BSCS-S23-009

03-134191-024 Usama Yousaf

Sign Language Recognition System

In partial fulfilment of the requirements for the degree of

Bachelor of Science in Computer Science

Supervisor: Umar Nasir

Department of Computer Sciences

Bahria University, Lahore Campus

January 2024

C e r t i f i c a t e

We accept the work contained in the report titled.

"Sign Language Recognition System"

written by

Usama Yousaf

as a confirmation to the required standard for the partial fulfilment of the degree of Bachelor of Science in Computer Science.

Approved by:

Supervisor: Umar Nasir

_____ (Signature)

Jan 10, 2024

DECLARATION

We hereby declare that this project report is based on our original work except for citations and quotations which have been duly acknowledged. We also declare that it has not been previously and concurrently submitted for any other degree or award at Bahria University or other institutions.

Enrolment	Name	Signature
03-134191-024	Usama Yousaf	

Date : Jan 10, 2024

Specially dedicated to

My Mother, My Father and My Teacher who helped me in this.

Usama Yousaf.

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Miss Umar Nasir for her invaluable advice, guidance, and his/her enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my loving parent and friends who had helped and given me encouragement.

Usama Yousaf

ABSTRACT

This project features a groundbreaking Sign Language Recognition System that combines OpenCV's image processing capabilities with the analytical prowess of Convolutional Neural Networks (CNN). Destined to become an important step forward in assistive technology, especially for the deaf and hard-of-hearing community. This system interprets even subtle sign language movements as text or speech.

The system is a real-time video process one. OpenCV plays an especially important role in capturing and enhancing sign language gestures. Each gesture, captured through a video stream. Every signal is carefully edited to enhance the clarity and detail before inputting it into an analysis platform. This is an extremely important pre-processing stage. The quality of the subsequent recognition process depends entirely on how well it does its job in eliminating complex hand motions and expressions.

The analytical core of the system is a trained CNN model, which has undergone intensive training on an extensive database. It contains many complex sign language gestures, from delicate distinctions to quite complicated combinations. Using such a rich dataset, the CNN model acquires an advanced understanding. It can discern and understand every type of sign with flawless accuracy.

One of the notable characteristics of this system is its excellent real-time interpretation. Precision Despite this, it is not only a technical feat. On the symbolic level too, its high degree of precision reveals overcoming obstacles to communication and creating new openings for interchange.

Also, since the system can recognize sign language and convert it into text or speech for all to read and hear, it becomes a truly comprehensive tool. Especially in educational, social, and professional settings this feature can really enhance communication and interaction.

The combination of OpenCV and CNN in this work goes beyond the mere technical aspect-it is a thoughtful melting together designed to add value. Pushing the limits of machine learning and computer vision, this system represents what technology can do for society, creating a more equal and integrated world.

TABLE OF CONTENTS

Table of Contents

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

CNN	Convolutional Neural Network
OpenCV	Open-source computer vision

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background

The deaf and hard of hearing community use sign language as a visual form of communication. It comprises a complicated code of gestures by which they can communicate information without speaking. The fact that sign language is a vital means of contact for several people and groups makes it difficult for those unfamiliar with it to interact appropriately with the hearing-impaired.

To tackle this problem, many automated sign language recognition systems based on computer vision as well as machine learning have been developed. This project will be dedicated to creating a sign language recognition system based on Python and OpenCV. In this way, it will detect and identify signs and hand gestures which will enable the deaf people to interact effectively with hearing people.

This involves generating an extensive dictionary of sign gestures and signs, pre-processing, and feature extraction of input pictures, building the CNN model for training, and testing and comparing the efficiency of the trained model. A possible use for this system is as a reliable tool that helps one communicate using gestures when he or she is unable to speak or hear.

1.2 Problem Statements

This project addresses a problem, that is about communication gap between people who hear well but does not understand their signed language and those who speak or sign but has hearing loss. Communication through signing is a widely accepted form of communication by both hearing and deaf persons as well as the larger community.

However, it may pose challenges for other hearing individuals that do not understand signing since they will be unable to engage in active interaction with them.

This problem can be addressed through the construction of Sign Language Recognition system in Python and OpenCV. It will, therefore, help the non-hearing people communicate with those who are hearing by recognizing and identifying the hand gestures and signs in real time. One major problem with designing this system is that sign language is very complicated, and it consists of thousands of different hand gestures, facial expressions, and shapes whose movements are unique.

To surpass this hurdle, we will create a comprehensive database for sign language gestures/signs using image processing and machine learning approaches on input pictures. Subsequently, we shall train of a CNN Model with the data and test our models using real time sign language recognition.

1.3 Aims and Objectives

The objectives of the Sign Language Recognition project using Python and OpenCV are:

• Develop a comprehensive database of sign language gestures and signs: The software will build a database containing different variants of both sign language hand gestures and signs.

• Pre-process and extract features from input images: Image processing techniques will be used to process input images, extracting features relevant for training machine learning model.

• Train a Convolutional Neural Network (CNN) model: Therefore, pre-processed data will be used to train a CNN model for live recognition of the sign language gesture and signs.

• Test and evaluate the accuracy of the trained model: As to this point, the completed model will be experimentally tested in live sign language recognition to determine whether it is valid and usable.

• Develop a user interface for the system: In addition, the system interface will be designed to provide an easy way for people to interact with it.

• Optimize the system for real-time performance: One is intended to ensure minimal delay in recognition and response to the movements and sign.

1.4 Scope of Project

This project will focus on developing a real time Sign Language Recogination in programming languages such as python or Javascript. This system will be able to detect, identify and communicate all kinds of sign language, enabling two parties – one hearing and the other non-hearing – to interact effectively.

To achieve this, the project will involve the following steps:

- An all-inclusive database of sign language gestures and signs.
- Image processing: preprocessing & feature extraction of input images
- The learned features can be used for training of CNN model.
- Real-time sign language recognition with the trained model tested and evaluated for its accuracy.
- Making a system-user interface for easy interactions between the system and users.

This project aims to create an interactive and effective web-based system or platform for learning about signs used by people who have speech and hearing disabilities so they may communicate better. This system will be able to pick- up a wide spectrum of signs used in sign language so it will make meaningful communication in different environments possible.

CHAPTER 2

LITERATURE REVIEW

2.1 Product Viewpoint

In developing the Gesture Guru Sign Language recogniser, special attention should be paid to the specific needs of the Deaf and Hard of Hearing people. It should convert the sign language gestures into text without any problem. Moreover, it comes with user authentications for users to register their accounts thereby improving their experience. You may find it essential to have social log in options so that the process of on boarding can be simplified. Such a content management system is essential when one aims at handling many sign language gestures. The system should also be configured for real-time gesture detection. Be highly conscious of accuracy when confronted by such languages' intricate grammatical constructions and dialects. Issues like different forms of the dialects of sign-language and recognition of signals of persons from them should be brought up, too. For testing and validation, it would be wise to incorporate sign language data sets and benchmarking tools in order to have an efficient system. The final idea is to design an instrument that promotes openness among members of the special community with treated conditions.

2.2 Operating Environment

To properly run the website some of the basic operating environments are necessary which are as follow:

Model Training:

- Python (OpenCV, Tensor Flow)

Development Essentials:

- Visual Studio Code
- Google Colab
- Jupyter Notebook

2.3 Design and Implementation

The Sign Language Recognition System is based on OpenCV and Convolutional Neural Networks (CNN). It recognizes sign language, creatively designed to be trained from a manually created dataset. Secondly, a specially chosen group of sign language gestures should be carefully collated and arranged to ensure variety. Preprocessing image collection using OpenCV, resizing and normalization. They are then amplified, so that they can be used as efficient training resources for machine learning. At the core of this system is a specially designed CNN model that learns from there custom dataset. There are also convolutional layers which can extract and learn the special features contained in sign language gestures. Once trained, the CNN does a very good job in discriminating and classifying different signs. This also massively powers up its accuracy at the same time as attuning it to extremely fine subtle distinctions in all aspects of translation from sign language into more universal formats giving individuals with hearing and speech impairments an important step forward practically for their communications.

