

# **SIGN LANGUAGE INTERPRETATION**

## **A MINI PROJECT REPORT**

**18CSC305J - ARTIFICIAL INTELLIGENCE**

*Submitted by*

**Thrideep.S [RA2111032010003]**

**Koushik [RA2111032010061]**

**Roshan [RA2111032010013]**

*Under the guidance of*

**Dr. C. Fancy**

Assistant Professor, Department of Networking and Communication

*in partial fulfillment for the award of the degree*

*of*

**BACHELOR OF TECHNOLOGY**

in

**COMPUTER SCIENCE & ENGINEERING**

of

**FACULTY OF ENGINEERING AND TECHNOLOGY**



S.R.M. Nagar, Kattankulathur, Chengalpattu District

**MAY 2024**

# **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

(Under Section 3 of UGC Act, 1956)

## **BONAFIDE CERTIFICATE**

Certified that Mini project report titled “**SIGN LANGUAGE INTERPRETER**” is the bonafide work of **Thrideep.S (RA2111032010003), Koushik (RA2111032010061), Roshan (RA2111032010013)** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

**SIGNATURE**

**Dr. C. Fancy**

Assistant Professor

Networking and Communication

## **ABSTRACT**

The Sign Language Interpreter project aims to bridge the communication gap between the deaf or hard-of-hearing community and the general public by developing a real-time sign language translation system. Sign language is a primary mode of communication for many deaf individuals, yet it presents a significant barrier for those who don't understand it. This project seeks to create a solution that translates sign language gestures into spoken language or text in real-time, facilitating better communication between deaf individuals and the wider community.

The system comprises several components, including computer vision for gesture recognition, machine learning models for classification, and natural language processing for translation. It utilizes a camera to capture hand gestures, processes the video feed to recognize and interpret the gestures, and then generates spoken or written translations. The project focuses on building a robust and accurate system capable of recognizing a wide range of sign language gestures in various environmental conditions.

# TABLE OF CONTENTS

|  |            |
|--|------------|
| <b>ABSTRACT</b>  | <b>iii</b> |
| <b>TABLE OF CONTENTS</b>   | <b>iv</b>  |
| <b>LIST OF FIGURES</b>   | <b>v</b>   |
| <b>ABBREVIATIONS</b>   | <b>vi</b>  |
| <b>1 INTRODUCTION</b>  | <b>7</b>   |
| <b>2 LITERATURE SURVEY</b>   | <b>8</b>   |
| <b>3 SYSTEM ARCHITECTURE AND DESIGN</b>                                  | <b>9</b>   |
| 3.1 Architecture diagram of proposed IoT based smart agriculture project | 9          |
| 3.2 Description of Module and components                                 | 10         |
| <b>4 METHODOLOGY</b>   | <b>14</b>  |
| 4.1 Methodological Steps   | 14         |
| <b>5 CODING AND TESTING</b>  | <b>15</b>  |
| <b>6 SREENSHOTS AND RESULTS</b>  |            |
| <b>7 CONCLUSION AND FUTURE ENHANCEMENT</b>                               | <b>23</b>  |
| <b>REFERENCES</b>  | <b>24</b>  |

## **LIST OF FIGURES**

|  |           |
|--|-----------|
| <b>3.1.1 Architecture block</b>                                      | <b>9</b>  |
| <b>3.2.1 Soil moisture sensor</b>                                    | <b>10</b> |
| <b>3.2.3 Temperature Sensor (DHT 11)</b>                             | <b>10</b> |
| <b>3.2.4 PIR Motion Sensor</b>                                       | <b>10</b> |
| <b>3.2.5 Motor</b>   | <b>11</b> |
| <b>3.2.6 Ultrasonic Sensor</b>                                       | <b>11</b> |
| <b>3.2.7 Buzzer</b>  | <b>12</b> |
| <b>3.2.8 WIFI MODULE ESP 8266</b>                                    | <b>13</b> |
| <b>6.1.1 Buzzer and PIR simulation on tinkercad</b>                  | <b>19</b> |
| <b>6.2.1 Ultrasonic sensor and Scarecrow simulation on tinkercad</b> | <b>19</b> |
| <b>6.3.1 Soilmoisture sensor and motor simulation on tinkercad</b>   | <b>20</b> |
| <b>6.4.1 LCD Screen with output</b>                                  | <b>20</b> |
| <b>6.5.1 Whole Circuit</b>   | <b>21</b> |
| <b>6.6.1 Screenshot of thingspeak server</b>                         | <b>22</b> |

## ABBREVIATIONS

|             |   |
|-------------|---|
| <b>IOT</b>  | Internet of Things                          |
| <b>PIR</b>  | Passive Infrared                            |
| <b>LCD</b>  | Liquid Crystal Diode                        |
| <b>DHT</b>  | Distributed hash table                      |
| <b>IR</b>   | Infra red                                   |
| <b>UART</b> | Universal Asynchronous Receiver/Transmitter |
| <b>IDE</b>  | Integrated Development Environment          |

## CHAPTER 1

### INTRODUCTION

The goal of this project was to build a neural network able to classify which letter of the American Sign Language (ASL) alphabet is being signed, given an image of a signing hand. This project is a first step towards building a possible sign language translator, which can take communications in sign language and translate them into written and oral language. Such a translator would greatly lower the barrier for many deaf and mute individuals to be able to better communicate with others in day-to-day interactions.

This goal is further motivated by the isolation that is felt within the deaf community. Loneliness and depression exist in higher rates among the deaf population, especially when they are immersed in a hearing world [1]. Large barriers that profoundly affect life quality stem from the communication disconnect between the deaf and the hearing. Some examples are information deprivation, limitation of social connections, and difficulty integrating in society.

