SIGN LANGUAGE RECOGNITION

A Project Report

Submitted in the partial fulfillment of the requirements

for the award of the degree

Of

Bachelor of Technology

in

Department of Computer Science and Engineering

By

Thangellapally Srirag	180330281
Peddy Shiva Sai	180330040
M Mosaad Ahmed	180330033

under the supervision

of

Mr. Sachin Kumar

Assistant Professor, CSE, KLH

Department of Computer Science and Engineering

K L University Hyderabad,

Aziz Nagar, Moinabad Road, Hyderabad – 500 075, Telangana, India.

November, 2021

DECLARATION

The Project Report entitled "**SIGN LANGUAGE RECOGNITION**" is a record of bonafide work of **Thangellapally Srirag (180330281), Peddy Shiva Sai (180330040), M Mosaad Ahmed (180330033)** submitted in partial fulfillment for the award of B.Tech in the Department of Computer Science and Engineering to the K L University, Hyderabad. The results embodied in this report have not been copied from any other Departments/ University/ Institute.

Thangellapally Srirag (180330281)

Peddy Shiva Sai (180330040)

M Mosaad Ahmed (180330033)

CERTIFICATE

This is to certify that the Project Report entitled "**SIGN LANGUAGE RECOGNITION**" is being submitted by **Thangellapally Srirag (180330281), Peddy Shiva Sai (180330040), M Mosaad Ahmed (180330033)** submitted in partial fulfillment for the award of B.Tech in **Computer Science & Engineering** to the K L University, Hyderabad is a record of bonafide work carried out under our guidance and supervision.

The results embodied in this report have not been copied from any other departments/ University/Institute.

Signature of the Supervisor

Mr. Sachin Kumar

Assistant Professor, CSE, KLH

Signature of the HOD

Signature of the External Examiner

ACKNOWLEDGEMENT

First and foremost, we thank the Lord Almighty for all of his kindness and mercy, which enabled us to successfully complete this endeavour.

We would like to take this occasion to express our gratitude to our beloved Founder and Chairman, who has been a continual source of support and motivation throughout our course. We owe a debt of gratitude to our Principal, **Dr. L. Koteswara Rao**, who has tirelessly championed all of our curricular activities.

We express our deep gratitude to **Dr. Chiranjeevi Manike**, our Department Head, for her outstanding supervision, monitoring, and consistent encouragement during the project. We express our heartfelt gratitude to our Project Coordinator, **Dr. K. G. Suma**, who has provided consistent logistical help and encouragement during our project. We thank **Mr. Sachin Kumar** of our Department who has supported throughout this project holding a position of supervisor.

We sincerely thank everyone of our department's teaching and non-teaching staff, without whom we would not have been able to complete this project. We would like to express our heartfelt gratitude to our parents, family members, and friends who have helped us make this initiative a huge success.

Thangellapally Srirag (180330281)

Peddy Shiva Sai (180330040)

M Mosaad Ahmed (180330033)

ABSTRACT

Sign language recognition is a natural way for humans and machines to communicate, and many researchers in academia and industry are working on it now. It allows humans to communicate with machines in a simple and comfortable manner. Sign language plays an important part in communicating with people who are physically disabled. Good communication fosters greater understanding among all members of the society, including the deaf. It is most widely used to interact with physically disabled people who have hearing or speech problems, as well as with normal people. This aids in the expression of people's thoughts and emotions. In the area of technology, a sign language recognition device may be useful. It offers a unique and user-friendly solution. The goal of this project is to offer a real-time vision system for visual interaction environments that recognizes hand gestures utilizing general-purpose hardware and low-cost sensors, such as a simple computer and a USB web camera, so that any user can use it at work or at home. A review of hand gesture recognition approaches for sign language recognition is reviewed, along with problems and future research directions, using various tools and algorithms used to sign language recognition systems. The input features and the selection of suitable feature representation are the most critical aspects of a hand gesture recognition system. This project gives a review of hand postures and gesture recognition systems, which is both a tough and promising subject in the context of human-computer interaction. With the explanation of the system recognition framework and its key phases, several applications and methodologies were covered.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1

INTRODUCTION

The word "Sign Language" refers to a language that is mainly used by deaf people and includes hand gestures as well as other motions such as facial expressions and body postures. Humans have the innate ability to recognize continuous and autonomous sign language after being taught to recognize and understand it. One out of every thousand babies were born deaf, according to the National Institute on Deafness. An additional one to six people out of every thousand are born with varying degrees of hearing loss.

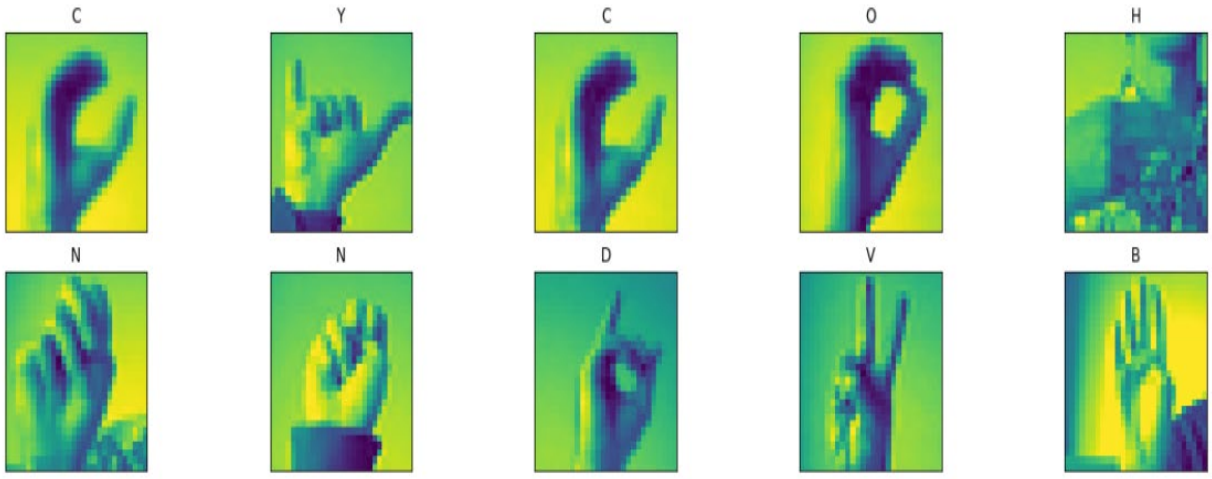

Fig.1.1. Dataset with different signs

Deaf and dumb people, like anyone else, can communicate their feelings and thoughts with or without words. Gestures can be used to control robots and operate various devices. We can change the channels on the television using hand gestures. Virtual reality interaction uses hand gestures to manipulate virtual actions using one or two hands for 2D and 3D interaction displays. Geometry architecture for dimensionality: Easily view various 3D designs from various angles (E.g. AutoCAD). Following the lead of other research papers, we noticed that the input is derived from images, videos, Motion detection. The current thesis examines a variety of studies on hand gesture recognition for recognizing various signals, as well as the various stages of recognition systems. Deep learning, machine learning algorithms and image processing techniques have also been used in numerous papers. A comparative analysis of all of these works is presented to guide beginners in their work, as well as our proposed methodology and results of models that we have implemented are also included.

CHAPTER 2

LITERATURE SURVEY

We looked at a lot of academic papers and saw how different principles and techniques are used in sign language recognition. The following are the researcher's methods and accuracy levels.

AI-LSTM, Max CNN-LSTM, Spatial AI-LSTM [9].

They used skeletal data, which is RGB video data, and used it to perform skeletal tracking. On this data and trained models, AI-LSTM, Max CNN-LSTM, Spatial AI-LSTM were performed. Following the completion of model training, these trained models were used to test and obtain accuracy.

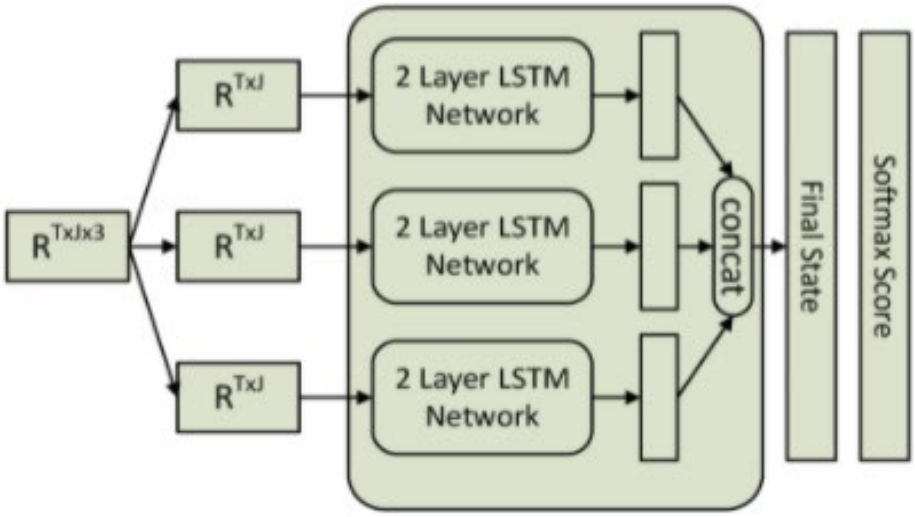

Fig-2.1. Architecture for AI-LSTM, Max CNN-LSTM, Spatial AI-LSTM

Data from distinct axes is divided and entered into three independent LSTM networks right before entering the main network. The model combines the final states from each of the individual LSTM networks, then feeds them into the softmax layer for classification. Axis Independent Architecture is a term used to describe this technique (Al LSTM). At a given time step, however, there may be spatial interactions with joints. It is unable to capture any such interaction between joints throughout time. We propose a simple new data augmentation technique for skeletal data to add spatial relationships among joints. This is accomplished by origin transfer. We use each wrist joint as the origin for each frame in a gesture sample and deduct that origin from all other joints data. This method adds spatial information to the input.

Support Vector Machines and Hidden Conditional Random Fields [2]

For classification they used support vector machine. As a measure of sparseness and performance, they chose results from SVM configurations and the average number of Support Vector evaluations. A two-layer architecture focused on automated learning of discriminative models to function on different stages of the SVM and HCRF recognition process for Sign Language Recognition.

In this case, HCRF is used to recognize hand gesture patterns.

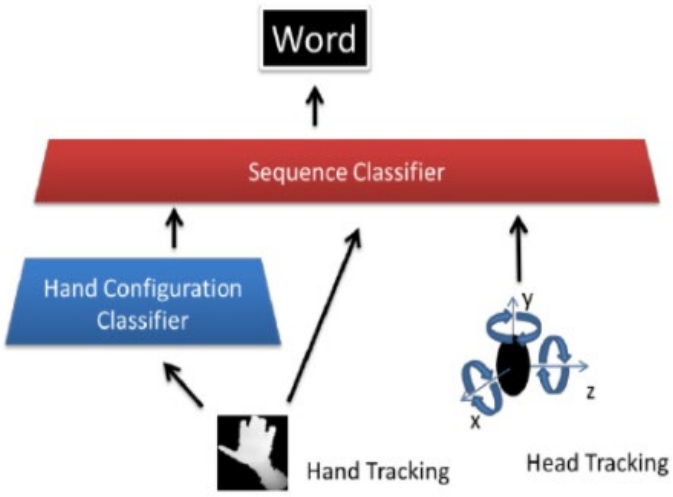

Fig.2.2. Methodology for Support Vector Machines and Hidden Conditional Random Fields

The approach for detecting sign words from Sign Languages is clearly influenced by the field of speech recognition, which has accumulated a large and growing body of knowledge over time. We have a first layer, aimed at detecting static gestures such as hand shapes from Brito's set of hand configurations, and a second layer, aimed at classifying sequences of static gestures plus temporal, trajectory, and facial information into words from a finite lexicon, similar to how speech recognition methods often build language models over phoneme classifiers.