2.4 Literature Review Table

Table 2.1: Literature Review Table

No	Year	Author(s)	Title	Publications	Methodology	Key findings
1	2021	Lakhtoiya et al. [1]	Real Time Sign Language Recognition Using Image Classification	2021 2nd International Conference for Emerging Technology (INCET)	Real-time image classification	Focused on real-time recognition of sign language for effective communication
2	2023	Urs et al. [2]	Action Detection for Sign Language Using Machine Learning	2023 International Conference on Network, Multimedia, and Information Technology (NMITCON)	Machine learning for action detection	Emphasized on detecting actions in sign language for accurate interpretation
3	2023	Seema, Singla [3]	A Comprehensive Review of CNN-Based Sign Language Translation System	Proceedings of Data Analytics and Management: ICDAM 2022	Review of CNN applications	Comprehensive review of the use of CNNs in translating sign language
4	2023	Sudha et al. [4]	SIGN LANGUAGE TRANSLATION USING	SIGN	Machine learning techniques	Focus on translating sign language using various

			MACHINE LEARNING			machine learning approaches
5	2023	Kozhamkulova et al. [5]	Two-Dimensional Deep CNN Model for Vision-based Fingerspelling Recognition System	International Journal of Advanced Computer Science and Applications	2D Deep CNN model	Developed a model for recognizing fingerspelling in sign language
6	2023	Srininvas et al. [6]	A framework to recognize the sign language system for deaf and dumb using mining techniques	Indonesian Journal of Electrical Engineering and Computer Science	Data mining techniques	Proposed a framework to recognize sign language using data mining
7	2023	Gupta et al. [7]	Hand-Sign to Text Using CNN	Journal of Pharmaceutical Negative Results	CNN for hand-sign to text conversion	Explored the conversion of hand signs to text using CNN models
8	2023	Kumar, Raajkumar [8]	Systematic Literature Survey on Sign Language Recognition Systems	Deep Learning Research Applications for Natural Language Processing	Literature survey	Surveyed various sign language recognition systems, highlighting advancements and challenges

CHAPTER 3

DESIGN AND METHODOLOGY

3.1 Methodology

Collecting and pre-processing different sets of data, coming up with a CNN architecture containing convolutional layers, pooling, fully connected layers, setting of appropriate activation functions, and optimization. The iteration process continues until the model gives acceptable output, which is evaluated based on validation and test datasets. And for this reason I use FDD(Feature Driven Development) for the planning and for building the product according to requirements.

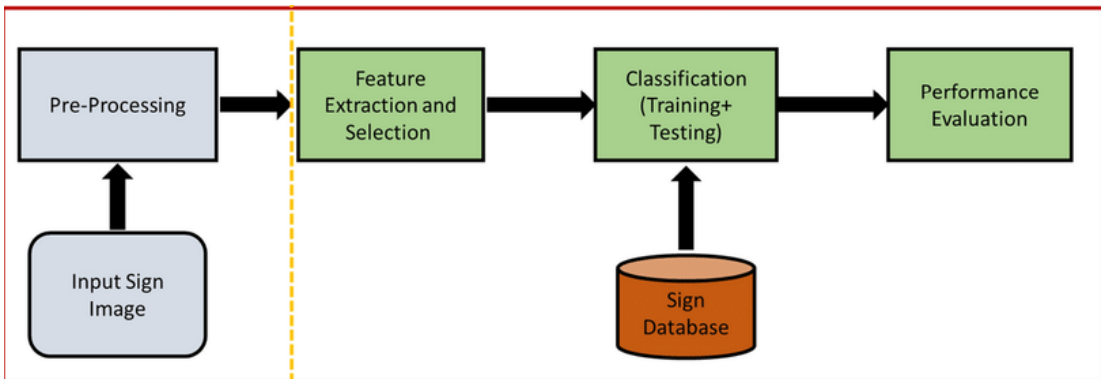


Figure 3.1: Methodology Diagram

3.1.1 CNN Model Structure

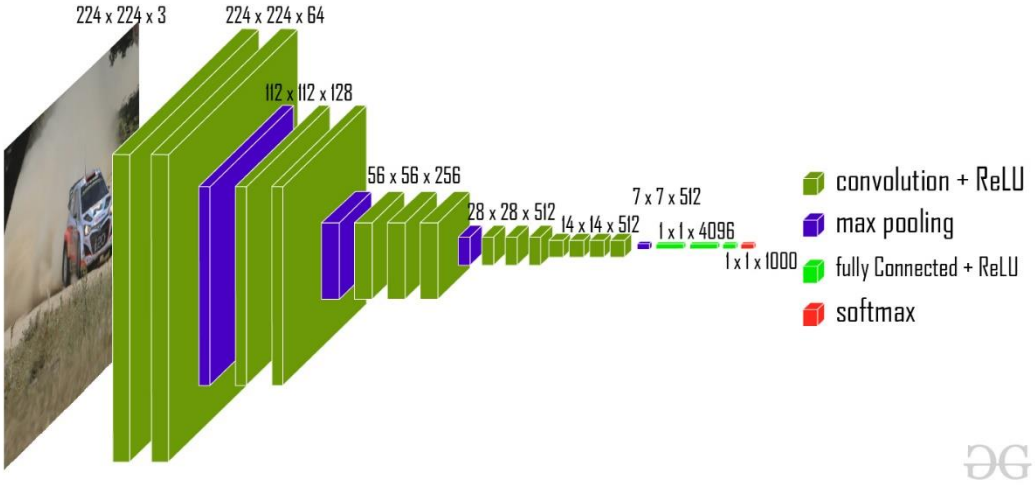


Figure 3.2: CNN Model

CNN uses convolutional layers that can extract hierarchical representations from input data. These layers use learnable filters to cross the input data, extracting spatial hierarchies and local features. These pooling layers, though, narrow the spatial dimensions leading to reduced computational costs. Finally, they are flattened before being pushed into fully connected layers for classifying or regressing. Backpropagation is used to adjust the network's weights towards a minimum loss. The effectiveness of CNNs lies on auto learning ability and representation of class information on a hierarchical basis.

CHAPTER 4

DATA AND EXPERIMENTS AND IMPLMENTATION

4.1 Implementation

The first step in Sign Language Recognition is building a Convolutional Neural Network (CNN) optimized for gesture recognition. The first step involves gathering data from all sources, including different categories of sign language. A neural network built with convolutional layers is good at detecting and eliminating features, as well as cutting down on the spatial dimensions of images. To strengthen the robustness of the model, data is normalized and augmented before being used in training. Once the training phase (in which parameters are adjusted for acceptable performance) is completed, a CNN model can be used in actual applications. Based on the actual requirements of a given project, it can be integrated into different systems. This is because the architecture of the CNN makes it possible to recognize sign language gestures accurately and quickly, so that this becomes an important way for many who suffer from hearing impairment or defects in speech. Its ability to process and recognize gestures in real-time makes the system more practical and effective for various applications.

4.1.1 Python (for deep learning)

It is also known as a versatile, high level yet very readable and easy-to-use programming language. Object-Oriented, Imperative and Functional programming paradigms that make it fit for almost all kinds of application. This explains why python is popularly used in tasks such as web development, data analytics, artificial intelligence, among others because it has an expansive standard library and massive ecosystem of third-party packages. Quickly developing cycles encourage a powerful and flexible form of programming, having a nature that is easy to interpretate.

4.1.1.1 Tensor Flow (python libraries)

The open-source machine learning framework, a development from the google. It enables one to develop and train deep learning models by providing a detailed set of tools for constructing neural networks and running them on several devices. Tensor Flow is compatible with processors that use either CPUs or GPUs, making it faster at running heavy-duty machine learning processes. The framework's high-level APIs such as Keras make model building simpler and, its flexibility accommodates advanced users for custom implementations.

4.1.1.2 OpenCV (ML library)

It's a free library for application development in various fields of computer vision and machine learning, called OpenCV (Open-Source Computer Vision Library). It serves as a powerful set of tools and algorithms widely applied as part of tasks like face recognition, object detection or processing and manipulations with image and videos. OpenCV can be used with bindings for different programming languages such as Python and C++ and is commonly employed in academic studies, industrial applications as well as amateur projects. It has a modular form and comprehensive documentation that allows not only novices but also advanced developers.

4.1.1.3 Scikit learn.

One of these libraries that has been utilized in this project is Scikit-learn which is essentially a machine learning library designed for Python to make data cleaning and model building fast and simple. It comprises numerous methods suited for job classifications, regressions, grouping, and reducing the number of dimensions. Scikit-learn has a simple and compatible API that makes it straightforward even for new users and rapid application creation process. It has data transformation, model assessment and selection features that can cater to learners as well as experts on Machine Learning.

4.2 Data and Experiment

As mentioned above, In the project we are using deep learning models and now we are discussing their work and the outcomes which we take from the model.