Most research implementations for this task have used depth maps generated by depth camera and high-resolution images. The objective of this project was to see if neural networks are able to classify signed ASL letters using simple images of hands taken with a personal device such as a laptop webcam. This is in alignment with the motivation as this would make a future implementation of a real time ASL-to-oral/written language translator practical in an everyday situation.

## CHAPTER 2

### LITERATURE SURVEY

1. **"Real-Time American Sign Language Recognition Using Convolutional Neural Networks"** by Thad Starner and Alex Pentland (1998). This pioneering work introduced the application of CNNs for real-time sign language recognition. The study used a glove embedded with sensors to capture hand gestures and CNNs to classify the gestures. Achieved high accuracy in recognizing American Sign Language gestures in real-time.
2. **"Real-Time Hand Gesture Recognition Using Convolutional Neural Networks"** by Zihao Zhu and Guangyao Fu (2018). Proposed a real-time hand gesture recognition system using CNNs. Used depth data from a depth camera to recognize hand gestures. Achieved real-time performance with high accuracy in recognizing dynamic hand gestures.
3. **"Real-Time Hand Gesture Recognition Using Convolutional Neural Networks"** by Dhananjay Borse and Dr. M. M. Raghuvanshi (2019). Developed a real-time hand gesture recognition system using CNNs. Captured hand gestures using a webcam and processed them in real-time. Achieved high accuracy in recognizing static and dynamic hand gestures.
4. **"Real-Time American Sign Language Alphabet Recognition Using Convolutional Neural Networks"** by Prachi Parashar and Mayank Vatsa (2020). Proposed a real-time American Sign Language alphabet recognition system using CNNs. Utilized a dataset of ASL alphabet images and trained a CNN model. Achieved real-time performance with high accuracy in recognizing ASL alphabet gestures.
5. **"Real-Time Sign Language Recognition Using Convolutional Neural Networks"** by Jie Li and Yun Fu (2020). Proposed a real-time sign language recognition system using CNNs. Utilized a large-scale sign language dataset and trained a deep CNN model. Achieved state-of-the-art accuracy in recognizing sign language gestures in real-time.

These studies demonstrate the effectiveness of CNN models in real-time sign language recognition, providing a foundation for the development of accurate and efficient real-time sign language interpreters using computer vision techniques.



## CHAPTER 3

### SYSTEM ARCHITECTURE AND DESIGN

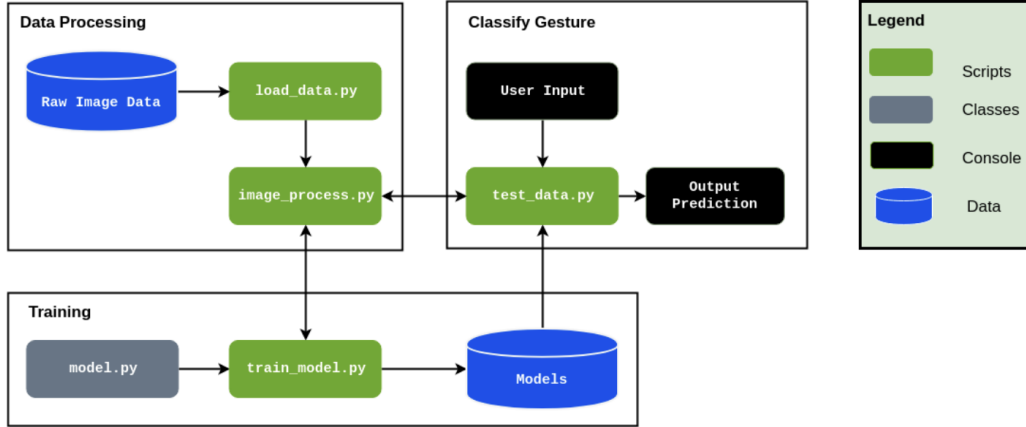


Figure 1: Block Diagram of Software

the project will be structured into 3 distinct functional blocks, Data Processing, Training, Classify Gesture. The block diagram is simplified in detail to abstract some of the minutiae:

- **Data Processing:** The load data.py script contains functions to load the Raw Image Data and save the image data as NumPy arrays into file storage. The process data.py script will load the image data from data.npy and preprocess the image by resizing/rescaling the image, and applying filters and ZCA whitening to enhance features. During training the processed image data was split into training, validation, and testing data and written to storage. Training also involves a load dataset.py script that loads the relevant data split into a Dataset class. For use of the trained model in classifying gestures, an individual image is loaded and processed from the filesystem.
- **Training:** The training loop for the model is contained in train model.py. The model is trained with hyperparameters obtained from a config file that lists the learning rate, batch size, image filtering, and number of epochs. The configuration used to train the model is saved along with the model architecture for future evaluation and tweaking for improved results. Within the training loop, the training and validation datasets are loaded as Data loaders and the model is trained using Adam Optimizer with Cross Entropy Loss. The model is evaluated every epoch on the validation set and the model with best validation accuracy is saved to storage for further evaluation and use. Upon finishing training, the training and validation error and loss is saved to the disk, along with a plot of error and loss over training.
- **Classify Gesture:** After a model has been trained, it can be used to classify a new ASL gesture that is available as a file on the filesystem. The user inputs the file path of the gesture image and the test data.py script will pass the file path to process data.py to load and preprocess the file the same way as the model has been trained.

## CHAPTER 4

### METHODOLOGY

#### Sources of Data:

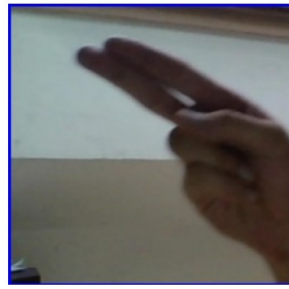
- **Data Collection:** The primary source of data for this project was the compiled dataset of American Sign Language (ASL) called the ASL Alphabet from Kaggle user Akash [3]. The dataset is comprised of 87,000 images which are 200x200 pixels. There are 29 total classes, each with 3000 images, 26 for the letters A-Z and 3 for space, delete and nothing. This data is solely of the user Akash gesturing in ASL, with the images taken from his laptop's webcam. These photos were then cropped, rescaled, and labelled for use.



