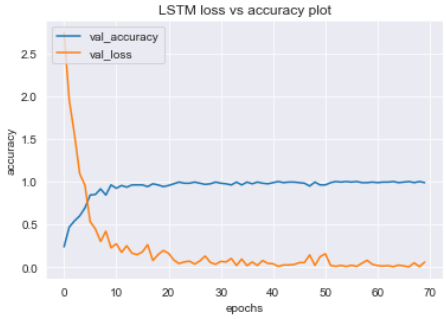
In [25]:
```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Conv1D loss plot')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend(['loss','val_loss'], loc='upper left')
```

Out[25]:
```
<matplotlib.legend.Legend at 0x2b805277fd0>
```



In [26]:
```
import matplotlib.pyplot as plt
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['val_loss'])
plt.title('LSTM loss vs accuracy plot')
#plt.ylim((0,5))
plt.ylabel('accuracy')
plt.xlabel('epochs')
plt.legend(['val_accuracy','val_loss'], loc='upper left')
```

Out[26]:
```
<matplotlib.legend.Legend at 0x2b8052c6f10>
```



8. Make Predictions

In [27]:
```
result = model.predict(X_test)
```

```
6/6 [==============================] - 0s 56ms/step
```

In [29]:
```
vision_class.selected_actions[np.argmax(result[2])]
```

Out[29]:
```
'B'
```

In [29]:
```
vision_class.selected_actions[np.argmax(result[2])]
```

Out[29]:
```
'B'
```

In [19]:
```
result[2]
```

Out[19]:
```
array([6.80671484e-15, 2.16641216e-09, 1.59764443e-12, 1.39515519e-08,
       7.01346536e-08, 1.12836078e-05, 7.42461478e-15, 6.53732346e-10,
       2.06298591e-06, 9.64631825e-16, 3.06622978e-12, 4.48164624e-15,
       2.24035864e-13, 1.91586994e-14, 9.99959946e-01, 1.17993338e-07,
       4.06369554e-06, 4.16253844e-14, 1.06012005e-11, 3.99292659e-13,
       2.20065031e-05, 1.92686524e-13, 1.20504737e-07, 3.13930457e-07,
       3.00692492e-11, 6.97016347e-08, 4.94351449e-10, 8.73127308e-13],
      dtype=float32)
```

In [75]:
```
vision_class.selected_actions[np.argmax([.7, .2, .1])]
```

Out[75]:
```
'A'
```

In [28]:
```
vision_class.selected_actions[np.argmax(y_test[2])]
```

Out[28]:
```
'B'
```

9. Save Weights

In [30]:
```
model.save('data_gloves_CONV1D_pose.h5')
```

In [31]:
```
del model
```

In [32]:
```
model = load_model('data_gloves_CONV1D_pose.h5')
```

10. Evaluation using Confusion Matrix and Accuracy

In [33]:
```
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score, classification_report
```

In [34]:
```
y_predict = model.predict(X_test)
```

```
6/6 [==============================] - 0s 3ms/step
```

In [35]:
```
ytrue = np.argmax(y_test, axis=1).tolist()
y_predict = np.argmax(y_predict, axis=1).tolist()
```

In [36]:
```
cf = multilabel_confusion_matrix(ytrue, y_predict)
cf
```

Out[36]:
```
array([[[161,   0],
        [  0,   7]],

       [[160,   2],
        [  0,   6]],

       [[165,   0],
        [  0,   3]],

       [[163,   0],
        [  0,   5]],

       [[161,   0],
        [  2,   5]],

       [[162,   0],
        [  0,   6]],

       [[165,   0],
```