We built a number of SVMs with different kernel functions and multi-class decision techniques for the first processing layer. For comparison, we developed feed-forward activation neural networks with varied numbers of hidden neurons. All categorization machines were built to learn from a rescaled depth image of the user's hand. Despite the features' simplicity, the raw performance of this layer in properly detecting each of the hand configurations in will be less essential than the classifi-regularity er's in classifying identical gestures with similar labels. The language model's hand configuration labels are merely utilised as a guide here; as long as the gesture recognition layer can detect patterns in the class labels created at this point, absolute correctness is not required.

Kohonen-SOM, BPN, RBFNN [3]

The kohonen self-organized map is used to classify static movements that include some of the ISL alphabets (Indian Sign Language).

The kohonen self-organized map creates topologically ordered mappings between input data and the map's processing elements, which are image data after conversion to binary data.

Feed forward BPN and RBFF was used in for the classification of static gestures involving alphabets of ASL (American sign language).

They also experimented with both ISL and ASL.

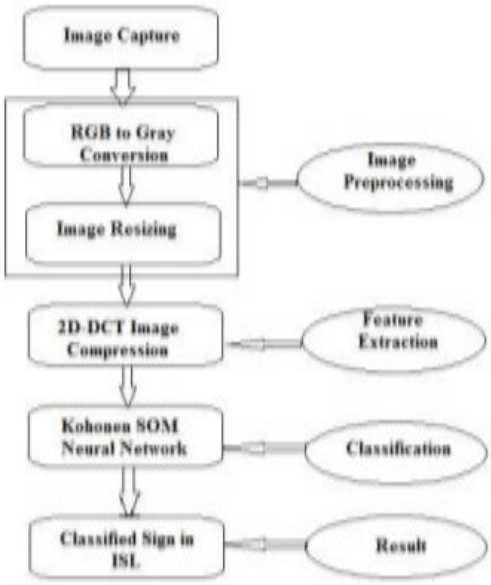

Fig.2.3. Schematic View of Proposed Hand Gesture Recognition System for ISL [12]

The system works with photographs of bare hands, allowing the user to interact with it in a natural manner. Hand movements in Indian sign language are both static and dynamic. During a brief period of time, a static gesture is seen. A dynamic gesture is one that is intended to evolve over time. To comprehend a whole communication, all static and dynamic gestures are interpreted over time. Hand gesture recognition is the name given to this difficult technique. One of the most common neural network models is the Self-Organizing Map (SOM). It belongs to the competitive learning networks category. It is based on unsupervised learning, which means that no manual intervention is required during the learning process and that little information about the input data is required. As a result, we may use the SOM to cluster data without knowing the input data's class memberships. Because the SOM may be used to detect problem-related features, it's also known as the Self-Organizing Feature Map, or SOFM (SOFM). In this article, we used a specific type of SOM known as a Kohonen Network. With a single computational layer arranged in rows and columns, this SOM has a feedforward structure.

K-Means Clustering, Sparse Autoencoder [4], We have observed that deep learning concepts and image processing techniques have shown high accuracy compared to other techniques.

K-Means and Sparse Autoencoder are both unsupervised learning algorithms in this case.

The K-Means algorithm is used to reduce the distance between data points and their centroids.

To reduce squared reconstruction error, the Sparse Autoencoder is used.

This method occurs after the image preprocessing is completed, and the preprocessed data is sent into these models.

	K-means			Sparse Autoencoder		
K	**LR-L1**	**LR-L2**	**SVM**	**LR-L1**	**LR-L2**	**SVM**
# of frames = 2						
100	70.63	69.62	68.87	67.40	66.53	65.73
300	73.73	74.05	73.03	72.83	73.48	70.52
500	75.30	**76.53**	75.40	72.28	**74.65**	68.72
# of frames = 3						
100	72.48	73.30	70.33	68.68	67.40	68.33
300	74.78	74.95	74.77	74.20	74.72	70.85
500	77.27	**77.50**	76.17	72.40	**75.45**	69.42
# of frames = 4						
100	74.85	73.97	69.23	68.68	67.80	68.80
300	76.23	76.58	74.08	74.43	75.20	70.65
500	**79.08**	78.63	76.63	73.50	**76.23**	70.53

Table 2: 3D filters $(15 * 15 * 2)$: Leave-one-signer-out cross-validation average accuracies.

	BSL	DSL	FBSL	FSL	GSL	NGT
BSL	**56.11**	2.98	1.79	3.38	24.11	11.63
DSL	2.87	**92.37**	0.95	0.46	3.16	0.18
FBSL	1.48	1.96	**79.04**	4.69	6.62	6.21
FSL	6.96	2.96	2.06	**60.81**	18.15	9.07
GSL	5.50	2.55	1.67	2.57	**86.05**	1.65
NGT	9.08	1.33	3.98	18.76	4.41	**62.44**

Fig-2.4. Metrics for K-means clustering and sparse autoencoder

Extract tiny videos (hence referred to as patches) at random from the video samples. We resize the patches so that they all have rrows, ccolumns, and fframes, then we mtimes extract patches. This results in X=x(1), x(1),..., x(m), where x(i)RNandN=rcf (the size of a patch). We extract 100,000 patches of size 15 15 1(2D) and 15 15 2(2D) for our experiments (3D).

Normalization and whitening (Hyvärinen and Oja, 2000) have been shown to improve performance in unsupervised feature learning (Coates et al.,2011). As a result, we subtract the mean from each patchx(i) and divide by the standard deviation of its parts to normalise it. Normalization refers to localbrightness and contrast normalisation for visual data.

3DCNN and LSTM with FSM Context-Aware Model [8].

By integrating these three models, they were able to conduct experiments in which 3DCNN (three-dimensional convolutional neural network) and LSTM (long short-term memory) were used to extract spatiotemporal features, and FSM (Finite state machine) was used to monitor the model's class decision effects.

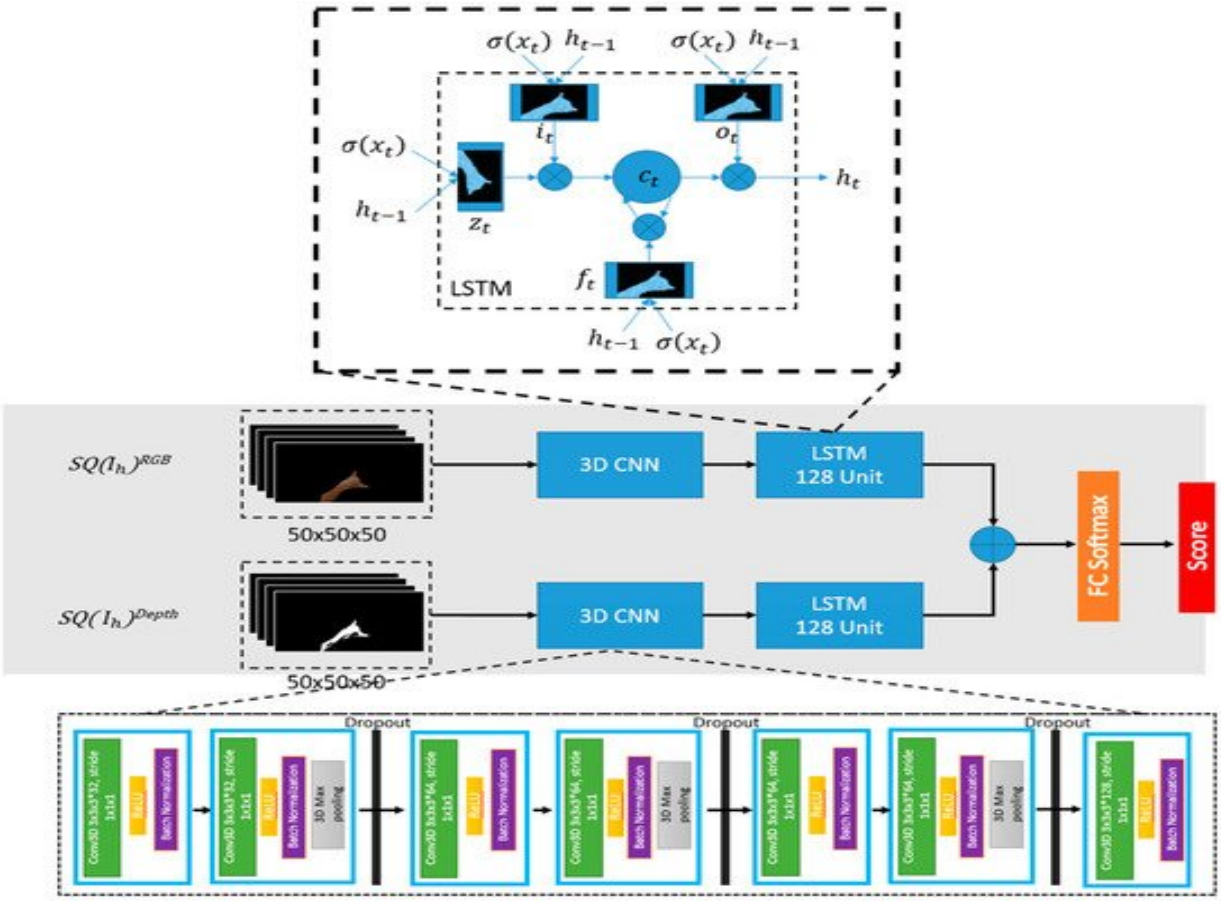

Fig-2.5 . Architecture for 3DCNN and LSTM with FSM Context-Aware Model

The 3DCNN + LSTM design that has been proposed. The model's general architecture is depicted in the centre of the figure, while the bottom and top depict how the 3DCNN layer is implemented and how the LSTM unit interacts with the gesture sequence, respectively. Using both depth and RGB as input data instead of just one stream input could result in a better result.

As a result, 3DCNN + LSTM multimodal model variants are proposed. According to the degree of fusion, there are three sorts of multimodal types. The first method is known as the early fusion model, and it only requires one model stream because it combines both RGB and depth data into channels. The new fusion input is the new picture IDi with four-channel combinations, given Ii RGB colour image with three channels and Di depth distance data with one channel.

Kinect Sensor Based on Conditional Random Fields [5].

The kinetic sensor (Microsoft Kinetic sensor) they used is to get 3D depth information from the hand emotions.

The next main procedure where the conditional random fields (CRF) is used for detecting the type of hand sign and then boost map embedding method is involved for verifying hand shapes of the segmented signs.