4.2.1 Preparation of Datasets

Training of model in CNN first we need to create the data set on which we are going to train. In this Model we capture 600 images for each sign image for the training. During the creation of dataset the code will take Gesture ID and Gesture name from user.

```
Enter gesture no.: 52
Enter gesture name/text: Ok
g_id already exists. Want to change the record? (y/n): y
```

Figure 4.1: Preparation of dataset

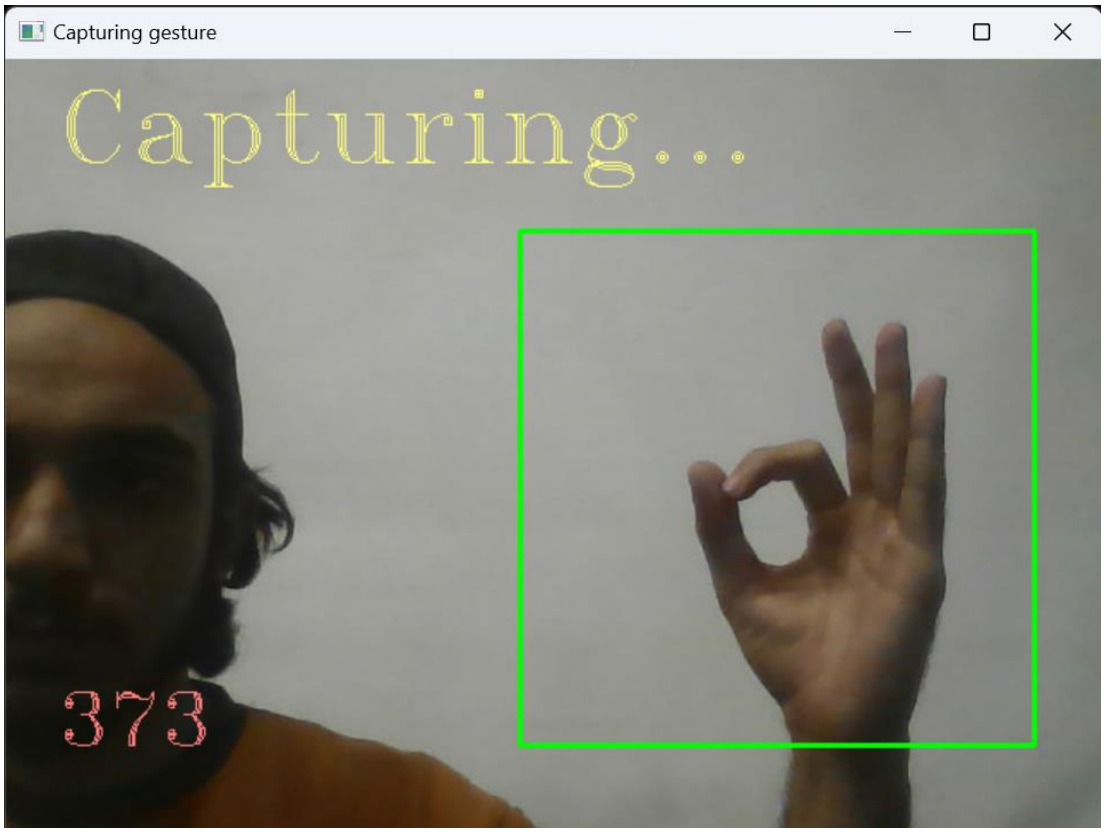


Figure 4.1.1: Storing images.

4.2.2 Processing on data

After the creation of the dataset, I process my data and set it according to my model requirements. Further, I used flipping technique to rotate images for better training of model and then I split my data into training and testing phases. From this the data split into test images, train images, test labels and train labels.

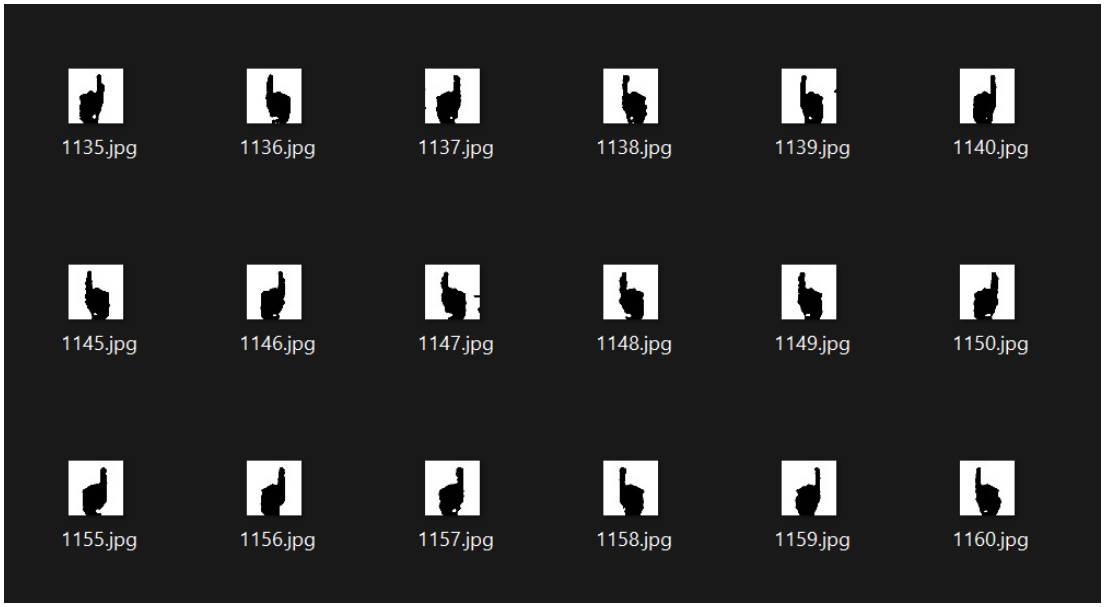


Figure 4.2: Flipping or rotating of images

```
if images_labels:
    images_labels = shuffle(images_labels)
    images, labels = zip(*images_labels)
    print("Length of images_labels", len(images_labels))

    train_images = images[:int(5/6*len(images))]
    print("Length of train_images", len(train_images))
    with open("train_images", "wb") as f:
        pickle.dump(train_images, f)
    del train_images

    train_labels = labels[:int(5/6*len(labels))]
    print("Length of train_labels", len(train_labels))
    with open("train_labels", "wb") as f:
        pickle.dump(train_labels, f)
    del train_labels
```

Figure 4.3: Splitting of data set in training and testing

4.2.3 CNN model Training (70-30):

In this model I set the test size 0.3 which is 30 percent and training size is 0.7 which is 70 percent of the dataset. The reason for splitting data set into 70-30 ratio is to check the model accuracy more wisely. When all these steps are done then we have to set the layers of the model and their activation method. The activation model we set in these layers is “relu” and “softmax”. The results of 70-30 ratio is given as follow:

```
Length of images_labels 61200
Length of train_images 42840
Length of train_labels 42840
Length of test_images 13260
Length of test_labels 13260
Length of val_images 5100
Length of val_labels 5100
```

Figure 4.4: Splitting of dataset (70-30)

4.2.4 Split dataset Training graph (70-30):

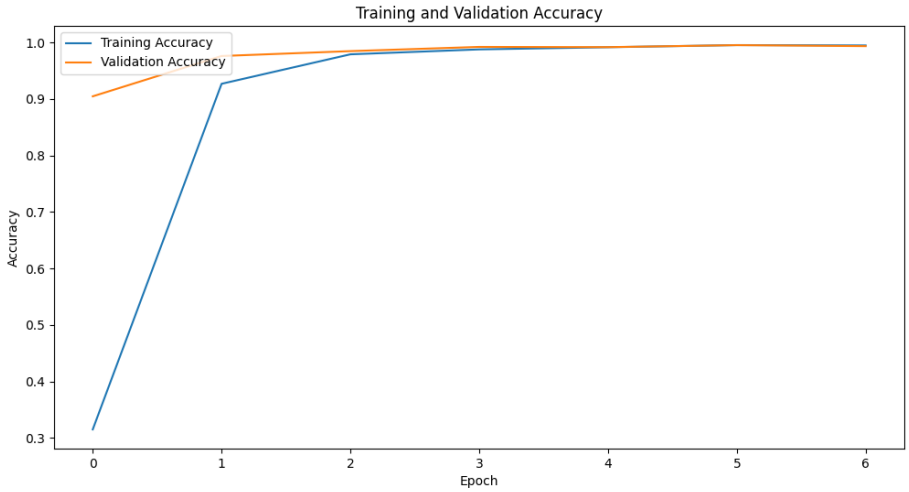


Figure 4.5: Training Graph (70-30)