(a) ASL letter A



(b) ASL letter E



(c) ASL letter H



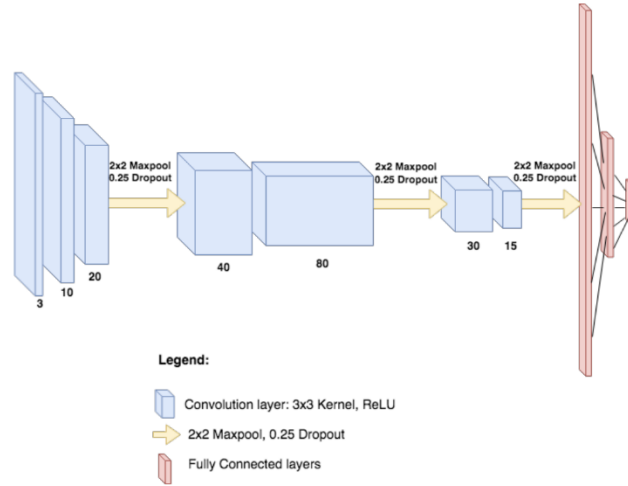
(d) ASL letter Y

A self-generated test set was created in order to investigate the neural network's ability to generalize. Five different test sets of images were taken with a webcam under different lighting conditions, backgrounds, and use of dominant/non-dominant hand. These images were then cropped and preprocessed.

- **Data Pre-processing:** The data preprocessing was done using the PILLOW library, an image processing library, and sklearn.decomposition library, which is useful for its matrix optimization and decomposition functionality.
1. **Image Enhancement:** A combination of brightness, contrast, sharpness, and color enhancement was used on the images. For example, the contrast and brightness were changed such that fingers could be distinguished when the image was very dark.
  2. **Edge Enhancement:** Edge enhancement is an image filtering technique that makes edges more defined. This is achieved by the increase of contrast in a local region of the image that is detected as an edge. This has the effect of making the border of the hand and fingers, versus the background, much clearer and more distinct. This can potentially help the neural network identify the hand and its boundaries.
  3. **Image Whitening:** ZCA, or image whitening, is a technique that uses the singular value decomposition of a matrix. This algorithm decorrelates the data, and removes the redundant, or obvious, information out of the data. This allows for the neural network to look for more complex and sophisticated relationships, and to uncover the underlying structure of the patterns it is being trained on. The covariance matrix of the image is set to identity, and the mean to zero.

## Machine Learning Model

The model used in this classification task is a fairly basic implementation of a Convolutional Neural Network (CNN). As the project requires classification of images, a CNN is the go-to architecture. The basis for our model design came from Using Deep Convolutional Networks for Gesture Recognition in American Sign Language paper that accomplished a similar ASL Gesture Classification task [4]. This model consisted of convolutional blocks containing two 2D Convolutional Layers with ReLU activation, followed by Max Pooling and Dropout layers. These convolutional blocks are repeated 3 times and followed by Fully Connected layers that eventually classify into the required categories. The kernel sizes are maintained at 3 X 3 throughout the model. Our originally proposed model is identical to the one from the aforementioned paper, this model is shown in Figure 5. We omitted the dropout layers on the fully connected layers at first to allow for faster training and to establish a baseline without dropout.

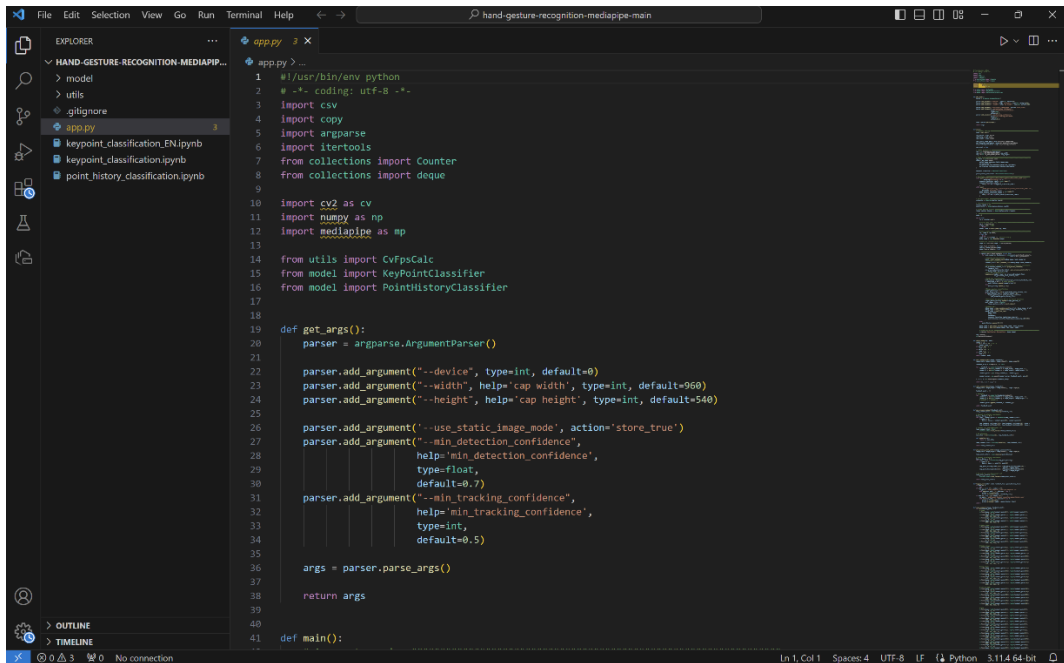


We also decided to design a separate model to compare with the model in the paper. This model was designed to be trained faster and to establish a baseline for problem complexity. This smaller model was built with only one “block” of convolutional layers consisting of two convolutional layers with variable kernel sizes progressing from 5 X 5 to 10 X 10, ReLU activation, and the usual Max Pooling and Dropout. This fed into three fully connected layers which output into the 29 classes of letters. The variation of the kernel sizes was motivated by our dataset including the background, whereas the paper preprocessed their data to remove the background. The design followed the thinking that the first layer with smaller kernel would capture smaller features such as hand outline, finger edges and shadows. The larger kernel hopefully captures combinations of the smaller features like finger crossing, angles, hand location, etc.