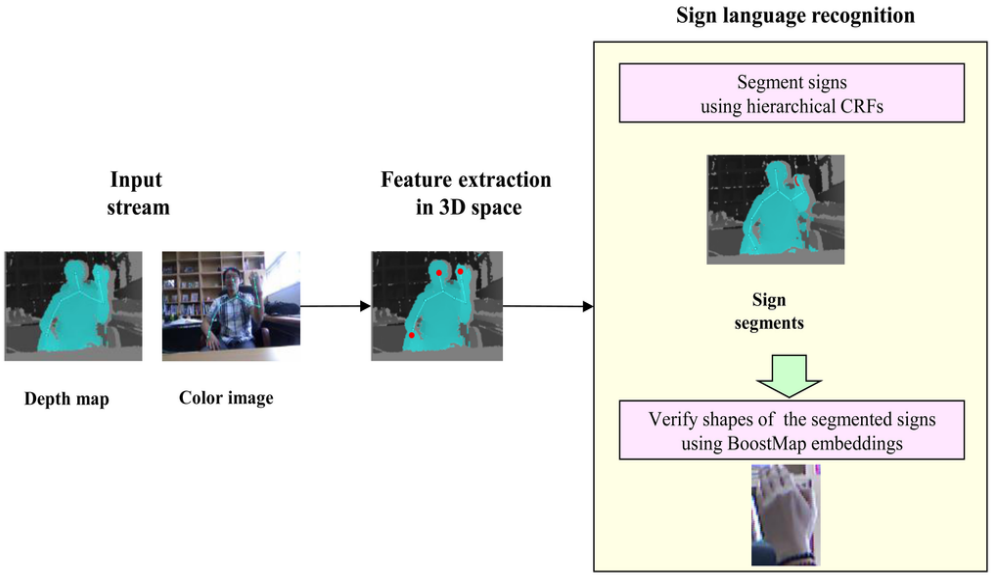

Fig-2.6. Methodology for CRFs

It depicts the sign language recognition system's framework. The Kinect is used because it captures both a colour image and a depth map. In a variety of illumination circumstances, the hand and facial locations are reliably identified. An H-CRF is used to detect potential sign segments utilising hand motions and positions after detecting the locations of the face and hands. The hand shapes of the segmented signs are then verified using the BoostMap embedding approach. The hand tracking component in the Kinect Windows software development kit reliably detects the face and hand locations. The skeletal model is made up of ten feature points taken from the upper body.

The hand region is determined by setting a threshold based on the hand position, To segment the hand form, the signer wears a black bracelet, which is detected using Random Sample Consensus (RANSAC). The hand form that has been observed has been normalised. Ten sequences for each sign in the 24-sign vocabulary were gathered to train the CRFs and H-CRFs. During data collection, the signer wore a black bracelet. The start and end positions of the ASL signs were manually inserted to the training data, and they were utilised as the ground truth to test the proposed method's performance. A Kinect device was used to record the video. Seven of the 24 signs were one-handed, while 17 were two-handed.

Eigen Value Weighted Euclidean Distance Based Classification Technique [1]

The procedure entails applying skin filtering to the input image, hand cropping from the binary image feature extraction, and calculating distance between two rows of data with integer or floating-point values using classification Euclidean distance.

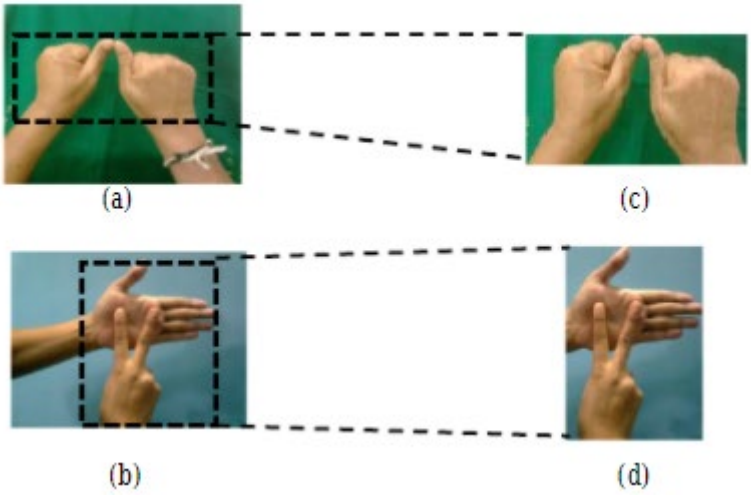

Fig-2.7. Hand Cropping

Only the hand piece till the wrist is required for different gesture identification, so the unnecessary area is chopped off using this hand cropping approach. The importance of employing this hand cropping is that it allows us to detect the wrist and hence reduce the unwanted area. Once the wrist is located, the fingers are easy to find because they are in the opposite region of the wrist. The feature extraction process begins when the desired section of the image has been cropped. The cropped image is used to determine Eigen values and Eigen vectors.

$$E.\ D. = \sqrt{\sum_{n=1}^{m} \left(EV1(n) - EV2(n)\right)^2}$$

$$C2 = (E.\ D.) * |E1\text{-}E2|$$

Classification based on Euclidean Distance: The distance between the Eigen vectors of the test image and the corresponding Eigen vectors of the database image was calculated using Euclidean distance. We got five Euclidean distances for each database image since five Eigen vectors were analyzed, and the minimum of each was found.

Classification based on weighted Euclidean distance based on Eigen values: The difference between the test image's Eigen values and the database image's Eigen values was discovered. The Euclidean Distance acquired in the first level of classification was then multiplied by C2 in the equation below. The sum of the results for each image was then summed, and the lowest of these was chosen as the recognized sign.

Leap Motion Controller and Two-Layer Bidirectional Recurrent Neural [10].

They used a leap motion controller to capture hand gestures and a two-layer bidirectional recurrent neural network to handle time-based sequences that display associations between closely related data items, as well as LSTM after data preprocessing.

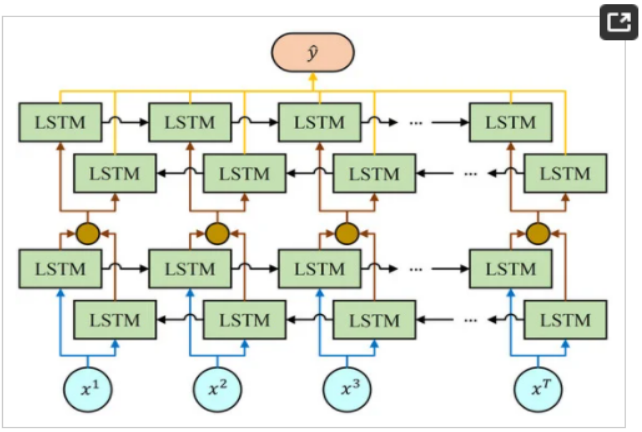

Figure 5. The two-layer Bidirectional Recurrent Neural Network. The outputs of the two corresponding LSTM units are added up in the first layer and sent to the second layer.

Fig-2.8. The Two-layer bidirectional RNN

In the network, the Long Short-Term Memory unit is referred to as the memory block and is made up of three multiplicative gates: input gate (update gate), output gate, and forget gate. The memory block also contains memory cells that are self-connected and save the network's temporal information. The input gate is in charge of processing the information input. The output gate directs the flow of cell activation output to the remaining units.One Recurrent Neural Network travels ahead in the single layer Bidirectional Recurrent Neural Network. Another, on the other hand, travels in the opposite direction. The input is delivered in opposing directions to two independent Long Short-Term Memory units at each time point, and their results are combined based on the hidden state. In this study, they used this structure to create a two-layer BRNN that combines the results of both networks' final Long Short-Term Memory unit.

In all types of neural networks, the learning rate is the most significant hyper-parameter. As a result, in this study, the Cyclical Learning Rate approach was used, which eliminates the requirement to adjust the learning rate while still achieving near-optimal classification accuracy. CLR allows the learning rate to vary cyclically between suitable border values that are attained by linearly raising the learning rate over a few epochs, rather than monotonically reducing it. Experiments were carried out using an LMC and a computer with an Intel i5 3.2GHz CPU and 8GB RAM to estimate the performance of the proposed two-layer BRNN model. Cross-entropy loss function, Adam gradient descent technique, and changing learning rate were used to train the model. The model was created with the help of a C++ framework.

Hidden Markov Model and Wearable Motion Sensors [7].

To obtain data on hand gestures, they used an EMG and accelerometer-based technique. Other sensors, such as a magnetometer and a gyroscope, are used in this approach to determine orientation and angular velocity. The data was trained and evaluated using the Hidden Markov Model, which is a probabilistic method.

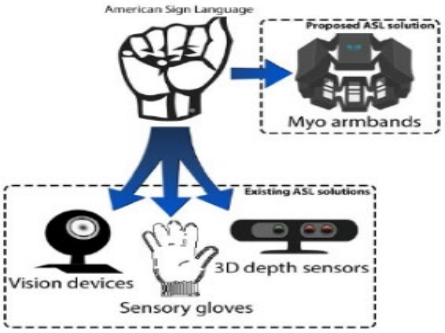

Fig-2.9. Wearable motion sensors

The sensors utilised in ASL recognition solutions are categorised into four broad groups. 3D depth sensors, vision-based approaches, and sensory gloves EMG/Accelerometer-based sensors, such as the Myo armband, have also been proven to help in sign language recognition. The orientation, gyroscope, and accelerometer data from both myos, as well as the EMG data from the dominant hand myo, were all utilised in the training and recognition of the HMMs. The information was organised into a 26-element time-series array. The IMU data is updated at a rate of 50 Hz, while the EMG data is received at a rate of 200 Hz, implying that a 5-second recording will offer 250 new IMU samples and 1000 new EMG samples. Combining the input from both myos into a single feature vector.The electrical activity of the muscles is represented by EMG data. The myo's EMG data is given as an 8-element byte array, with EMG values ranging from -128 to 127 and no units. EMG data is also incredibly user-dependent, as elements like arm length, arm circumference, and the exact position of the arm from which the data originates all play a role in realising the data. The acceleration of a free-fall is measured by an accelerometer unit. The myo device comprises a tri-axial accelerometer unit, which consists of three accelerometers that are aligned in the x,y, and z axes and measure relative acceleration along their respective dimensions. The angular velocity of a gyroscope is measured in one dimension. It is feasible to obtain changes in the device's rotational orientation from a frame of reference or from a known orientation of the device. Myo's gyroscope is also tri-axial, with each gyroscope recording the change in rotational orientation along its respective dimension. In each of its axes, the magnetometer determines the device's orientation in relation to the Earth's magnetic field. A magnetometer, to put it simply, measures the direction and strength of the magnetic field in one dimension.

SVM, Random Forest, Hierarchical Classification [6].

To begin with, they have segmented the skin portion of the picture.

They used HSV and SIFT in the segmentation and feature extraction phase before practicing.

After segmenting the skin, they extracted relevant features and fed them into supervised machine learning models like SVM, Random Forest, and Hierarchical Classifier, which they then trained so that the trained model could be used on test data and evaluated.