4.2.5 Confusion Metrics:

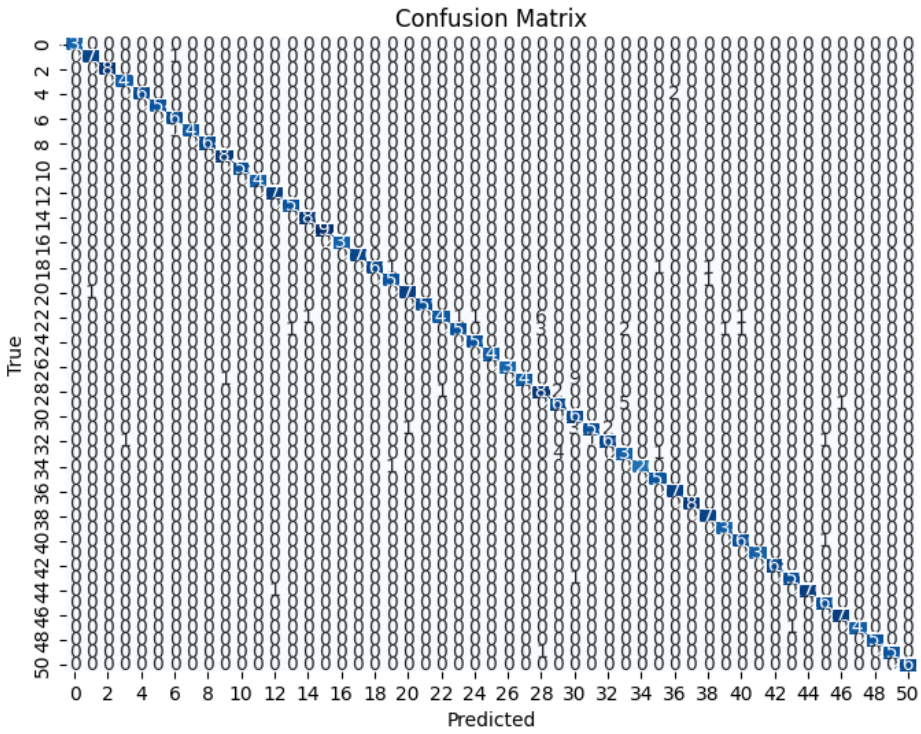


Figure 4.6: Confusion Metrics (70-30)

4.2.6 confusion Metrics Bar plot:

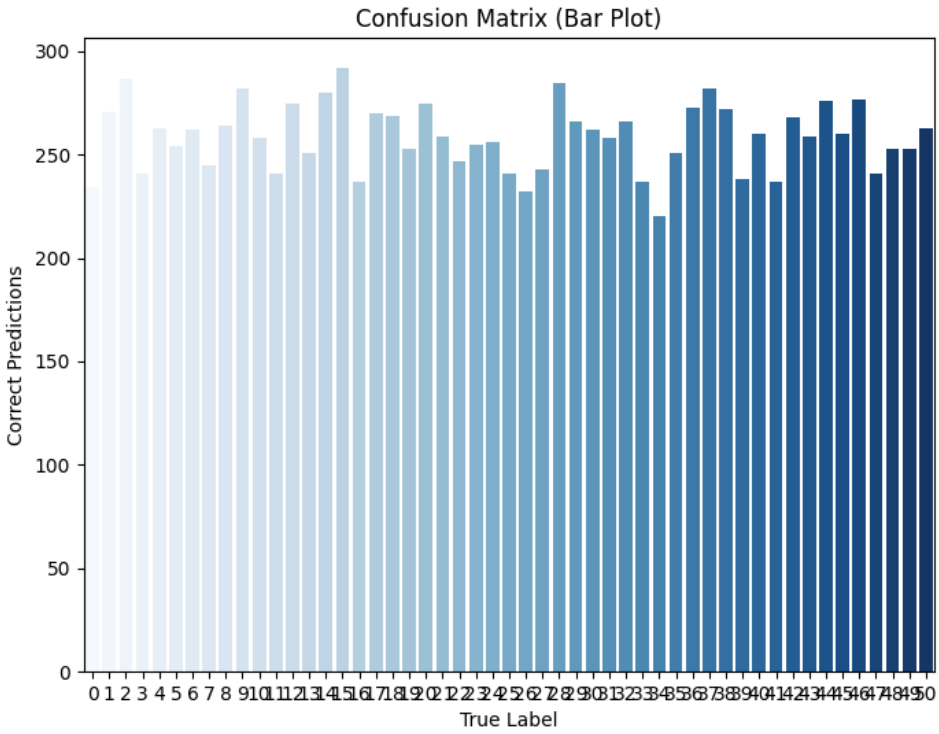


figure 4.7: Confusion Metrics Bar plot (70-30)

4.2.7 AUC and ROC:

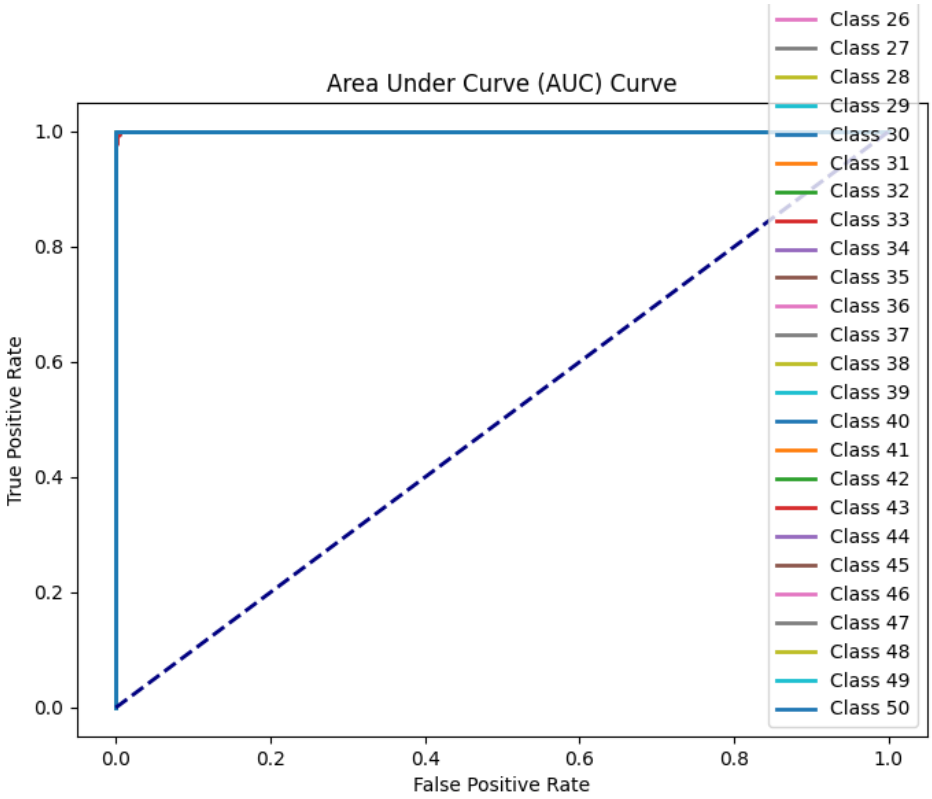


figure 4.8: AUC and ROC (70-30)

4.2.8 Training of CNN Model (60-40):

In this model I set the test size 0.3 which is 30 percent and training size is 0.7 which is 70 percent of the dataset. The reason for splitting data set into 70-30 ratio is to check the model accuracy more wisely. When all these steps are done then we have to set the layers of the model and their activation method. The activation model we set in these layers is “relu” and “softmax”. The results of 70-30 ratio is given as follow:

```
Length of images_labels 61200
Length of train_images 36720
Length of train_labels 36720
Length of test_images 19380
Length of test_labels 19380
Length of val_images 5100
Length of val_labels 5100
```

Figure 4.9: Split dataset (60-40)

4.2.9 Split dataset training graph (60-40):

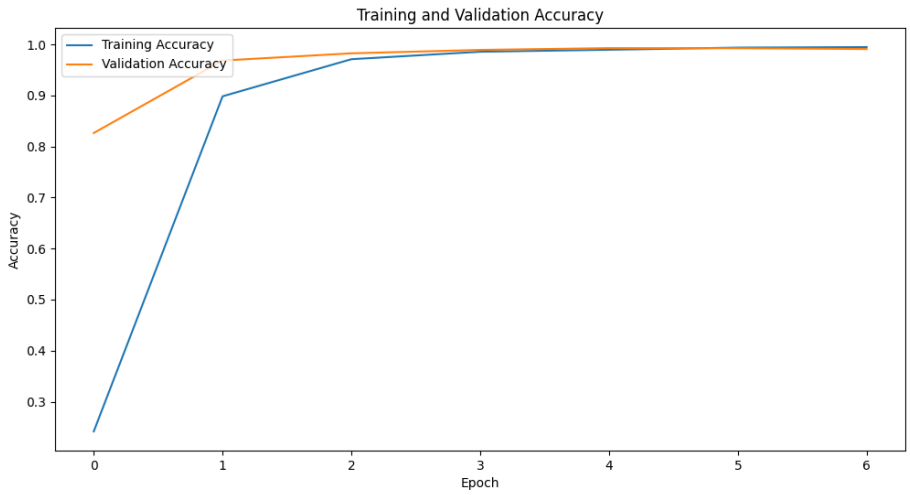


Figure 4.10: Training Graph (60-40)

Confusion Metrics:

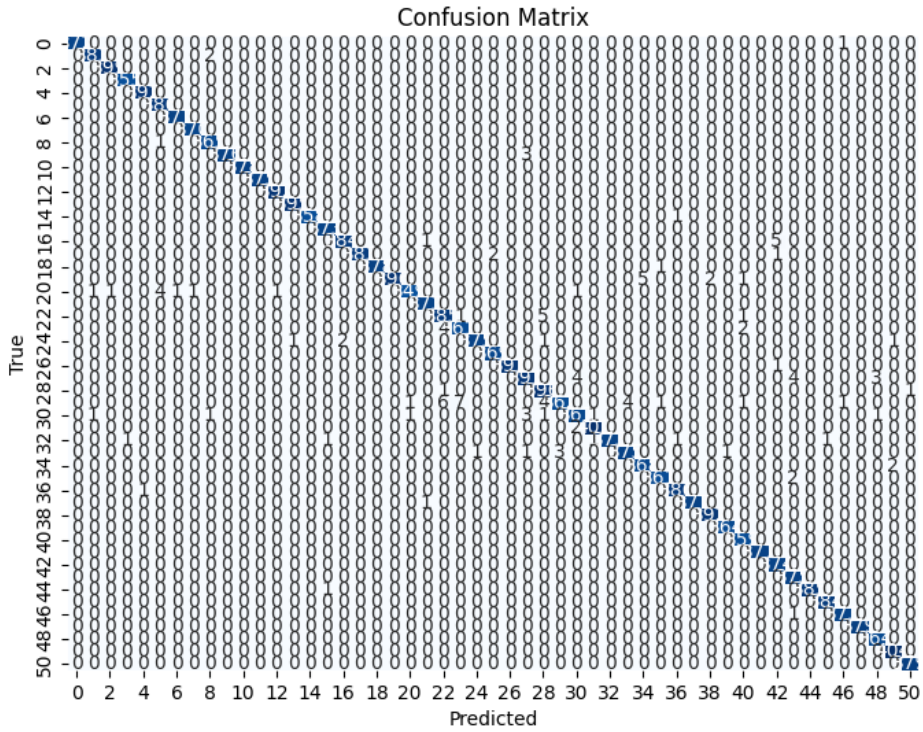


Figure 4.11: Confusion Metrics (60-40)

4.2.10 Confusion Metrics Bar Plot (60-40):

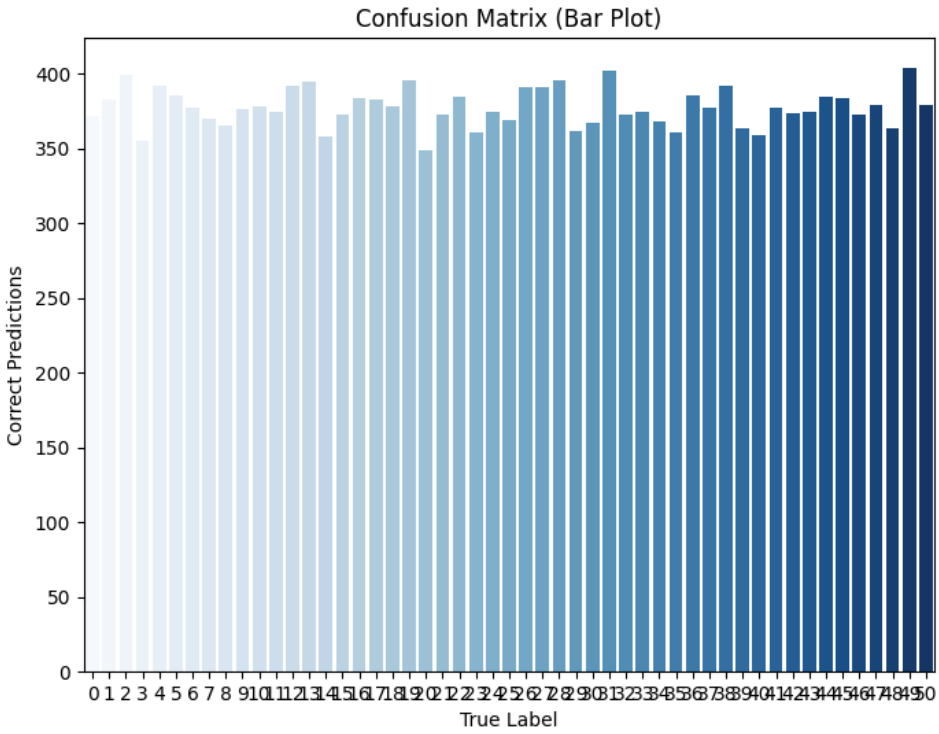


Figure 4.12: Confusion Metrics Bar Plot (60-40)

4.2.11 AUC and ROC:

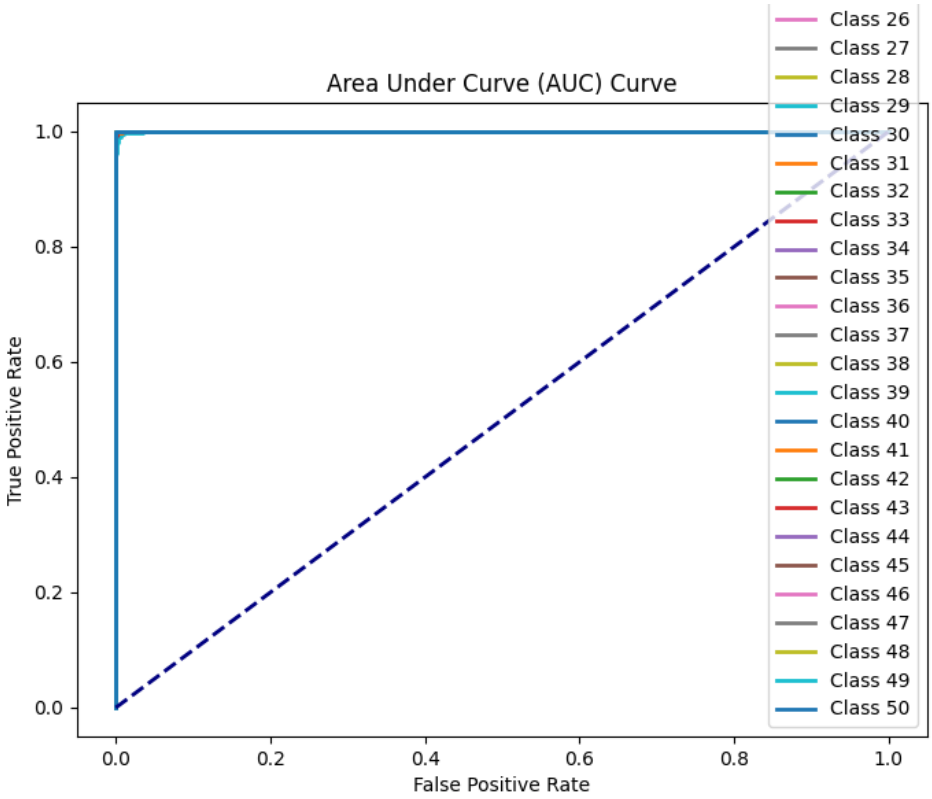


Figure 4.13: AUC and ROC (60-40)

4.2.12 CNN Model Training (80-20)

In this model I set the test size 0.2 which is 20 percent and training size is 0.8 which is 80 percent of the dataset. When all these steps are done then we have to set the layers of the model and their activation method. The activation model we set in these layers is “relu” and “softmax”.

```
def cnn_model():
    num_of_classes = 51
    num_of_classes = get_num_of_classes()
    model = Sequential()
    model.add(Conv2D(16, (2, 2), input_shape=(image_x, image_y, 1), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=(3, 3), padding='same'))
    model.add(Conv2D(64, (5, 5), activation='relu'))
    model.add(MaxPooling2D(pool_size=(5, 5), strides=(5, 5), padding='same'))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(num_of_classes, activation='softmax'))
```

Figure 4.14: CNN model Layers

4.2.13 Execution of model

In the end of the training model is the execution of the model. In the model set the value of epoch is 7 which means my model is train in 7 steps. And in each steps, the accuracy of my model is getting better. This means our model is learning and classify things better.

```
51
(5100, 51)
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 49, 49, 64)        320

 max_pooling2d (MaxPooling2  (None, 25, 25, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 23, 23, 128)       73856

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 128)         0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 256)         819456

 max_pooling2d_2 (MaxPoolin  (None, 1, 1, 256)         0
 g2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 512)               131584

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 51)                26163

=================================================================
Total params: 1051379 (4.01 MB)
Trainable params: 1051379 (4.01 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 4.15: Execution of CNN Model

4.2.14 Training of CNN Model

The following figure show the model of the training and tells how model is getting better. And the following figure shows the growth of the model after every epoch.