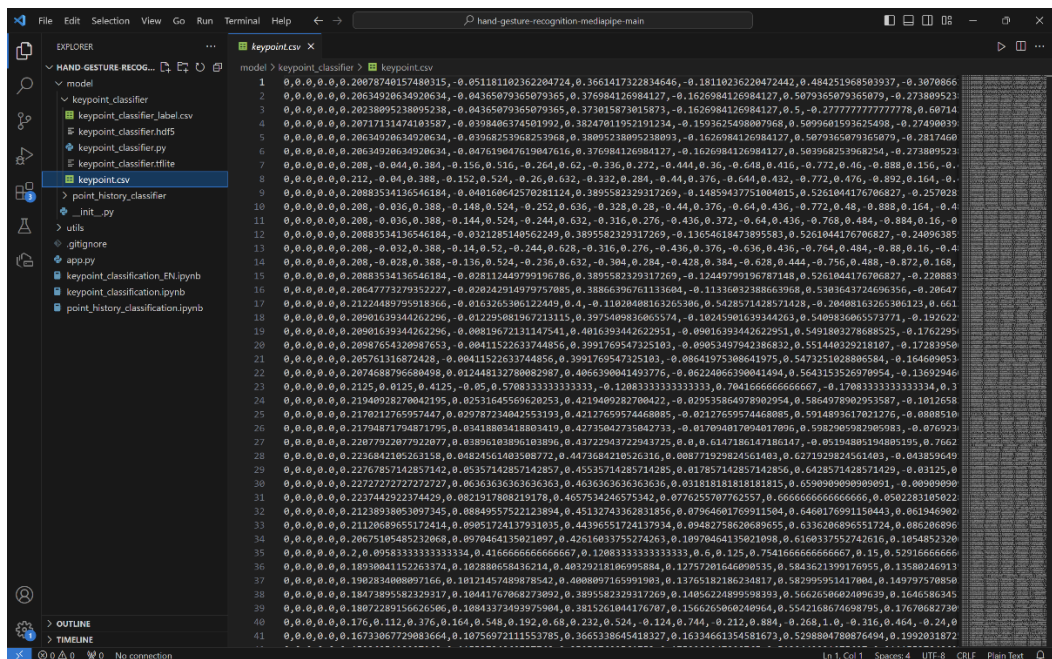
## CHAPTER 5

### CODING AND TESTING

### Source Code:

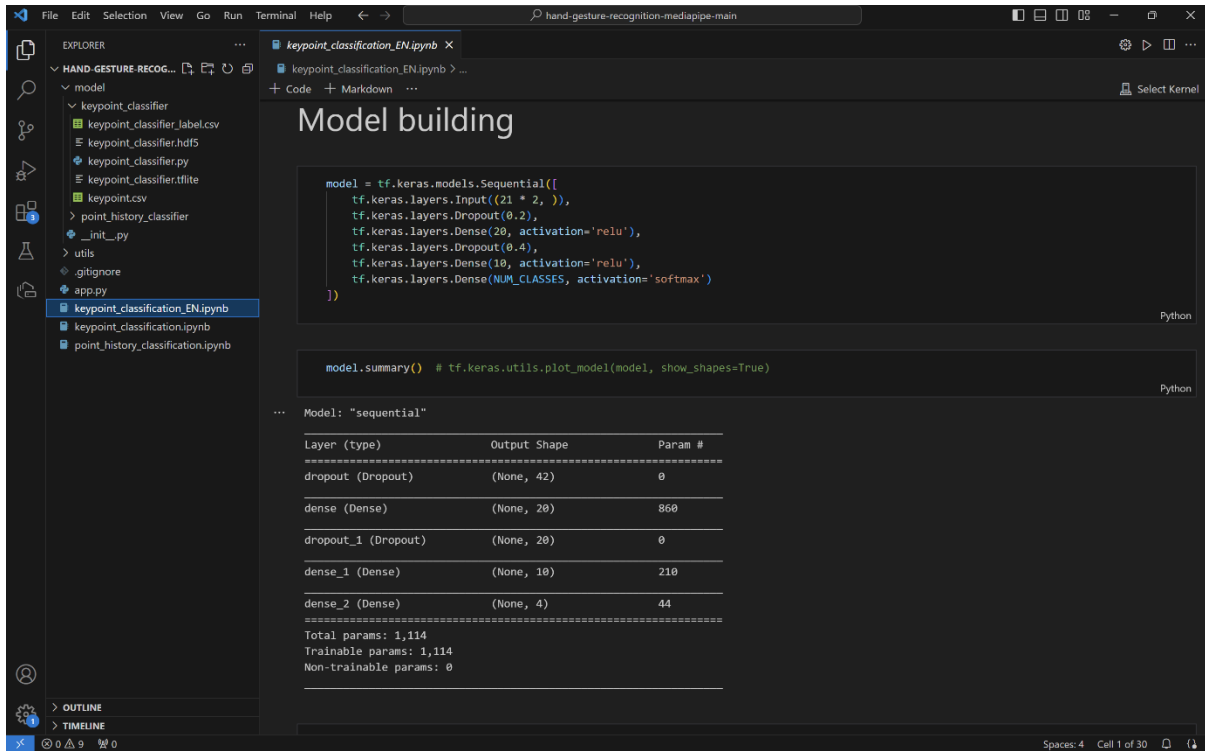


## Dataset:



## CHAPTER 5

### Model Building:



**Model building**

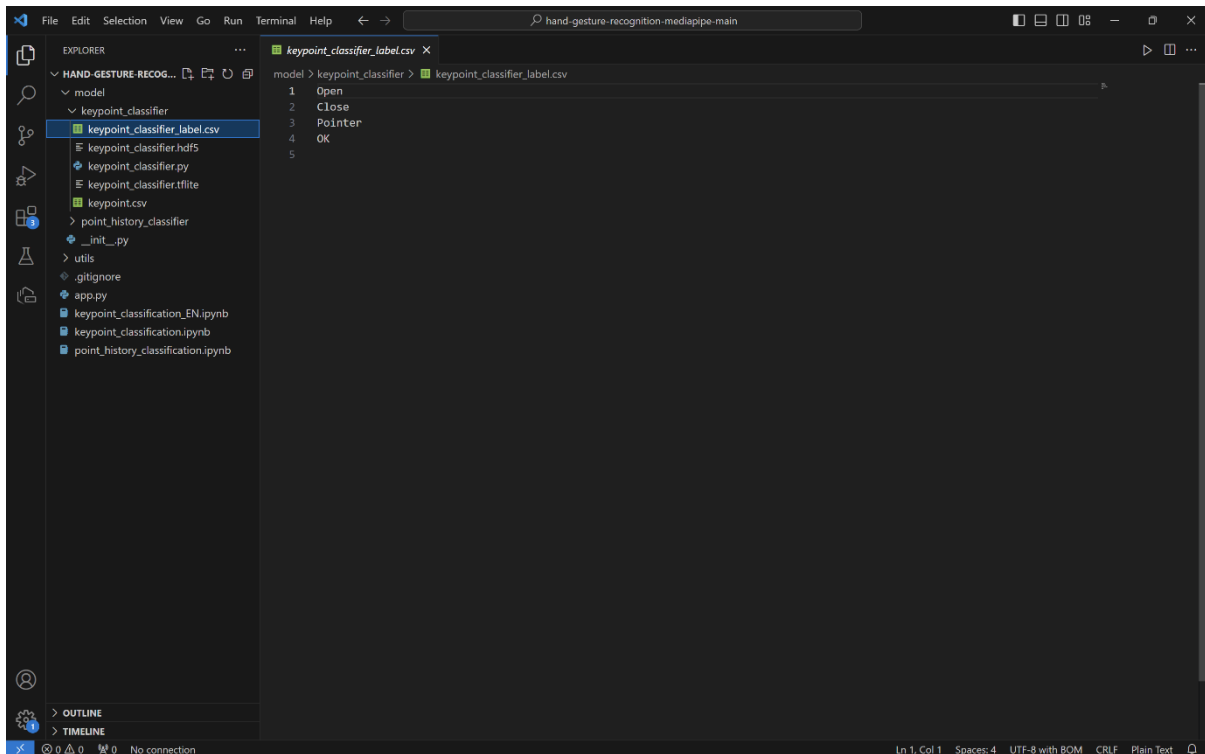
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Input((21 * 2, )),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(20, activation='relu'),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
])
```

```
model.summary() # tf.keras.utils.plot_model(model, show_shapes=True)
```

```
... Model: "sequential"
```

| Layer (type)            | Output Shape | Param # |
|-------------------------|--------------|---------|
| dropout (Dropout)       | (None, 42)   | 0       |
| dense (Dense)           | (None, 20)   | 860     |
| dropout_1 (Dropout)     | (None, 20)   | 0       |
| dense_1 (Dense)         | (None, 10)   | 210     |
| dense_2 (Dense)         | (None, 4)    | 44      |
| Total params: 1,114     |              |         |
| Trainable params: 1,114 |              |         |
| Non-trainable params: 0 |              |         |

### Labels:



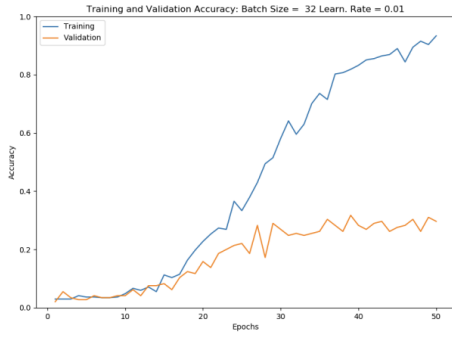
```
model > keypoint_classifier > keypoint_classifier_label.csv
```