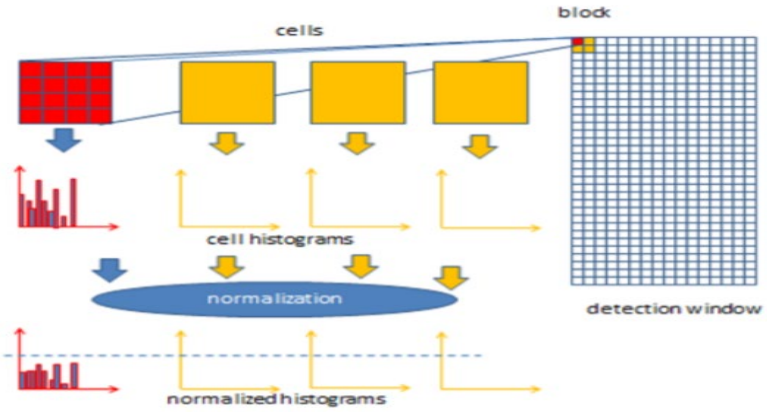

Figure 10: Histogram of Oriented Features

src : https://software.intel.com/en-us/node/529070

Fig-2.10. Histogram of oriented features

They fed the whole 32000-dimensional hog feature vectors into a multiclass SVM with linear kernel. The 4 fold CV improved little to 54.63 percent, indicating that not much information was lost through Gaussian Random Projection dimension reduction.The accuracies they mentioned in the previous section correspond to their best results for each of the feature vectors. However, before obtaining the best results, they investigated the following techniques on the feature vectors obtained. Almost every feature vector was subjected to a multiclass SVM with a linear kernel.On all of the feature vectors, multiclass SVMs were used. For all feature vectors, the results of linear kernel and fourfold Cross Validated accuracies are presented.

The confusion matrices in the following sections correspond to the various strategies that were explored using linear kernel MultiClass SVMs. This method was found to have the highest accuracy. On HOG feature vectors, their attempt using the 'rbf' kernel failed badly, with only 4.76 percent accuracy.They used Random Forest with HOG feature vectors on the compressed images because it was a 26-class problem. With 46.45 percent 4 fold CV accuracy, it fell short of the Multiclass SVM. Their interpretation is that decision tree implementation in Python is monothetic, which may have resulted in the construction of rectangular regions despite the fact that the real borders were not rectangular.

Study	Implemented Model	Accuracy
Indian Sign Language Recognition Using Eigen Value Weighted Euclidean Distance Based Classification Technique.	Eigen Value Weighted Euclidean Distance	97%
Sign Language Recognition with Support Vector Machines and Hidden Conditional Random Fields: Going from Fingerspelling to Natural Articulated Words	SVM and HCRF	95.28%
Indian sign language recognition	SOM,BPN,RBFNN	80%,92.78%,99.9%
Unsupervised Feature Learning for Visual Sign Language Identification	K-Means Clustering	84.03%
Sign Language Recognition with the Kinect Sensor Based on Conditional Random Fields	CRF	90.4%
Mukerjee, Indian Sign Language Character Recognition	SVM, Random Forest	54.63%,46.46%
American Sign Language Recognition using Hidden Markov Models and Wearable Motion Sensors.	HMM	96.15%
Dynamic Hand Gesture Recognition Using 3DCNN and LSTM with FSM Context-Aware Model.	LSTM	89%
Sign Language Recognition Analysis using Multimodal Data.	AI-LSTM, Spatial AI-LSTM	58%,69%
Dynamic Hand Gesture Recognition Based on a Leap Motion Controller and Two-Layer Bidirectional Recurrent Neural Network	RNN	95.2%

Table 2.1 Table comparing accuracies of models

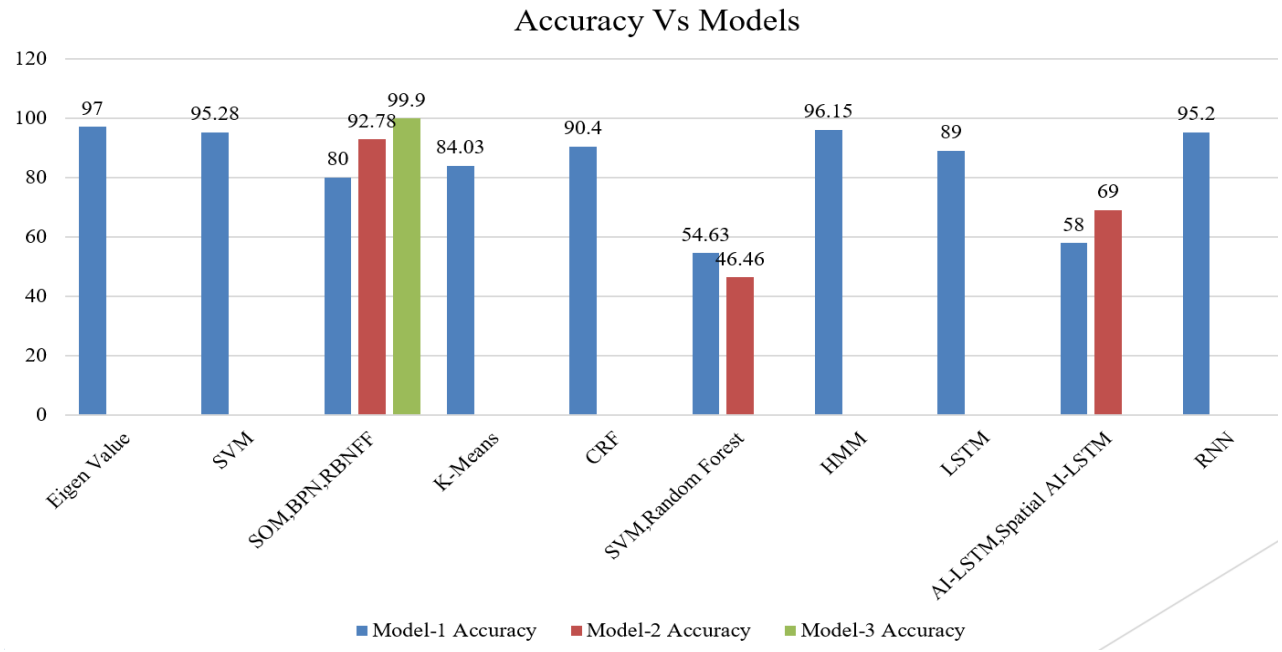

Fig-2.11. Visualization of accuracies of models

CHAPTER 3

HARDWARE & SOFTWARE REQUIREMENTS

3.1 Hardware Requirements:

- Intel or AMD x86-64 processor
- Minimum 4GB of RAM.
- Minimum 100 GB of internal storage
- Camera

3.2 Software Requirements:

- Windows
- Anaconda-Jupyter Notebook/Visual Studio Code

CHAPTER 4

FUNCTIONAL & NON-FUNCTIONAL REQUIREMENTS

4.1 Functional Requirements:

Functional requirements define what a system or system component must be able to do.

1) Capturing images and Reading pixel values

2) Angle identification between fingers

4.2 Non-Functional Requirements:

Non-functional requirements are the conditions under which a system can run.

1) Real-time - The device can recognize signs and translate them into speech in an unnoticeable amount of time for its users.

2) Accuracy – Signs should not be mixed up, and the machine should be able to interpret the correct symbol.

3) Usability – The framework should allow users to communicate in a natural way. The hearing-impaired person just needs to be concerned with executing signals

CHAPTER 5

PROPOSED SYSTEM

- Fetching the pixel values of various images from the dataset.
- Data preprocessing-Converting multi class labels to binary labels and reshape according to image size, splitting into train and test.
- To train the data, deep learning concepts such as Convolutional Neural Networks (CNN) and machine learning algorithms are used.
- Using the qualified model, evaluating the test data and estimating the accuracy of the test data
- Ensemble Learning of machine learning models.
- There is also an additional method in which we have used image processing techniques and OpenCV to recognize the meaning of hand gesture which will be captured from camera.

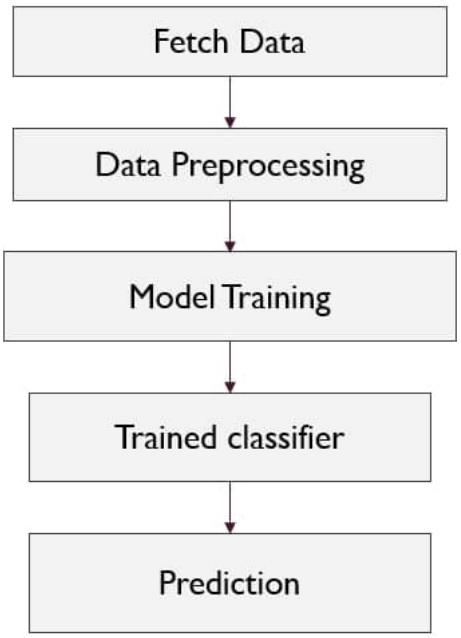

Fig-5.1. Proposed Methodology

CHAPTER 6

IMPLEMENTATION

The dataset we use is the MNIST for Sign Language. It's in CSV format, with single rows of labels and pixel values. Every training and test case represents a mark (0-25) as a one-to-one map for each alphabetic letter A-Z (due to gesture movements, there are no cases for 9=J or 25=Z). There are 27,455 cases in the training data and 7172 cases in the test data, each with a header row of mark, pixel 1 to pixel 784, representing a single 28x28 pixel image with grayscale values ranging from 0-255.

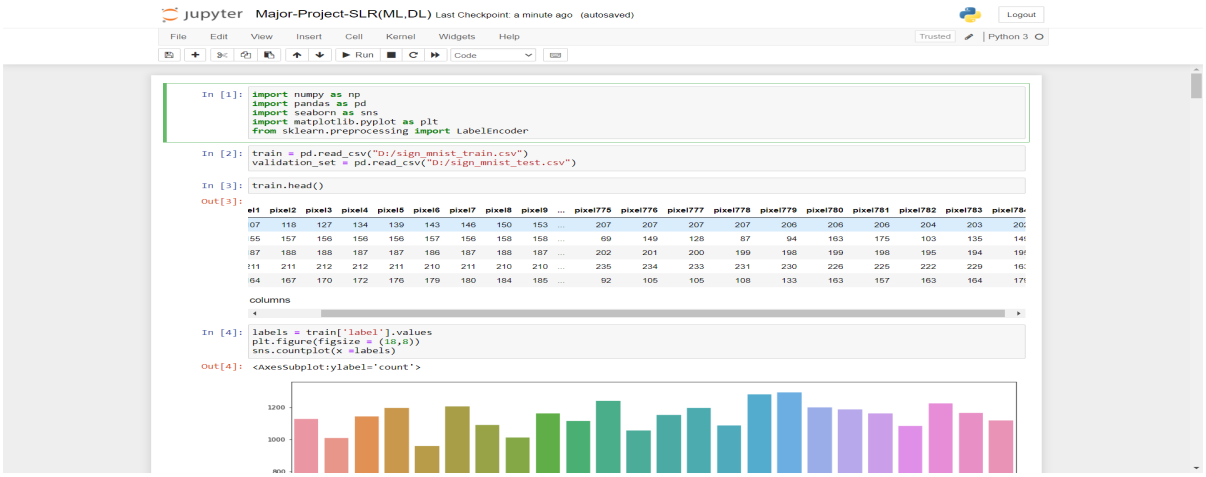

Fig-6.1. Fetching data

Fig-6.2. Plotting dataset and allocating data

The machine learning and deep learning models that were employed are listed below.