```
 best model only with val_acc available, skipping.
102/102 [==============================] - 91s 872ms/step - loss: 6.4268 - accuracy: 0.1489 - val_loss: 1.0559
- val_accuracy: 0.7090
Epoch 2/7
102/102 [==============================] - ETA: 0s - loss: 0.4657 - accuracy: 0.8553WARNING:tensorflow:Can save
 best model only with val_acc available, skipping.
102/102 [==============================] - 95s 927ms/step - loss: 0.4657 - accuracy: 0.8553 - val_loss: 0.1052
- val_accuracy: 0.9684
Epoch 3/7
102/102 [==============================] - ETA: 0s - loss: 0.0996 - accuracy: 0.9682WARNING:tensorflow:Can save
 best model only with val_acc available, skipping.
102/102 [==============================] - 96s 944ms/step - loss: 0.0996 - accuracy: 0.9682 - val_loss: 0.0444
- val_accuracy: 0.9867
Epoch 4/7
102/102 [==============================] - ETA: 0s - loss: 0.0473 - accuracy: 0.9852WARNING:tensorflow:Can save
 best model only with val_acc available, skipping.
102/102 [==============================] - 95s 935ms/step - loss: 0.0473 - accuracy: 0.9852 - val_loss: 0.0301
- val_accuracy: 0.9896
Epoch 5/7
102/102 [==============================] - ETA: 0s - loss: 0.0282 - accuracy: 0.9905WARNING:tensorflow:Can save
 best model only with val_acc available, skipping.
102/102 [==============================] - 97s 955ms/step - loss: 0.0282 - accuracy: 0.9905 - val_loss: 0.0296
- val_accuracy: 0.9896
Epoch 6/7
102/102 [==============================] - ETA: 0s - loss: 0.0213 - accuracy: 0.9931WARNING:tensorflow:Can save
 best model only with val_acc available, skipping.
102/102 [==============================] - 97s 947ms/step - loss: 0.0213 - accuracy: 0.9931 - val_loss: 0.0130
- val_accuracy: 0.9961
Epoch 7/7
102/102 [==============================] - ETA: 0s - loss: 0.0146 - accuracy: 0.9952WARNING:tensorflow:Can save
 best model only with val_acc available, skipping.
102/102 [==============================] - 95s 931ms/step - loss: 0.0146 - accuracy: 0.9952 - val_loss: 0.0160
- val_accuracy: 0.9951
Testing Accuracy: 99.43%
CNN Error: 0.57%
```

Figure 4.16: Training of CNN Model

4.2.15 Classification Report of the Model

The following figure shows the classification report of the classes on which the CNN model is trained.

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.98	0.99	112
1	0.94	1.00	0.97	93
2	1.00	0.97	0.98	91
3	0.99	1.00	1.00	112
4	1.00	1.00	1.00	83
5	1.00	1.00	1.00	89
6	0.98	1.00	0.99	108
7	1.00	0.97	0.98	101
8	1.00	0.98	0.99	126
9	0.99	0.98	0.99	105
10	1.00	1.00	1.00	97
11	1.00	1.00	1.00	91
12	1.00	1.00	1.00	102
13	1.00	1.00	1.00	94
14	1.00	1.00	1.00	88
15	0.99	1.00	1.00	109
16	1.00	0.99	0.99	97
17	1.00	1.00	1.00	89
18	1.00	1.00	1.00	111
19	1.00	1.00	1.00	95
20	0.96	0.97	0.96	96
21	1.00	1.00	1.00	111
22	0.99	0.96	0.98	106
23	0.97	1.00	0.98	87
24	1.00	0.98	0.99	97
25	1.00	1.00	1.00	109
26	0.99	1.00	1.00	101
27	0.96	1.00	0.98	105
28	0.99	0.98	0.98	93
29	0.97	1.00	0.98	91
30	1.00	0.98	0.99	112
31	1.00	1.00	1.00	100
32	1.00	0.97	0.99	75

32	1.00	0.97	0.99	75
33	0.99	1.00	0.99	97
34	1.00	1.00	1.00	93
35	1.00	0.98	0.99	108
36	1.00	1.00	1.00	104
37	1.00	1.00	1.00	93
38	1.00	1.00	1.00	102
39	1.00	1.00	1.00	89
40	0.99	0.99	0.99	99
41	1.00	1.00	1.00	95
42	0.99	1.00	0.99	93
43	1.00	1.00	1.00	126
44	1.00	1.00	1.00	95
45	1.00	1.00	1.00	115
46	1.00	1.00	1.00	95
47	1.00	1.00	1.00	110
48	1.00	1.00	1.00	117
49	1.00	1.00	1.00	72
50	1.00	1.00	1.00	121
accuracy			0.99	5100
macro avg	0.99	0.99	0.99	5100
weighted avg	0.99	0.99	0.99	5100

Figure 4.17: Classification Report

4.2.16 Validation graph

The validation graph of training and validation accuracy is as follow:

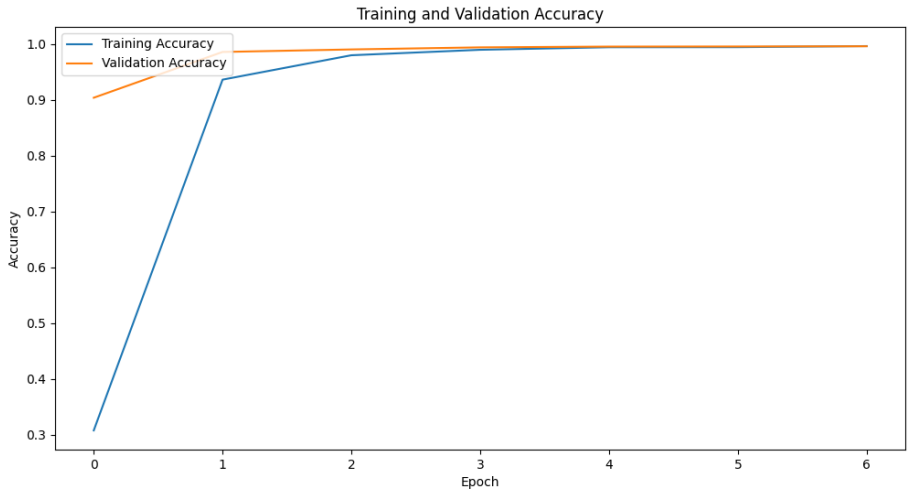


Figure 4.18: Validation Graph

4.2.17 Confusion Matrix

The confusion matrix heatmap working is given as follow:

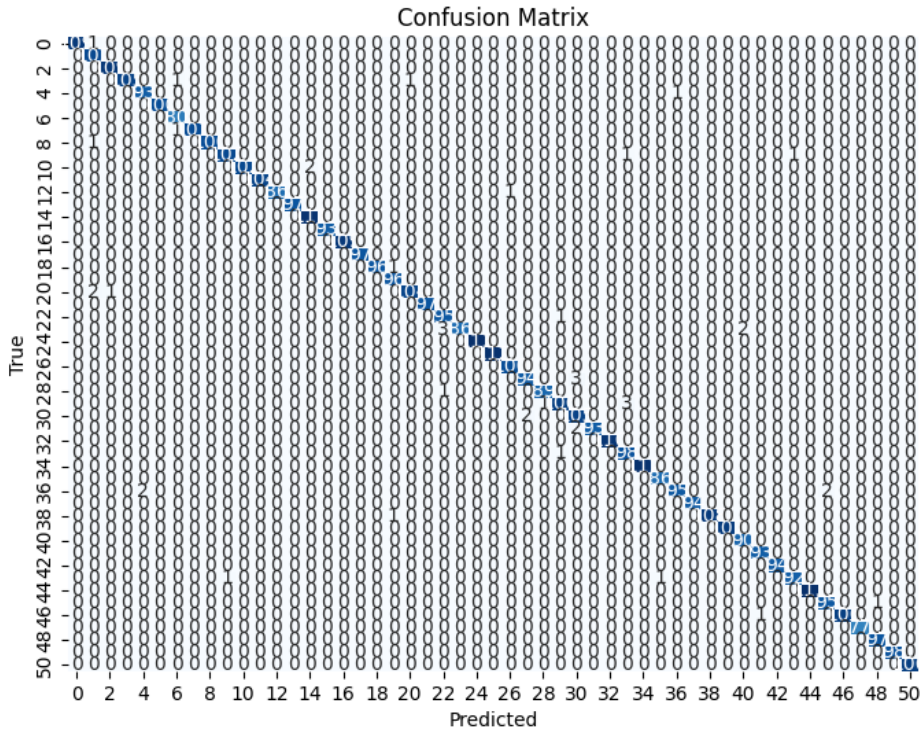


Figure 4.19: Confusion Matrix

4.2.18 Confusion Matrix Bar Plot:

The confusion matrix bar plot is given as follow:

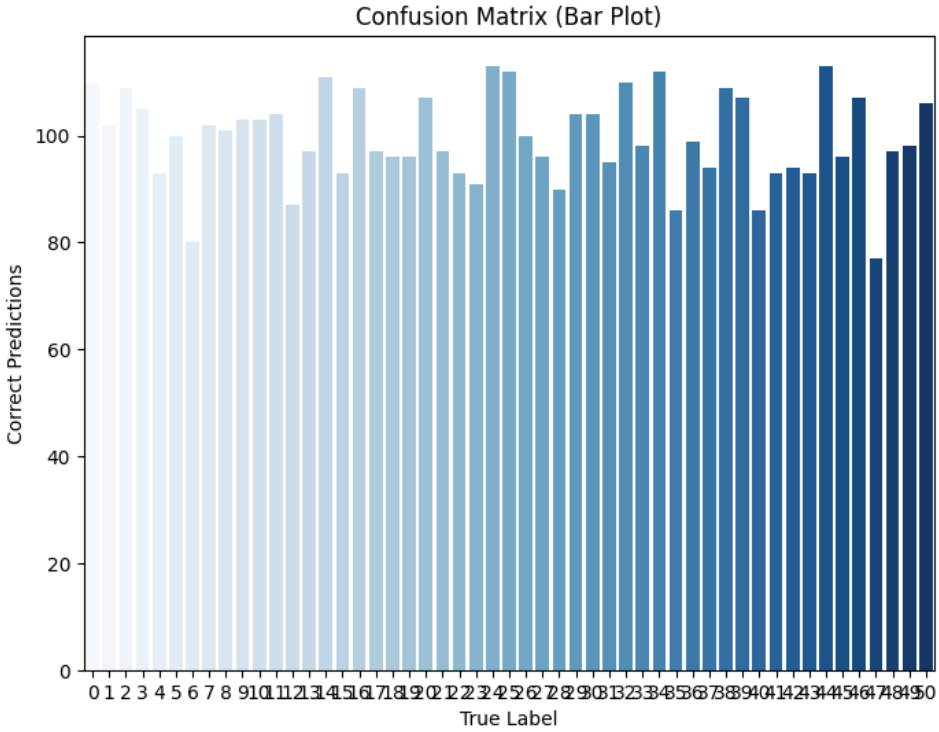


Figure 4.20: Confusion Matrix Bar Plot

4.2.19 AUC and ROC:

The AUC score of this model after training is 1.00 and the ROC curve is given as follow:

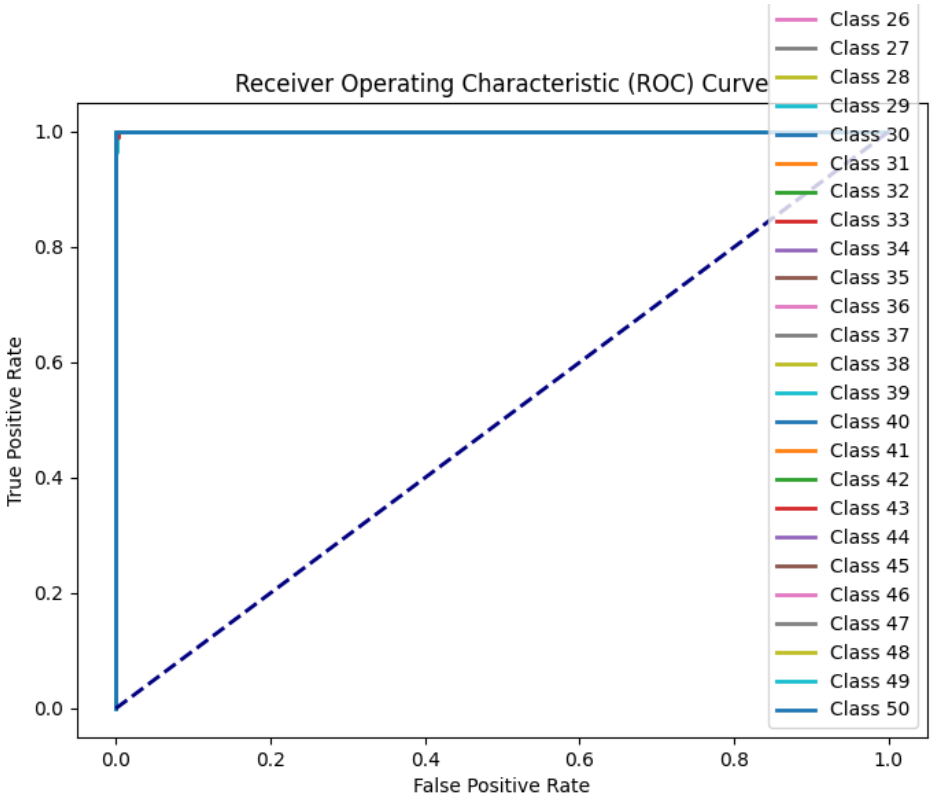


Figure 4.21: AUC and ROC

4.2.20 All Training Results

All of three training results is given as follows.

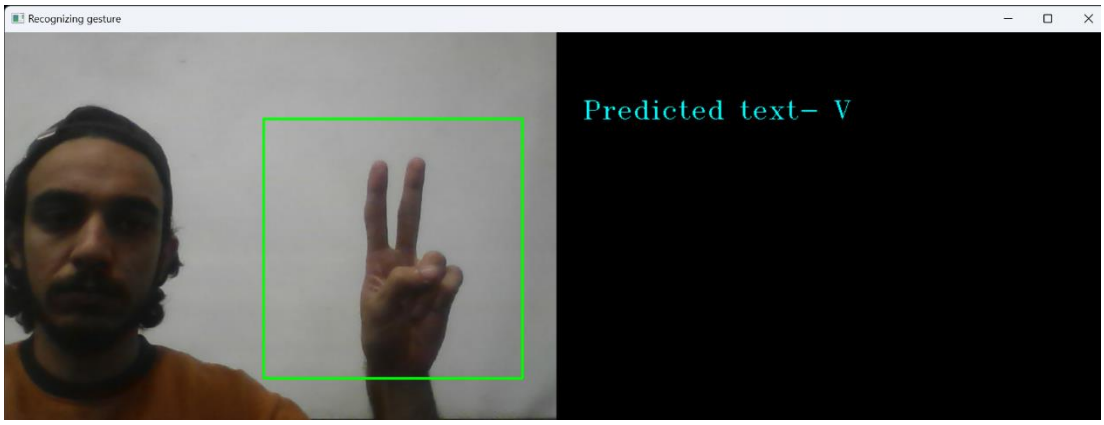
Table 4.1: Accuracy table of Training Results

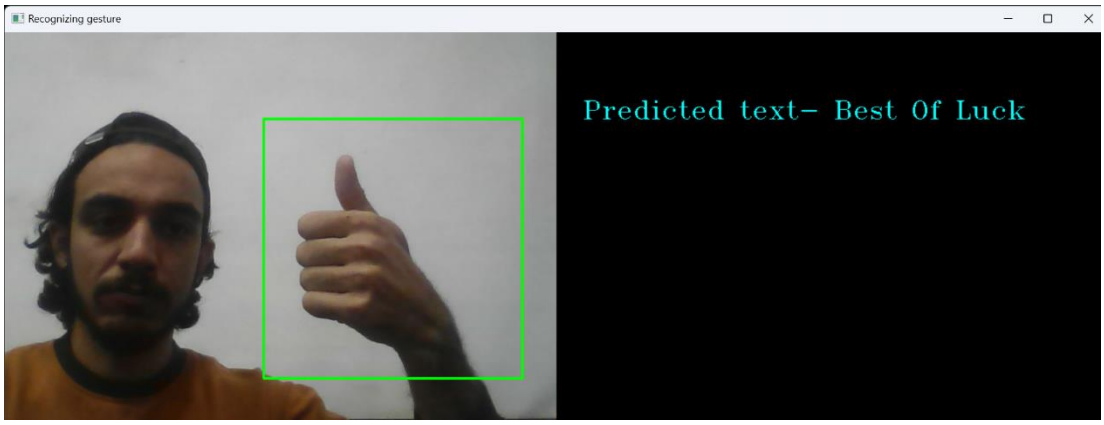