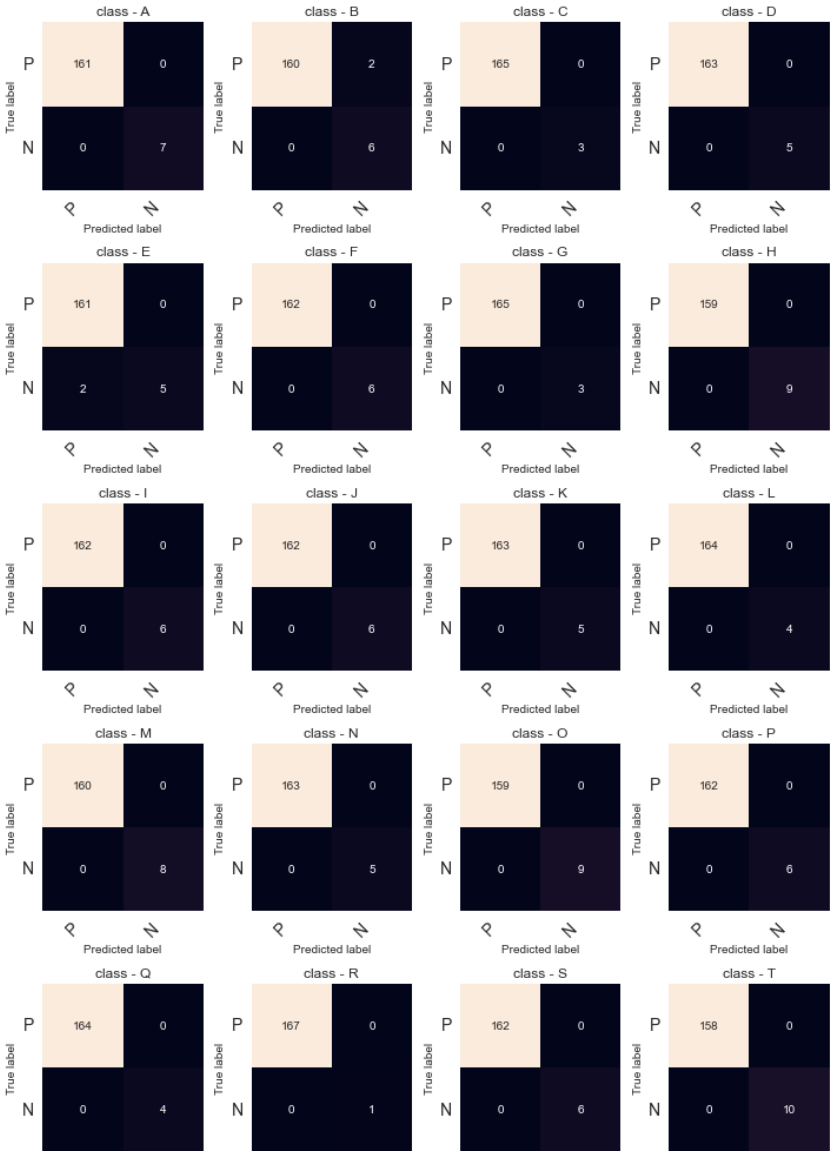
In [37]:
```
accuracy_score(ytrue, y_predict)
```

Out[37]:
```
0.9821428571428571
```

In [38]:
```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

def print_confusion_matrix(confusion_matrix, axes, class_label, class_names, fontsize=14):

    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
    )

    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d", cbar=False, ax=axes)
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right', fontsize=fontsize)
    axes.set_ylabel('True label')
    axes.set_xlabel('Predicted label')
    axes.set_title("class - " + class_label)
```

In [39]:

```
fig, ax = plt.subplots(7, 4, figsize=(10, 20))
for axes, cfs_matrix, label in zip(ax.flatten(), cf, vision_class.selected_actions):
    print_confusion_matrix(cfs_matrix, axes, label, ["P", "N"])
fig.tight_layout()
plt.show()

#cf_df = pd.DataFrame(cf, index=vision_class.selected_actions, columns=vision_class.selected_actions)
#sk.metrics.plot_confusion_matrix(ytrue,y_predict,figsize=(20,20))
```



In [40]:
```
print(classification_report(ytrue,y_predict))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       0.75      1.00      0.86         6
           2       1.00      1.00      1.00         3
           3       1.00      1.00      1.00         5
           4       1.00      0.71      0.83         7
           5       1.00      1.00      1.00         6
           6       1.00      1.00      1.00         3
           7       1.00      1.00      1.00         9
           8       1.00      1.00      1.00         6
           9       1.00      1.00      1.00         6
          10       1.00      1.00      1.00         5
          11       1.00      1.00      1.00         4
          12       1.00      1.00      1.00         8
          13       1.00      1.00      1.00         5
          14       1.00      1.00      1.00         9
          15       1.00      1.00      1.00         6
          16       1.00      1.00      1.00         4
```

11. Test in Real Time

In [ ]:
```
# 1. New detection variables
frame_sequence = []
sentence = []

cap = cv2.VideoCapture(0)
# Set mediapipe model
with vision_class.mp_holistic.Holistic(min_detection_confidence=0.6, min_tracking_confidence=0.6) as holistic:
        while cap.isOpened():
            _, frame = cap.read()
            image, result = vision_class.mediapipe_detection(frame, holistic)
            if result.right_hand_landmarks is None and result.left_hand_landmarks is None:
                frame_sequence.clear()
                count = 0
                vision_class.draw_styled_landmarks(image, result, is_pose)
                cv2.imshow('Testing feed', cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break
                continue
            else:
                vision_class.draw_styled_landmarks(image, result, is_pose)
                key_points = vision_class.extract_datapoints(result, is_pose)
                frame_sequence.append(key_points)
                cv2.imshow('Testing feed', cv2.cvtColor(image,cv2.COLOR_BGR2RGB))
                count += 1
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break
            if count == 30:
                vision_class.model_predict(frame_sequence, model, sentence, False)
                if len(sentence) == 0:
                    pass
                else:
                    if sentence[len(sentence) - 1] == '.':
                        sentence.pop()
                        audio.speak(''.join(sentence))
                        sentence.clear()
                    else:
                        pass
                count = 0
cap.release()
cv2.destroyAllWindows()
```