K-nearest neighbours:

K-nearest neighbours (KNN) algorithm uses ‘feature similarity’ to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. With the help of K-NN, we can easily identify the category or class of a particular dataset. The KNN algorithm predicts the values of new datapoints based on 'feature similarity,' which implies that the new data point will be assigned a value depending on how closely it matches the points in the training set.

KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report

KNN = KNeighborsClassifier(n_neighbors=165)
classifier = KNN.fit(x_train,y_train)

y_pred = classifier.predict(x_test)
acc = accuracy_score(y_test,y_pred)
print("Accuracy of KNN : {0:.2f}".format(accuracy_score(y_pred,y_test) * 100))
print("\nClassification report:\n")
print(classification_report(y_test, y_pred))
```

```
Accuracy of KNN : 60.79

Classification report:

              precision    recall  f1-score   support

           0       0.63      0.97      0.77       331
           1       0.56      0.69      0.61       432
           2       0.84      0.98      0.90       310
           3       0.52      0.83      0.64       245
           4       0.80      0.82      0.81       498
           5       0.84      0.60      0.70       247
           6       0.62      0.78      0.69       348
           7       0.83      0.69      0.75       436
           8       0.59      0.58      0.58       288
           9       0.48      0.34      0.40       331
          10       0.97      0.65      0.78       209
          11       0.72      0.25      0.37       394
          12       0.39      0.34      0.36       291
          13       0.96      0.62      0.75       246
          14       0.94      0.97      0.95       347
          15       0.96      0.70      0.81       164
          16       0.09      0.15      0.11       144
          17       0.38      0.44      0.41       246
          18       0.44      0.86      0.58       248
          19       0.21      0.38      0.27       266
          20       0.55      0.32      0.41       346
          21       0.37      0.40      0.38       206
          22       0.91      0.62      0.74       267
          23       0.84      0.26      0.40       332

    accuracy                           0.61      7172
   macro avg       0.64      0.59      0.59      7172
weighted avg       0.66      0.61      0.61      7172
```

Fig-6.3. K-nearest neighbour

Logistic Regression:

The outcome of a variable with a value of 0 or 1 is predicted using logistic regression. Here we have sigmoid function. The sigmoid function is a mathematical formula for converting expected values into probabilities. It converts every real value between 0 and 1 into another value. The definition of the threshold value is used in logistic regression to describe the likelihood of either 0 or 1. Values above the threshold value tend to be 1, while those below the threshold value tend to be 0.

Logistic Regression

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()

model.fit(x_train, y_train)
Logisticprediction=model.predict(x_test)

print('Accuracy of logistic regression: {0:.2f}'.format(accuracy_score(Logisticprediction,y_test) * 100))
print("\nClassification report:\n")
print(classification_report(y_test, Logisticprediction))
```

```
Accuracy of logistic regression: 66.01

Classification report:

              precision    recall  f1-score   support

           0       0.88      0.96      0.92       331
           1       0.99      0.91      0.95       432
           2       0.70      0.95      0.80       310
           3       0.81      0.76      0.78       245
           4       0.82      0.82      0.82       498
           5       0.74      0.90      0.81       247
           6       0.74      0.71      0.72       348
           7       0.76      0.71      0.73       436
           8       0.69      0.65      0.67       288
           9       0.58      0.37      0.45       331
          10       0.48      0.77      0.59       209
          11       0.68      0.61      0.64       394
          12       0.64      0.51      0.56       291
          13       0.76      0.54      0.63       246
          14       0.85      0.97      0.90       347
          15       0.62      0.78      0.69       164
          16       0.15      0.28      0.20       144
          17       0.25      0.37      0.30       246
          18       0.34      0.34      0.34       248
          19       0.60      0.46      0.52       266
          20       0.71      0.40      0.51       346
          21       0.45      0.60      0.51       206
          22       0.52      0.45      0.48       267
          23       0.69      0.57      0.63       332

    accuracy                           0.66      7172
   macro avg       0.64      0.64      0.63      7172
weighted avg       0.68      0.66      0.66      7172
```

Fig-6.4. Logistic Regression

Random Forest:

It creates a "forest" out of an ensemble of decision trees, which are commonly trained using the "bagging" method. The bagging method's basic premise is that combining several learning models improves the overall output. To generate a more accurate and reliable prediction, random forest creates numerous decision trees and blends them together. While growing the trees, the random forest adds more randomness to the model. When splitting a node, it looks for the best feature from a random subset of features rather than the most essential feature.

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

random_forest_model = RandomForestClassifier(random_state=42)
random_forest_model.fit(x_train, y_train)
rfprediction=random_forest_model.predict(x_test)

print('Accuracy of random forest: {0:.2f}'.format(accuracy_score(y_test,rfprediction) * 100))
print("\nClassification report:\n")
print(classification_report(y_test, rfprediction))
```

```
Accuracy of random forest: 81.64

Classification report:

              precision    recall  f1-score   support

           0       0.91      1.00      0.95       331
           1       0.98      0.93      0.95       432
           2       0.94      1.00      0.97       310
           3       0.83      0.98      0.90       245
           4       0.87      0.99      0.93       498
           5       0.93      0.91      0.92       247
           6       0.93      0.86      0.89       348
           7       0.99      0.94      0.96       436
           8       0.83      0.80      0.82       288
           9       0.72      0.64      0.68       331
          10       0.82      1.00      0.90       209
          11       0.88      0.70      0.78       394
          12       0.75      0.52      0.61       291
          13       0.99      0.88      0.93       246
          14       0.92      1.00      0.96       347
          15       0.91      0.99      0.95       164
          16       0.29      0.55      0.38       144
          17       0.60      0.83      0.70       246
          18       0.63      0.83      0.71       248
          19       0.64      0.59      0.62       266
          20       0.77      0.56      0.65       346
          21       0.48      0.58      0.53       206
          22       0.84      0.67      0.74       267
          23       0.91      0.61      0.73       332

    accuracy                           0.82      7172
   macro avg       0.81      0.81      0.80      7172
weighted avg       0.83      0.82      0.82      7172
```

Fig-6.5. Random Forest

Multi-layer perceptron:

It consists of three layers input layer, hidden layer and output layer. The signal to be processed is received by the input layer. The output layer is in charge of performing tasks like prediction and classification. The hidden layers can be 1 or more between the input and output layers. The backpropagation helps in training of neurons. The error will also be reduced due to updation of weights that happen in this training process and accuracy also increases. Once the model is trained it can be used to evaluate test data. Predictions are made by passing the input through the network and allowing it to produce an output that can be used as a prediction.

MLP

```
from sklearn.neural_network import MLPRegressor, MLPClassifier
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(x_train, y_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
mlpc = MLPClassifier().fit(x_train,y_train)
predict = mlpc.predict(x_test)

print('Accuracy of MLP: {:.2f}%'.format(accuracy_score(y_test,predict) * 100))
print("\nClassification report:\n")
print(classification_report(y_test, predict))
```

```
Accuracy of MLP: 81.29%

Classification report:

              precision    recall  f1-score   support

           0       0.87      1.00      0.93       331
           1       1.00      0.91      0.95       432
           2       1.00      0.93      0.96       310
           3       0.77      1.00      0.87       245
           4       0.90      0.92      0.91       498
           5       0.80      0.91      0.85       247
           6       0.94      0.88      0.91       348
           7       0.95      0.94      0.95       436
           8       0.86      0.78      0.82       288
           9       0.79      0.71      0.75       331
          10       0.95      0.89      0.92       209
          11       0.90      0.68      0.78       394
          12       0.66      0.67      0.66       291
          13       0.97      0.83      0.89       246
          14       1.00      1.00      1.00       347
          15       0.69      1.00      0.82       164
          16       0.21      0.43      0.28       144
          17       0.61      0.65      0.63       246
          18       0.71      0.73      0.72       248
          19       0.59      0.62      0.60       266
          20       0.81      0.62      0.70       346
          21       0.63      0.71      0.67       206
          22       0.76      0.69      0.72       267
          23       0.90      0.70      0.79       332

    accuracy                           0.81      7172
   macro avg       0.80      0.80      0.80      7172
weighted avg       0.83      0.81      0.82      7172
```

Fig-6.6. Multi-Layer Perceptron

Support vector machine:

SVM categorizes data points by mapping them to a high-dimensional feature space, even when the data is not otherwise linearly separable. After finding a separator between the categories, the data are converted so that the separator can be drawn as a hyperplane. Following that, fresh data features can be utilized to determine which category a new record should belong to. We used linear kernel in the model. When the data is linearly separable, that is, when it can be separated using a single line, the Linear Kernel is utilized. It is one of the most often utilized kernels. A Linear Kernel is faster than any other Kernel for training an SVM.

SVM

```
from sklearn import svm

clf = svm.SVC(kernel='linear')
clf.fit(x_train, y_train)
predict = clf.predict(x_test)
print('Accuracy of SVM: {:.2f}%'.format(accuracy_score(y_test,predict) * 100))
print("\nClassification report:\n")
print(classification_report(y_test, predict))
```

```
Accuracy of SVM: 80.54%

Classification report:

              precision    recall  f1-score   support

           0       0.81      1.00      0.89       331
           1       1.00      0.95      0.97       432
           2       0.81      1.00      0.89       310
           3       0.96      1.00      0.98       245
           4       0.95      1.00      0.97       498
           5       0.67      0.87      0.76       247
           6       0.87      0.93      0.90       348
           7       0.98      0.92      0.95       436
           8       0.72      0.76      0.74       288
           9       0.78      0.50      0.61       331
          10       0.61      0.65      0.63       209
          11       0.84      0.74      0.79       394
          12       0.82      0.64      0.72       291
          13       0.89      0.70      0.78       246
          14       0.92      1.00      0.96       347
          15       0.93      1.00      0.96       164
          16       0.35      0.72      0.47       144
          17       0.62      0.66      0.64       246
          18       0.84      0.68      0.75       248
          19       0.61      0.63      0.62       266
          20       0.84      0.67      0.75       346
          21       0.78      0.70      0.74       206
          22       0.56      0.53      0.54       267
          23       0.86      0.73      0.79       332

    accuracy                           0.81      7172
   macro avg       0.79      0.79      0.78      7172
weighted avg       0.82      0.81      0.81      7172
```

Fig-6.7. Support Vector Machine

Ensemble Learning:

Ensemble learning is a general meta-approach to machine learning that combines predictions from different models to improve predictive performance. We used voting classifier in ensemble learning. A voting classifier is a classification approach that makes predictions using many classifiers. It's especially useful when a data scientist or machine learning engineer isn't sure which categorization algorithm to apply. As a result, the voting classifier produces predictions based on the most frequent of the predictions from several classifiers.

ENSEMBLE