```
1 Open
2 Close
3 Pointer
4 OK
5
```

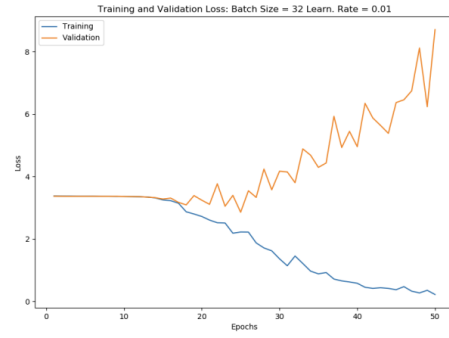
## CHAPTER 5

### Testing:

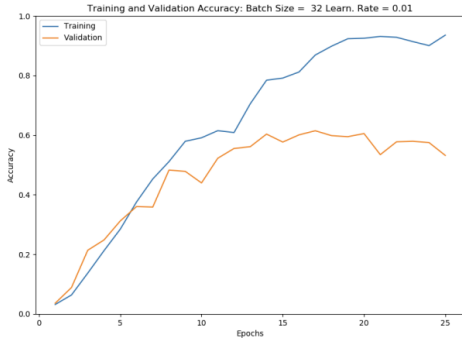
To determine whether our preprocessing of images actually results in a more robust model, we verified on a test set comprised of images from the original dataset, and our own collected image data. The performance of the models on the test set is shown in Table 2. We see that the model trained on preprocessed images performs much better than the model trained on the original images, likely due to the former's lack of overfitting.



(a) Training and Validation Accuracy



(b) Training and Validation Loss



(a) Training and Validation Accuracy



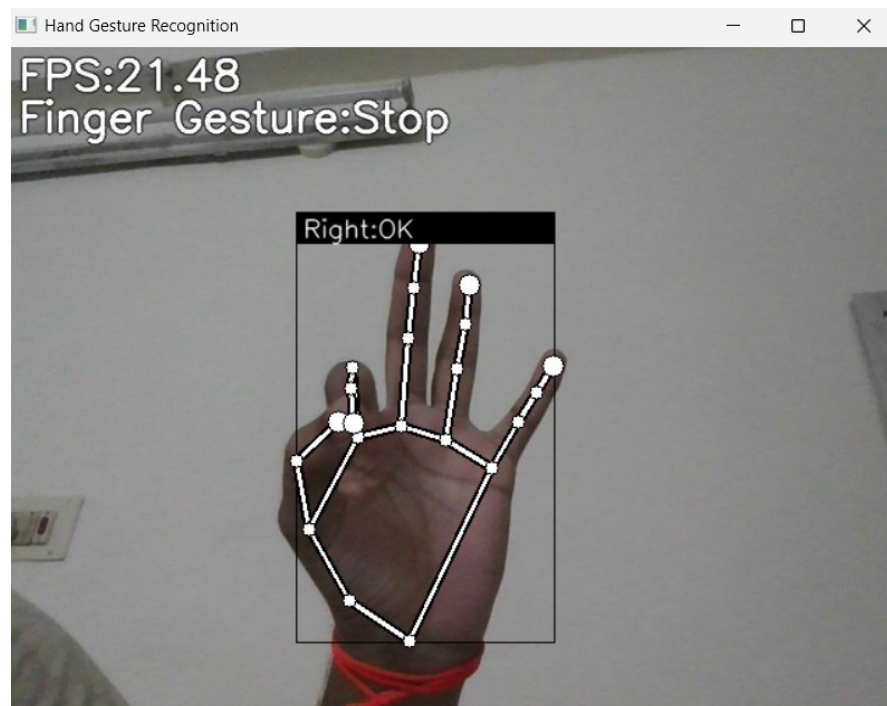
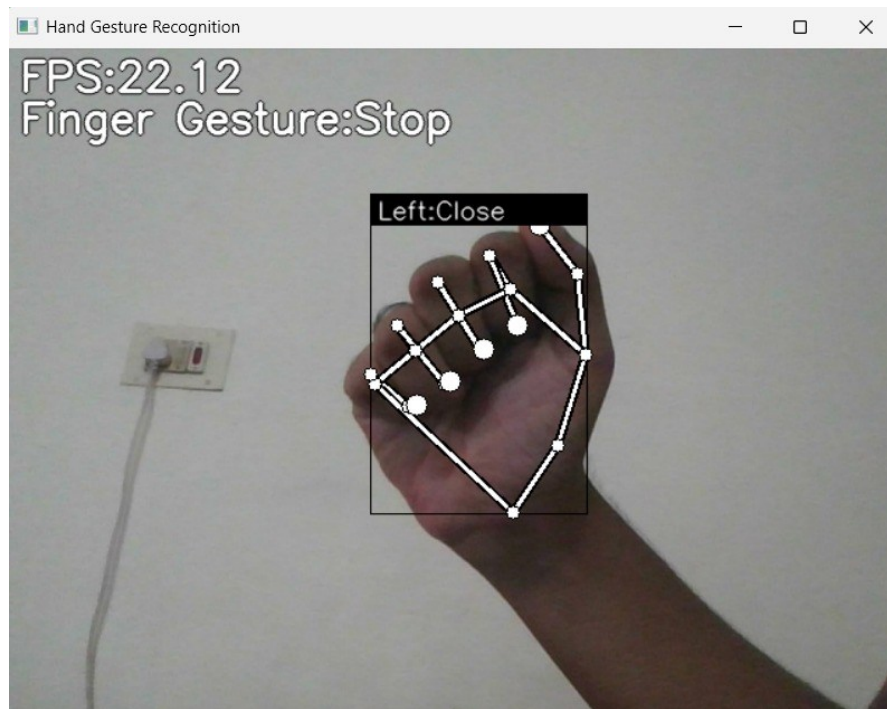
(b) Training and Validation Loss

Figure 8: Training and Validation Performance of Model Trained on Original Images

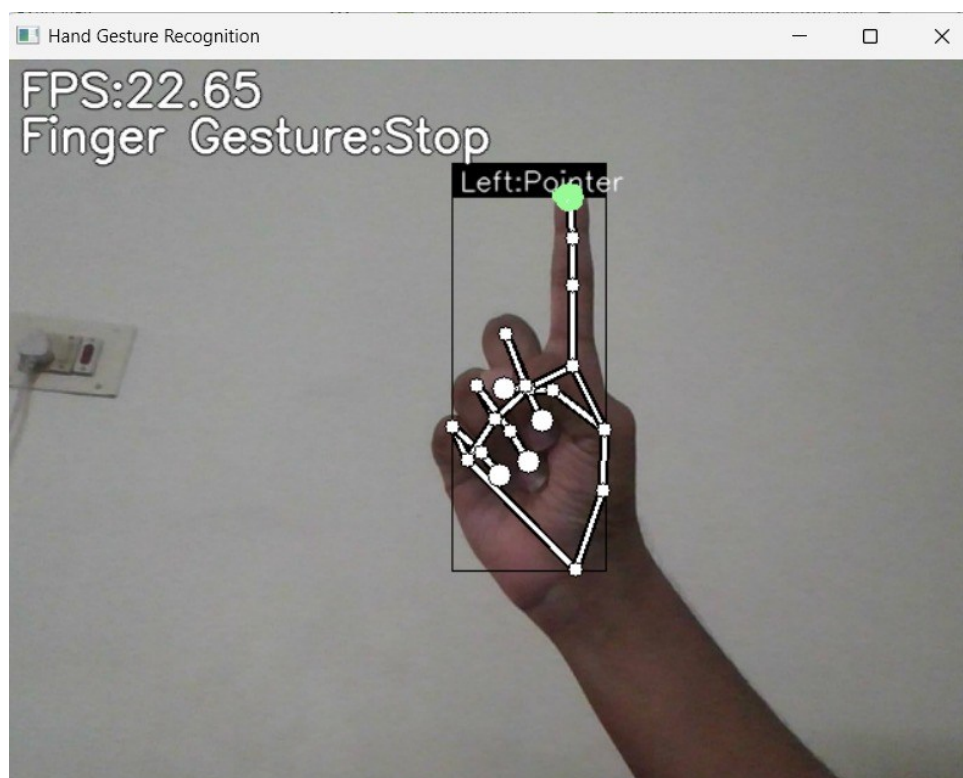
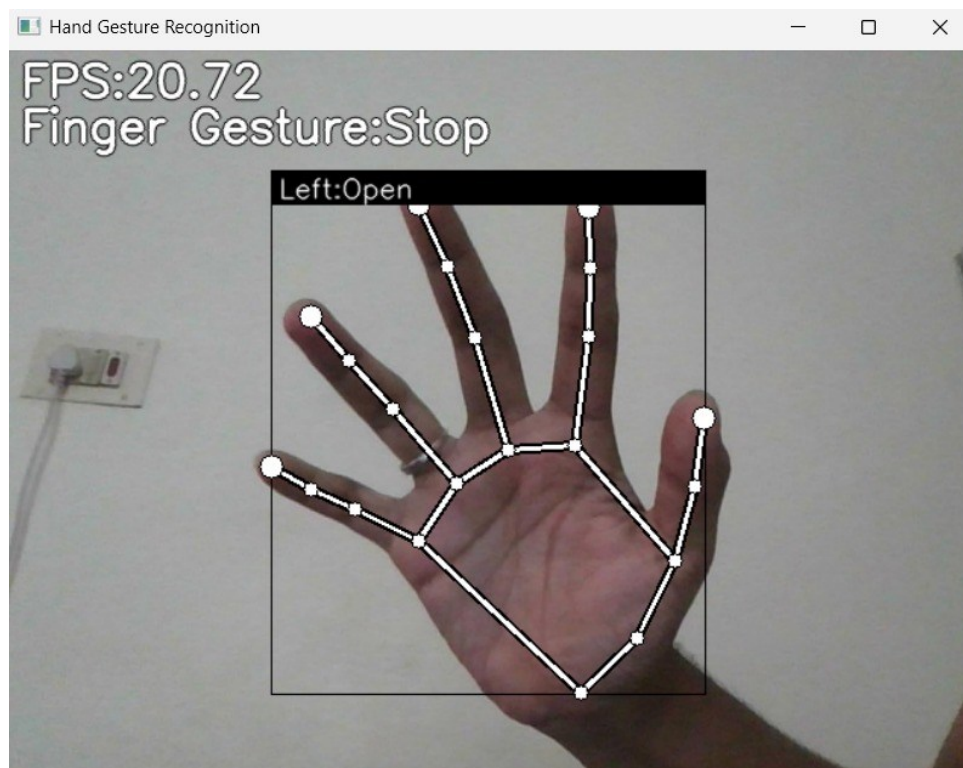


## CHAPTER 6

### SCREENSHOTS AND RESULTS



## CHAPTER 6

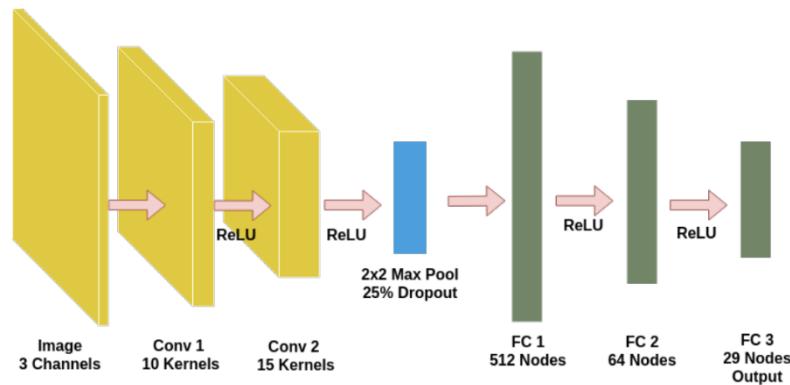




## CHAPTER 6

### Model Performance

- **Training and Validation:** Our models were trained using Adam optimizer and Cross Entropy Loss. Adam optimizer is known for converging quickly in comparison with Stochastic Gradient Descent (SGD), even while using momentum. However, initially Adam would not decrease our loss thus we abandoned it to use SGD. Debugging Adam optimizer after our final

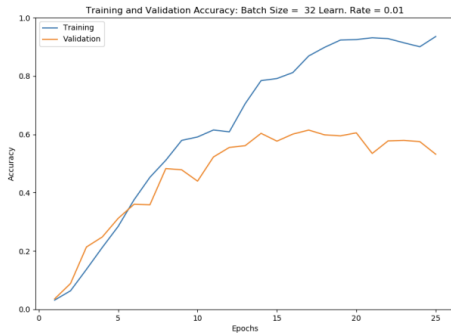


presentation taught us that lowering learning rate significantly can help Adam to converge during training. Thus, allowing us to train more models towards the end of our project. Of our two models, the one based on the paper was shown not to be viable, as it took much longer to train without showing any significant decrease in accuracy or loss. We believe this is likely due to the more difficult nature of our classification with the inclusion of background in the images and the lower resolution, causing training to be more difficult. Thus, we decided to focus on improving our smaller model which initially trained to 40% validation accuracy. Although we had a very large dataset to work with; 3,000 samples for each of 29 classes, after processing the images into NumPy arrays, we found our personal computers could load a maximum of 50-100 samples/class and our Google Cloud server could load 200 samples/class. The need to load small datasets actually led us to test the effect of increasing the data available to our models. On our preliminary model, which used strides of 2 on both layers (instead of the strides of 1 and then 2, on our final model) we found the following relation between samples/class and model accuracy:

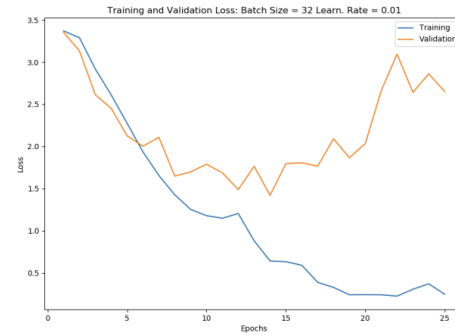
## CHAPTER 6

| Samples/Class | Avg. Validation Accuracy |
|---------------|--------------------------|
| 25            | 27.3%                    |
| 50            | 34.8%                    |
| 100           | 46.1%                    |
| 200           | 58.2%                    |

Training our initial models on less data led to the models quickly overfitting as shown in Figure 7. This is likely due to the small amount of samples to train on leading to bad generalization and learning of the sample space. Increasing the size of our dataset to 200 samples/class led to better model results, with peak validation accuracy of 60.3% in epoch 17. However, taking a look at our loss function, we see that the validation loss is increasing, indicating overfitting of the model. After we applied filtering, enhancement, and ZCA whitening to our dataset, the model performance increased drastically as shown in Figure 9. The peak validation accuracy achieved is 77.25% in epoch 24. As shown by the plot of loss, the validation loss is still decreasing, albeit at a slower rate than the training loss, indicating that the model is not drastically overfitting. This shows that preprocessing our images by applying filters and ZCA whitening helps to enhance relevant features for the model to learn.



(a) Training and Validation Accuracy



(b) Training and Validation Loss

Figure 8: Training and Validation Performance of Model Trained on Original Images

## **CHAPTER 7**

### **CONCLUSION AND FUTURE ENHANCEMENTS**

#### **Key Learnings**

**Train Early and Often:** Due to the long training time of our model, it became cumbersome to test various hyperparameters, architectures, image filtering, etc. In future projects, training earlier can identify such issues and more time can be dedicated to training.

**Experiment With Optimizers:** Due to our initial difficulties with Adam, we abandoned it for the slower SGD. However, further testing with Adam and varying our hyperparameters allowed Adam to converge. The lesson to take away is to try various optimizers early on to establish which ones converge the quickest for faster training.

**Amount of Data:** As shown in Section 5.1, increasing the training set size leads to drastic improvement in model performance. This likely increases the robustness of the model through learning more of the possible sample space.