Data Set Ratio	Optimizers	Validation Accuracy
70-30	Adam	0.9951
60-40	Adam	0.9914
80-20	Adam	0.9951

CHAPTER 5

RESULTS AND DISCUSSIONS (or USER MANUAL)

5.1 Trained Model Output





CHAPTER 6

CONCLUSION AND RECOMMENDATIONS

6.1 Conclusion

The Sign Language Recognition system, harnessing the power of OpenCV and Convolutional Neural Networks (CNN), is a big step forward for assistive technology. The system effectively marries the sophisticated pattern recognition capabilities of CNNs with OpenCV's robust image processing, expertly converting complex sign language gestures into more accessible forms. This exciting integration not only enhances communication opportunities for the hearing and speech impaired, but it also highlights the power of technology in overcoming communications barriers. A custom-trained CNN, tuned to a variety of sign gestures--for high accuracy and real-time responsiveness means this is not only well suited for actual use but highly reliable as well. Its value may be most clearly seen in the contribution of this project to inclusivity and improved quality of life. Through AI, technology can truly make people's lives better.

6.2 Recommendations

There are several recommendations that must be considered when developing this Sign Language Recognition System based on OpenCV and CNN. More comprehensively, a wider set of data needs to be collected so that the system can handle many aspects across different languages and environments. The differentiation of gestures must be more fine-tuned, and differences in movement need to be considered as well. Second, by incorporating the more advanced deep learning techniques of transfer learning or

generative adversarial networks (GANs), model efficiency and accuracy can be greatly improved. Thirdly, constant user feedback testing will help improve the UI and provide an all-around better experience for users. Real-time feedback mechanisms that quickly correct any recognition errors will help. Also, working on integrating haptic feedback for blind users is another approach to making the system more accessible. Lastly, to guarantee that it can always be current and well maintained--that its neural network and image processing algorithms are properly modelled too will mean the system will remain robust even in the face of changing technologies or user requirements.

REFERENCES

[1] Lakhotiya, H., Pandita, H.S. and Shankarmani, R., 2021, May. Real Time Sign Language Recognition Using Image Classification. In *2021 2nd International Conference for Emerging Technology (INCET)* (pp. 1-4). IEEE.

[2] Urs, A.S., Raj, V.B., Pooja, S., Madhu, B.R. and Kumar, V., 2023, September. Action Detection for Sign Language Using Machine Learning. In *2023 International Conference on Network, Multimedia and Information Technology (NMITCON)* (pp. 1-6). IEEE.

[3] Seema and Singla, P., 2023. A Comprehensive Review of CNN-Based Sign Language Translation System. *Proceedings of Data Analytics and Management: ICDAM 2022*, pp.347-362.

[4] Sudha, M.V., Kumar, P.L., Areefa, S., Siva, K. and Rao, T.D.V.R., 2023. SIGN LANGUAGE TRANSLATION USING MACHINE LEARNING. *SIGN*.

[5] Kozhamkulova, Z., Nurlybaeva, E., Kuntunova, L., Amanzholova, S., Vorogushina, M., Maikotov, M. and Kenzhekhan, K., 2023. Two-Dimensional Deep CNN Model for Vision-based Fingerspelling Recognition System. *International Journal of Advanced Computer Science and Applications*, *14*(9). Kozhamkulova, Z., Nurlybaeva, E., Kuntunova, L., Amanzholova, S., Vorogushina, M., Maikotov, M. and Kenzhekhan, K., 2023. Two-Dimensional Deep CNN Model for Vision-based Fingerspelling Recognition System. *International Journal of Advanced Computer Science and Applications*, *14*(9).

[6] Srininvas, L.V., Raminaidu, C., Ravibabu, D. and Reddy, S.S., 2023. A framework to recognize the sign language system for deaf and dumb using mining techniques. *Indonesian Journal of Electrical Engineering and Computer Science*, *29*(2), pp.1006-1016.

[7] Gupta, A., Khan, A., Bansal, A., Verma, K. and Singh, J., 2023. Hand-Sign to Text Using CNN. *Journal of Pharmaceutical Negative Results*, pp.1160-1168.

[8] Kumar, A. and Raajkumar, G., 2023. Systematic Literature Survey on Sign Language Recognition Systems. *Deep Learning Research Applications for Natural Language Processing*, pp.195-203.

APPENDICES

APPENDIX A: Computer Program Listing