```
from sklearn import model_selection
from sklearn.ensemble import VotingClassifier

kfold = model_selection.KFold(n_splits=10)

estimators = []

estimators.append(('SVM', clf))
estimators.append(('Random Forest', random_forest_model))
estimators.append(('Logistic regression', model))
estimators.append(('KNN', classifier))
ensemble = VotingClassifier(estimators)

ensemble.fit(x_train, y_train)
y_test_pred = ensemble.predict(x_test)
```

```
print(f"\nAccuracy of Ensemble: {accuracy_score(y_test, y_test_pred) * 100 :.2f}%")
print(f"\nClassification report:\n{classification_report(y_test, y_test_pred)}")
```

```
Accuracy of Ensemble: 79.34%

Classification report:
              precision    recall  f1-score   support

           0       0.77      1.00      0.87       331
           1       0.96      0.95      0.96       432
           2       0.83      1.00      0.91       310
           3       0.88      1.00      0.93       245
           4       0.88      0.99      0.93       498
           5       0.80      0.91      0.85       247
           6       0.84      0.93      0.89       348
           7       0.97      0.86      0.91       436
           8       0.76      0.82      0.79       288
           9       0.68      0.57      0.62       331
          10       0.75      0.87      0.80       209
          11       0.87      0.72      0.79       394
          12       0.82      0.59      0.69       291
          13       0.99      0.74      0.84       246
          14       0.98      1.00      0.99       347
          15       0.96      0.99      0.98       164
          16       0.28      0.58      0.37       144
          17       0.63      0.64      0.64       246
          18       0.63      0.78      0.70       248
          19       0.57      0.54      0.55       266
          20       0.76      0.56      0.65       346
          21       0.52      0.51      0.52       206
          22       0.78      0.54      0.64       267
          23       0.91      0.59      0.71       332

    accuracy                           0.79      7172
   macro avg       0.78      0.78      0.77      7172
weighted avg       0.81      0.79      0.79      7172
```

Fig-6.8. Ensemble Learning

The combination of Support Vector Machine, Random Forest, Logistic Regression and K-nearest neighbors has been done in ensemble learning

Convolutional Neural Network:

The first two hidden layers are convolution and max pooling layers. The first hidden layer is made up of a number of nodes, each of which receives a weighted sum of the 784 input values. After that, the weighted total of the inputs is fed into an activation function. Negative inputs will be set to 0 by the Relu. The current Relu's outputs will be the Next hidden layer's inputs. Flatten and dense layers are in charge of reducing data to one dimension and determining the class of an image. Select an acceptable value for the number of epochs to improve the model's performance. Each epoch updates the weights applied to each input, which are learned during the training phase.

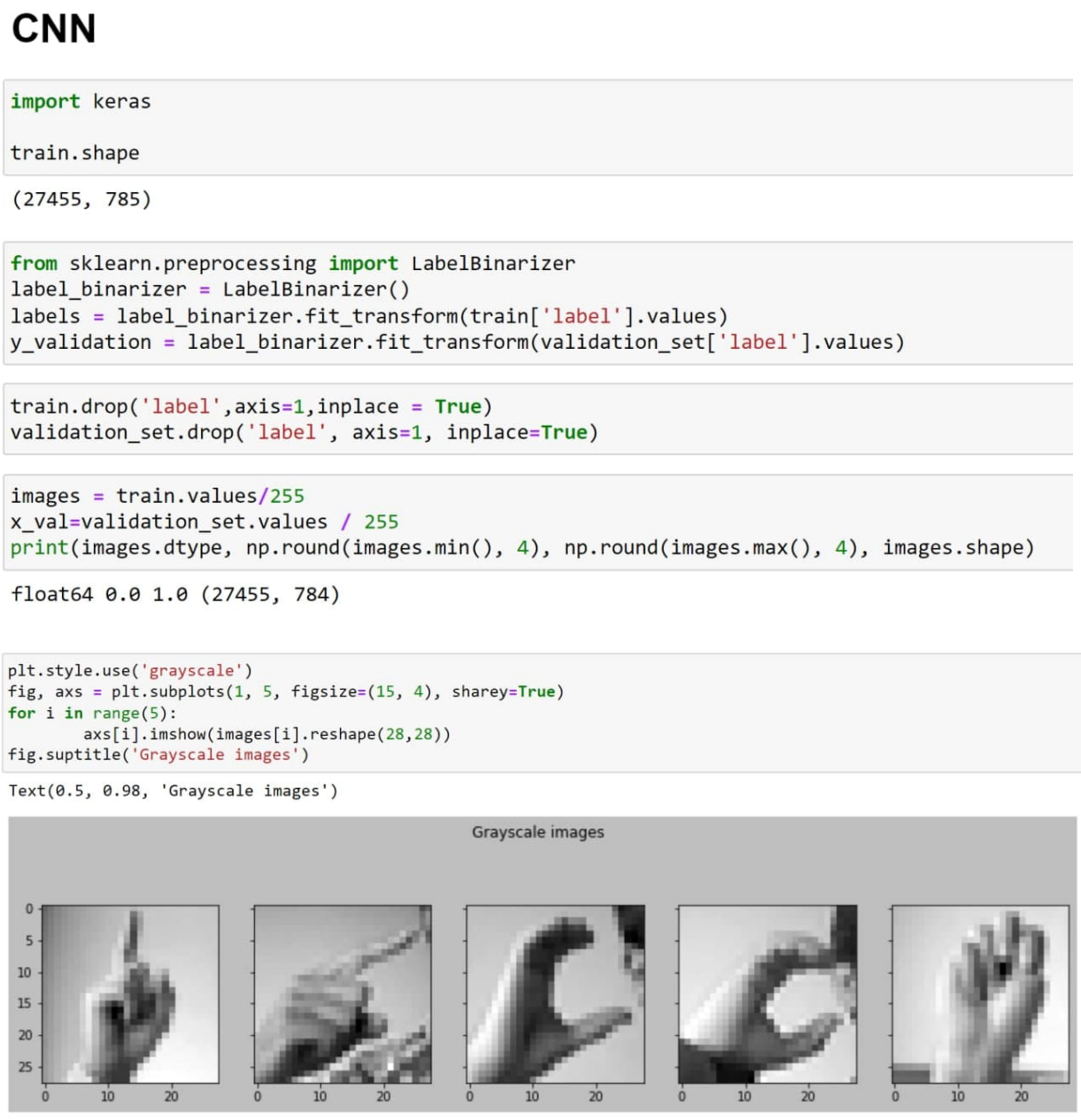

Fig-6.9. Convolutional Neural Network

Converting multi class labels to binary labels and plotting grayscale images has been done in the above figure.

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(images, labels, test_size = 0.3, random_state = 1)
```

```
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
print(x_val.shape[0], 'val samples')
```

```
19218 train samples
8237 test samples
7172 val samples
```

```
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
x_val = x_val.reshape(x_val.shape[0], 28, 28, 1)
```

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential()

model.add(Conv2D(16, kernel_size=(4,4), activation = 'relu', input_shape=(28, 28 ,1), padding='same' ))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(32, kernel_size=(4,4), activation = 'relu', input_shape=(28, 28 ,1), padding='same' ))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(64, kernel_size=(4,4), activation = 'relu', input_shape=(28, 28 ,1), padding='same' ))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(24, activation='softmax'))
model.summary()
```

Fig-6.10. Splitting into train and test data

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 16)        272
_________________________________________________________________
dropout (Dropout)            (None, 28, 28, 16)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 32)        8224
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 64)          32832
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 500)               288500
_________________________________________________________________
dropout_3 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_1 (Dense)              (None, 24)                12024
=================================================================
```

Fig-6.11. Implementation of CNN algorithm

```
model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
                  metrics=['accuracy'])
```

```
from keras.callbacks import ModelCheckpoint

checkpointer = ModelCheckpoint(filepath='model.weights.best.hdf5', verbose=1,
                               save_best_only=True)
hist = model.fit(x_train, y_train, batch_size=32, epochs=20,
          validation_data=(x_val, y_validation), callbacks=[checkpointer],
          verbose=2, shuffle=True)
```

Fig-6.12. Training model

```
Epoch 1/20

Epoch 00001: val_loss improved from inf to 0.56880, saving model to model.weights.best.hdf5
601/601 - 24s - loss: 1.1121 - accuracy: 0.6447 - val_loss: 0.5688 - val_accuracy: 0.7932
Epoch 2/20

Epoch 00002: val_loss improved from 0.56880 to 0.25907, saving model to model.weights.best.hdf5
601/601 - 24s - loss: 0.1424 - accuracy: 0.9511 - val_loss: 0.2591 - val_accuracy: 0.9172
Epoch 3/20

Epoch 00003: val_loss improved from 0.25907 to 0.23938, saving model to model.weights.best.hdf5
601/601 - 27s - loss: 0.0519 - accuracy: 0.9817 - val_loss: 0.2394 - val_accuracy: 0.9364
Epoch 4/20

Epoch 00004: val_loss improved from 0.23938 to 0.17415, saving model to model.weights.best.hdf5
601/601 - 26s - loss: 0.0340 - accuracy: 0.9890 - val_loss: 0.1742 - val_accuracy: 0.9492
Epoch 5/20

Epoch 00005: val_loss improved from 0.17415 to 0.13759, saving model to model.weights.best.hdf5
601/601 - 23s - loss: 0.0238 - accuracy: 0.9922 - val_loss: 0.1376 - val_accuracy: 0.9569
Epoch 6/20

Epoch 00006: val_loss did not improve from 0.13759
601/601 - 24s - loss: 0.0192 - accuracy: 0.9941 - val_loss: 0.1568 - val_accuracy: 0.9550
Epoch 7/20

Epoch 00007: val_loss improved from 0.13759 to 0.12663, saving model to model.weights.best.hdf5
601/601 - 27s - loss: 0.0175 - accuracy: 0.9946 - val_loss: 0.1266 - val_accuracy: 0.9632
Epoch 8/20

Epoch 00008: val_loss did not improve from 0.12663
601/601 - 23s - loss: 0.0154 - accuracy: 0.9947 - val_loss: 0.3839 - val_accuracy: 0.9389
Epoch 9/20

Epoch 00009: val_loss did not improve from 0.12663
601/601 - 22s - loss: 0.0165 - accuracy: 0.9959 - val_loss: 0.2115 - val_accuracy: 0.9499
Epoch 10/20

Epoch 00010: val_loss did not improve from 0.12663
601/601 - 22s - loss: 0.0128 - accuracy: 0.9961 - val_loss: 0.1668 - val_accuracy: 0.9573


Epoch 11/20

Epoch 00011: val_loss did not improve from 0.12663
601/601 - 23s - loss: 0.0108 - accuracy: 0.9966 - val_loss: 0.1347 - val_accuracy: 0.9591
Epoch 12/20

Epoch 00012: val_loss did not improve from 0.12663
601/601 - 22s - loss: 0.0123 - accuracy: 0.9966 - val_loss: 0.1465 - val_accuracy: 0.9600
Epoch 13/20

Epoch 00013: val_loss did not improve from 0.12663
601/601 - 22s - loss: 0.0112 - accuracy: 0.9969 - val_loss: 0.1376 - val_accuracy: 0.9621
Epoch 14/20

Epoch 00014: val_loss did not improve from 0.12663
601/601 - 23s - loss: 0.0122 - accuracy: 0.9970 - val_loss: 0.1600 - val_accuracy: 0.9490
Epoch 15/20

Epoch 00015: val_loss did not improve from 0.12663
601/601 - 22s - loss: 0.0082 - accuracy: 0.9982 - val_loss: 0.2020 - val_accuracy: 0.9611
```