#### **Advantages**

**Scalability:** AI-driven sign language interpretation can be scaled across various platforms and devices, from mobile apps to web services, making it accessible to a wider audience. This scalability ensures that individuals can access interpretation services whenever and wherever needed.

**Consistency and Reliability:** AI algorithms provide consistent and reliable interpretation, reducing the variability that may occur with human interpreters. This ensures that the message conveyed through sign language remains accurate and clear, fostering better communication outcomes.

**Cost-effectiveness:** By automating the interpretation process, AI reduces the need for hiring human interpreters for every interaction. This cost-effectiveness makes sign language interpretation more financially feasible for organizations and institutions, ultimately increasing accessibility for the Deaf and hard-of-hearing community.

**Continuous Improvement:** AI algorithms can continuously learn and improve over time through machine learning techniques. As more data is collected and analyzed, the accuracy and efficiency of sign language interpretation will likely improve, providing users with an even better experience over time.

## **Future Steps**

## **CHAPTER 7**

**Use Dynamic Loading for Dataset:** Our original dataset was quite large and is impossible to use without a server with a lot of RAM and disk space. A possible solution is to split the file names into training, validation, and test sets and dynamically loading images in the Dataset class. Using such a loading technique would allow us to train the model on more samples in the dataset

**Multimodal Communication:** AI research should explore integrating sign language interpretation with other modes of communication, such as speech recognition and synthesis. This multimodal approach can facilitate more seamless interactions between individuals who use different forms of communication, further bridging the gap between signers and non-signers

# REFERENCES

## 1.Datasets:

RWTH-PHOENIX-Weather 2014T: A large-scale dataset for continuous sign language recognition collected from German Sign Language (DGS) videos.

## 2.Libraries and Tools:

OpenPose: A library for real-time multi-person keypoint detection and pose estimation.

TensorFlow/Keras: Deep learning frameworks that can be used for training and deploying neural networks for sign language recognition.

OpenCV: <https://opencv.org/>

<https://github.com/kinivi/hand-gesture-recognition-mediapipe>