```
import cv2
import numpy as np
import pickle, os, sqlite3, random

image_x, image_y = 51, 51

def get_hand_hist():
    with open("hist", "rb") as f:
        hist = pickle.load(f)
    return hist

def init_create_folder_database():
    # create the folder and database if not exist1
    if not os.path.exists("gestures"):
        os.mkdir("gestures")
    if not os.path.exists("gesture_db.db"):
        conn = sqlite3.connect("gesture_db.db")
        create_table_cmd = "CREATE TABLE gesture ( g_id INTEGER NOT NULL PRIMARY KEY AUTOINC
        conn.execute(create_table_cmd)
        conn.commit()

def create_folder(folder_name):
    if not os.path.exists(folder_name):
        os.mkdir(folder_name)
```

```
def store_in_db(g_id, g_name):
    conn = sqlite3.connect("gesture_db.db")
    cmd = "INSERT INTO gesture (g_id, g_name) VALUES (%s, \'%s\')" % (g_id, g_name)
    try:
        conn.execute(cmd)
    except sqlite3.IntegrityError:
        choice = input("g_id already exists. Want to change the record? (y/n): ")
        if choice.lower() == 'y':
            cmd = "UPDATE gesture SET g_name = \'%s\' WHERE g_id = %s" % (g_name, g_id)
            conn.execute(cmd)
        else:
            print("Doing nothing...")
            return
    conn.commit()
```

```
def store_images(g_id):
    total_pics = 600
    hist = get_hand_hist()
    cam = cv2.VideoCapture(1)
    if cam.read()[0] == False:
        cam = cv2.VideoCapture(0)
    x, y, w, h = 300, 100, 300, 300
    create_folder("gestures/" + str(g_id))
    pic_no = 0
    flag_start_capturing = False
    frames = 0

    while True:
        img = cam.read()[1]
        img = cv2.flip(img, 1)
        imgHSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        dst = cv2.calcBackProject([imgHSV], [0, 1], hist, [0, 180, 0, 256], 1)
        disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10, 10))
        cv2.filter2D(dst, -1, disc, dst)
        blur = cv2.GaussianBlur(dst, (11, 11), 0)
        blur = cv2.medianBlur(blur, 15)
        thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
        thresh = thresh[y:y + h, x:x + w]
        contours, _ = cv2.findContours(thresh.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
```

```
import cv2, os

def flip_images():
    gest_folder = "gestures"
    images_labels = []
    images = []
    labels = []
    for g_id in os.listdir(gest_folder):
        for i in range(600):
            path = gest_folder+"/"+g_id+"/"+str(i+1)+".jpg"
            new_path = gest_folder+"/"+g_id+"/"+str(i+1+600)+".jpg"
            print(path)
            img = cv2.imread(path, 0)
            img = cv2.flip(img, 1)
            cv2.imwrite(new_path, img)

flip_images()
```

```python
import cv2
from glob import glob
import numpy as np
from sklearn.utils import shuffle
import pickle
import os

def pickle_images_labels():
    images_labels = []
    images = glob("gestures/*/*.jpg")
    images.sort()
    for image in images:
        print(image)
        label = image[image.find(os.sep)+1: image.rfind(os.sep)]
        img = cv2.imread(image, 0)
        images_labels.append((np.array(img, dtype=np.uint8), int(label)))
    return images_labels

images_labels = pickle_images_labels()
```

```python
if images_labels:
    images_labels = shuffle(images_labels)
    images, labels = zip(*images_labels)
    print("Length of images_labels", len(images_labels))

    train_images = images[:int(5/6*len(images))]
    print("Length of train_images", len(train_images))
    with open("train_images", "wb") as f:
        pickle.dump(train_images, f)
    del train_images

    train_labels = labels[:int(5/6*len(labels))]
    print("Length of train_labels", len(train_labels))
    with open("train_labels", "wb") as f:
        pickle.dump(train_labels, f)
    del train_labels

    test_images = images[int(5/6*len(images)):int(11/12*len(images))]
    print("Length of test_images", len(test_images))
    with open("test_images", "wb") as f:
        pickle.dump(test_images, f)
    del test_images

    test_labels = labels[int(5/6*len(labels)):int(11/12*len(images))]
    print("Length of test_labels", len(test_labels))
    with open("test_labels", "wb") as f:
        pickle.dump(test_labels, f)
    del test_labels
```

```
    val_images = images[int(11/12*len(images)):]
    print("Length of val_images", len(val_images))
    with open("val_images", "wb") as f:
        pickle.dump(val_images, f)
    del val_images

    val_labels = labels[int(11/12*len(labels)):]
    print("Length of val_labels", len(val_labels))
    with open("val_labels", "wb") as f:
        pickle.dump(val_labels, f)
    del val_labels
else:
    print("No images found in the specified directory.")
```

```
import numpy as np
import pickle
import cv2, os
import tensorflow as tf
from glob import glob
from keras import optimizers
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.utils import to_categorical
from keras.callbacks import ModelCheckpoint
from keras import backend as K
```

```
def cnn_model():
    num_of_classes = get_num_of_classes()
    model = Sequential()
    model.add(Conv2D(64, (2, 2), input_shape=(image_x, image_y, 1), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=(3, 3), padding='same'))
    model.add(Conv2D(256, (5, 5), activation='relu'))
    model.add(MaxPooling2D(pool_size=(5, 5), strides=(5, 5), padding='same'))
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(num_of_classes, activation='softmax'))

    # sgd = optimizers.SGD(learning_rate=1e-2)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    filepath = "cnn_model_h5.keras.h5"
    checkpoint1 = ModelCheckpoint(filepath="cnn_model_h5.keras.h5", monitor='val_acc', verbose=1, save_best_only=True, mode='max')
    callbacks_list = [checkpoint1]

    return model, callbacks_list
```