Fig-6.13. Number of epochs in training model

```
model.load_weights('model.weights.best.hdf5')
```

```
test=validation_set
test_images = test.values/255
test_images = np.array([np.reshape(i, (28, 28)) for i in test_images])
test_images = np.array([i.flatten() for i in test_images])
test_labels = y_validation
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1)


test_images.shape
```

```
(7172, 28, 28, 1)
```

```
y_pred = model.predict(test_images)
from sklearn.metrics import accuracy_score,classification_report
y_pred = y_pred.round()
print('Accuracy : {:.2f}%'.format(accuracy_score(test_labels,y_pred) * 100))
print("\nClassification report:\n")
print(classification_report(test_labels,y_pred))
```

```
Accuracy : 97.39%
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       331
           1       1.00      1.00      1.00       432
           2       0.94      1.00      0.97       310
           3       1.00      0.94      0.97       245
           4       0.96      1.00      0.98       498
           5       1.00      1.00      1.00       247
           6       0.90      1.00      0.95       348
           7       1.00      1.00      1.00       436
           8       1.00      1.00      1.00       288
           9       1.00      0.94      0.97       331
          10       0.93      1.00      0.97       209
          11       1.00      0.97      0.98       394
          12       0.95      0.94      0.95       291
          13       1.00      0.92      0.96       246
          14       1.00      1.00      1.00       347
          15       1.00      1.00      1.00       164
          16       0.80      1.00      0.89       144
          17       0.99      0.91      0.95       246
          18       1.00      0.76      0.86       248
          19       0.94      1.00      0.97       266
          20       1.00      0.94      0.97       346
          21       0.98      1.00      0.99       206
          22       1.00      1.00      1.00       267
          23       0.95      1.00      0.97       332

   micro avg       0.97      0.97      0.97      7172
   macro avg       0.97      0.97      0.97      7172
weighted avg       0.98      0.97      0.97      7172
 samples avg       0.97      0.97      0.97      7172
```

Fig-6.14. Prediction and Accuracy

In the method of OpenCV we have used image processing techniques such as Gaussian blur, threshold and Convex Hull.

Gaussian Blur: A Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image with a Gaussian function in image processing (named after mathematician and scientist Carl Friedrich Gauss).

It's a common effect in graphics software for reducing visual noise and reducing detail. This blurring technique produces a smooth blur similar to that seen when viewing an image through a translucent screen, as opposed to the bokeh effect generated by an out-of-focus lens or the shadow cast by an object under normal lighting. In computer vision methods, Gaussian smoothing is also employed as a pre-processing stage to improve image structures at different sizes (see scale space representation and scale space implementation).

Threshold: The situation is straightforward in this case. The same threshold value is used for each pixel. If the pixel value is less than the threshold, it is set to 0, otherwise a maximum value is used. The thresholding is done with the cv.threshold function. The source image, which should be a grayscale image, is the first argument. The threshold value is the second input, and it is used to categories the pixel values. The maximum value assigned to pixel values surpassing the threshold is the third input. The fourth argument of the function in OpenCV enables multiple methods of thresholding.

Convex Hull: An item that has no interior angles higher than 180 degrees is said to be convex. Non-Convex or Concave refers to a shape that is not convex. The exterior or shape of an object is referred to as the hull.

As a result, a shape's or a group of points' Convex Hull is a tightly fitting convex boundary around the points or the shape. A convex object's Convex Hull is simply its boundaries. A concave shape's Convex Hull is the convex boundary that encloses it the most closely.

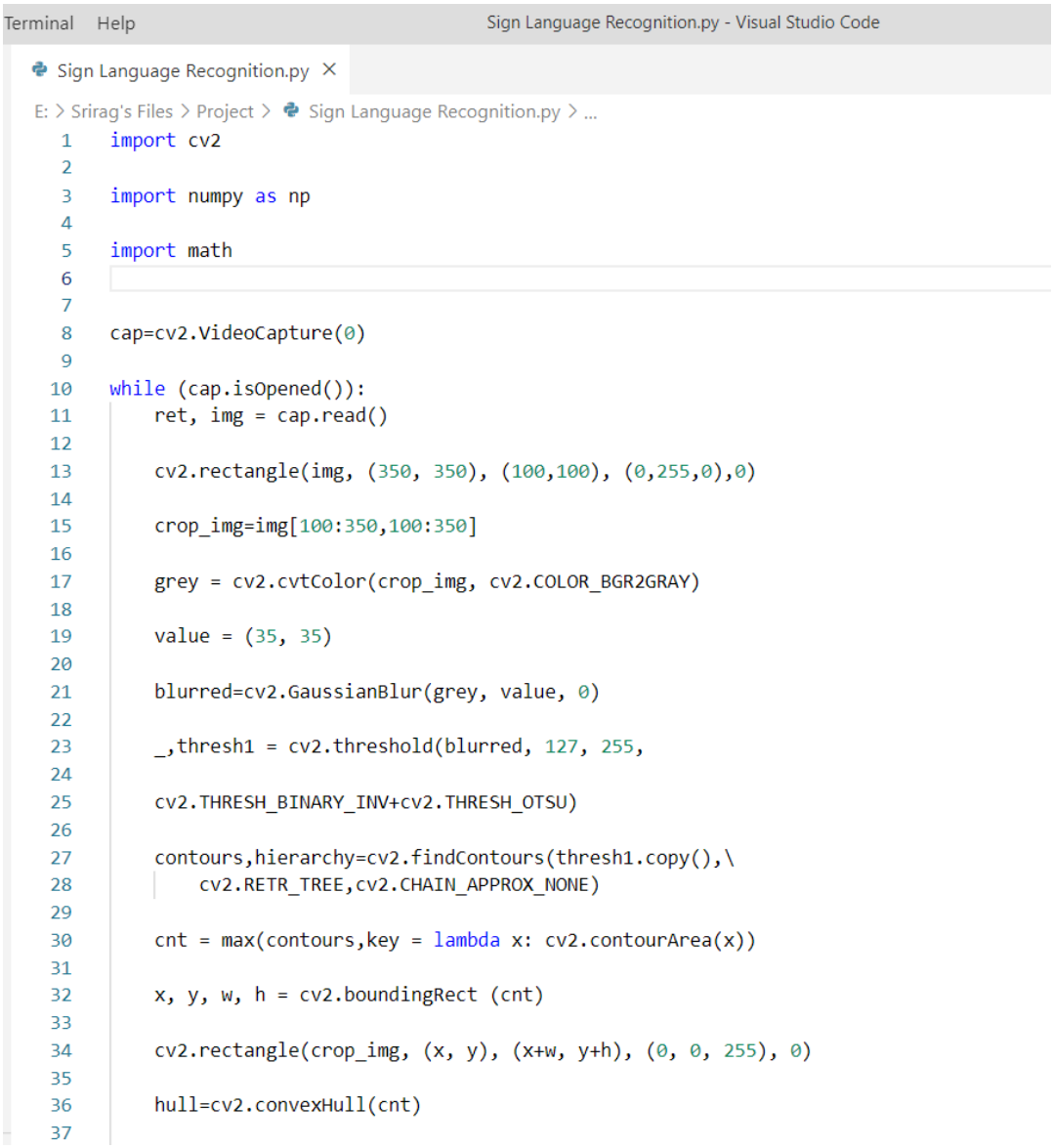

Fig-6.15. Capturing Image and applying gaussian blur, threshold and convex hull

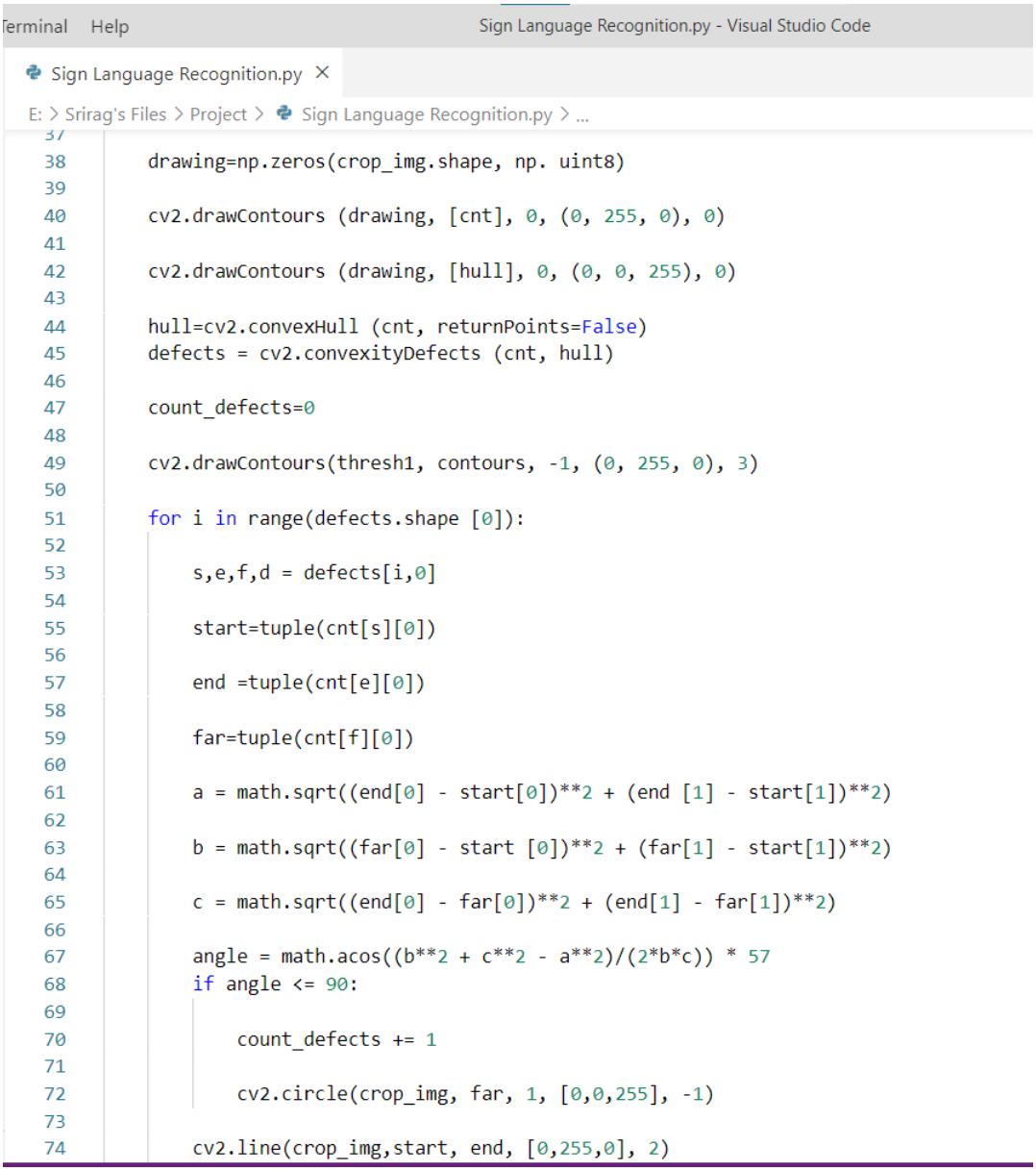

Fig-6.16. Drawing contours and identifying hand gesture

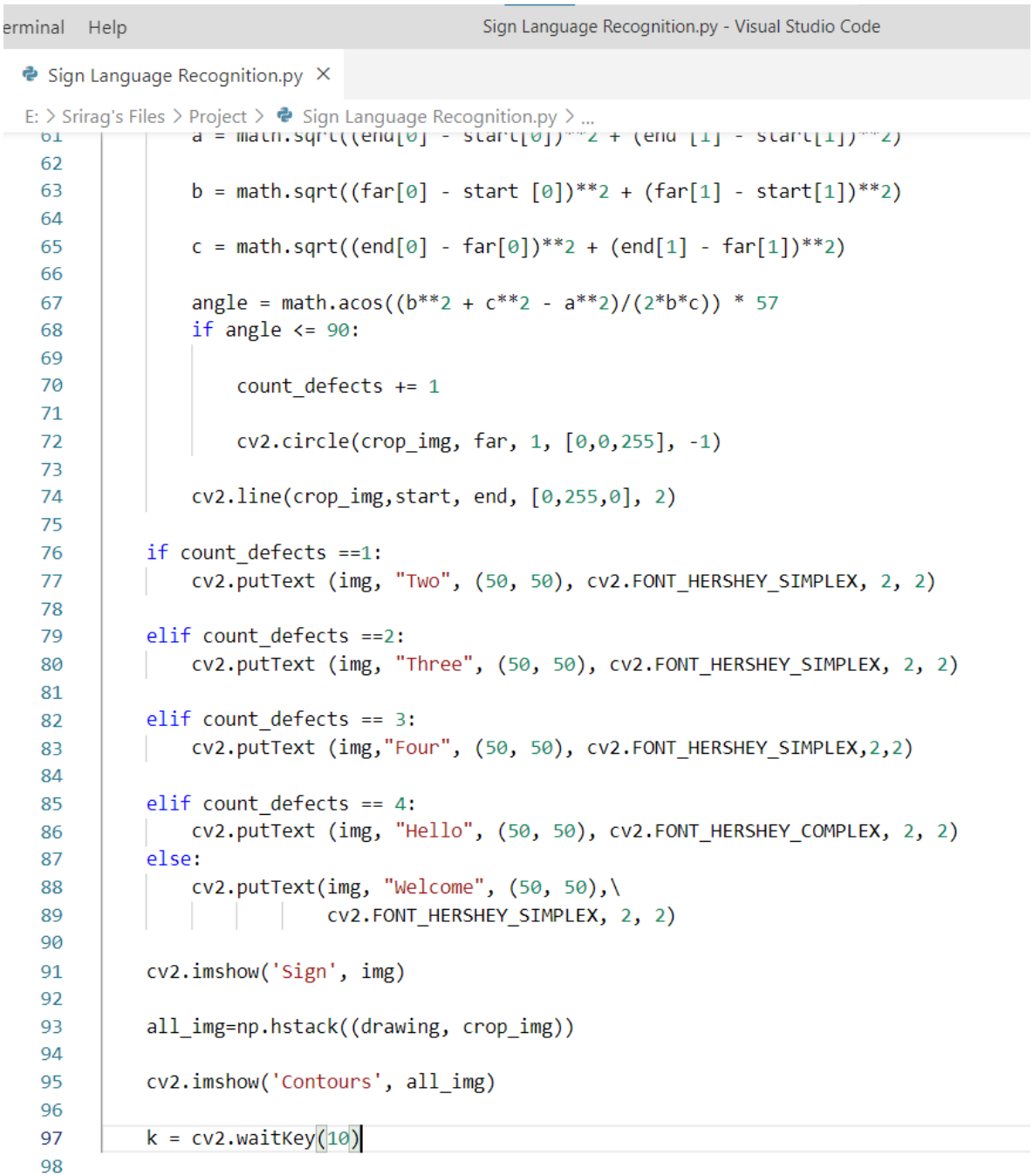

Fig-6.17. Number of hand gestures to be recognized.

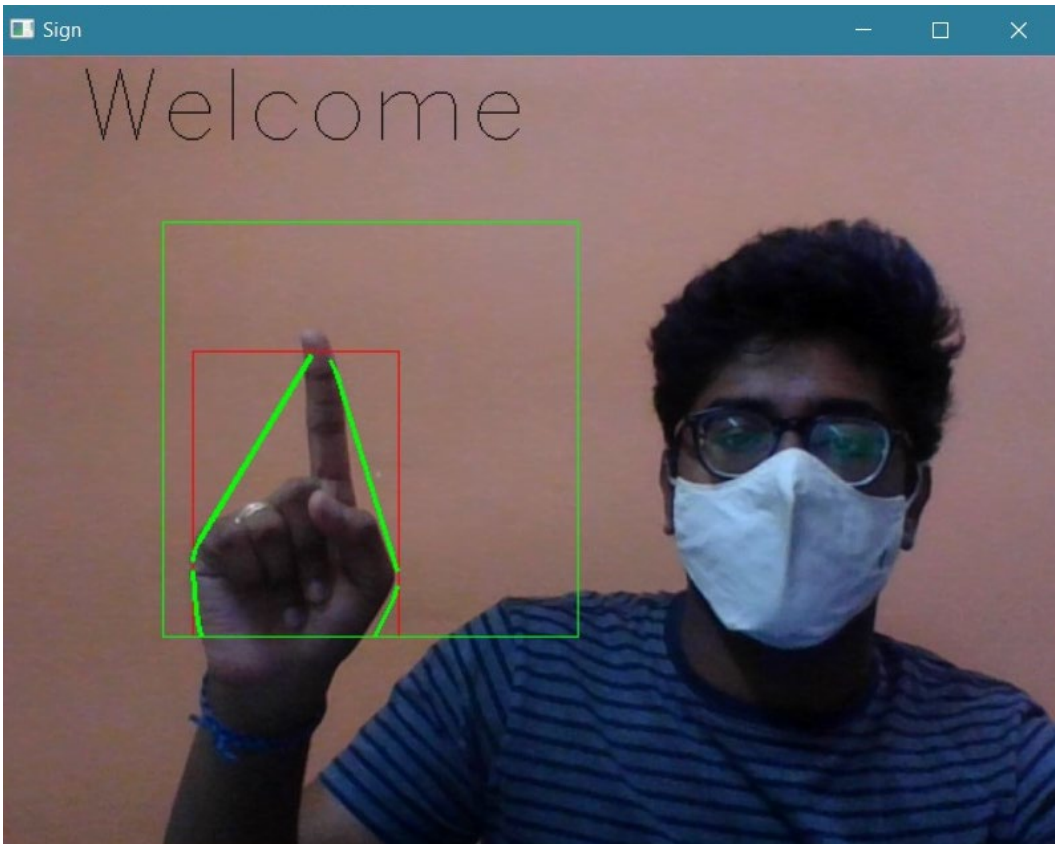

Fig-6.18. Gesture-1

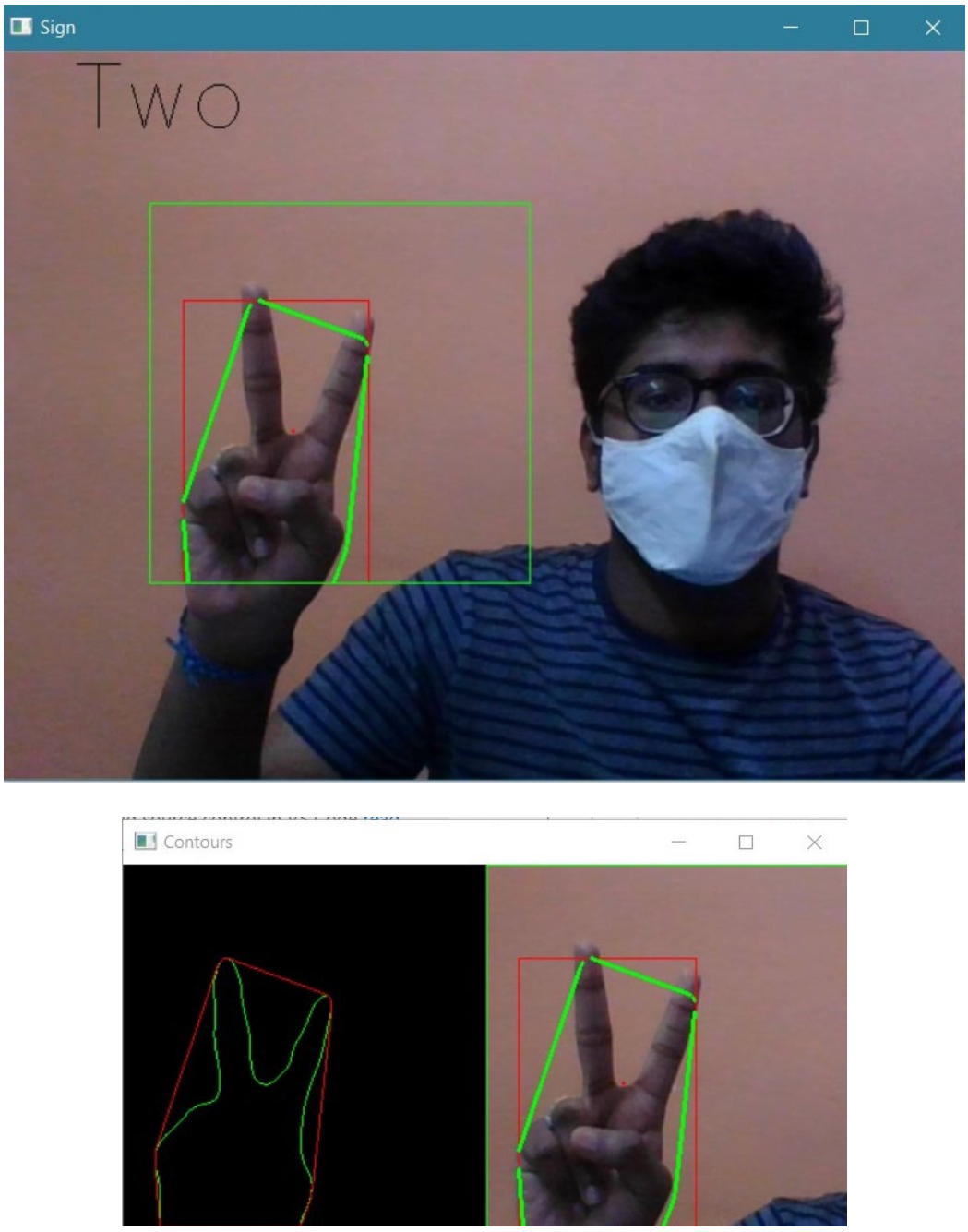

Fig-6.19. Gesture-2

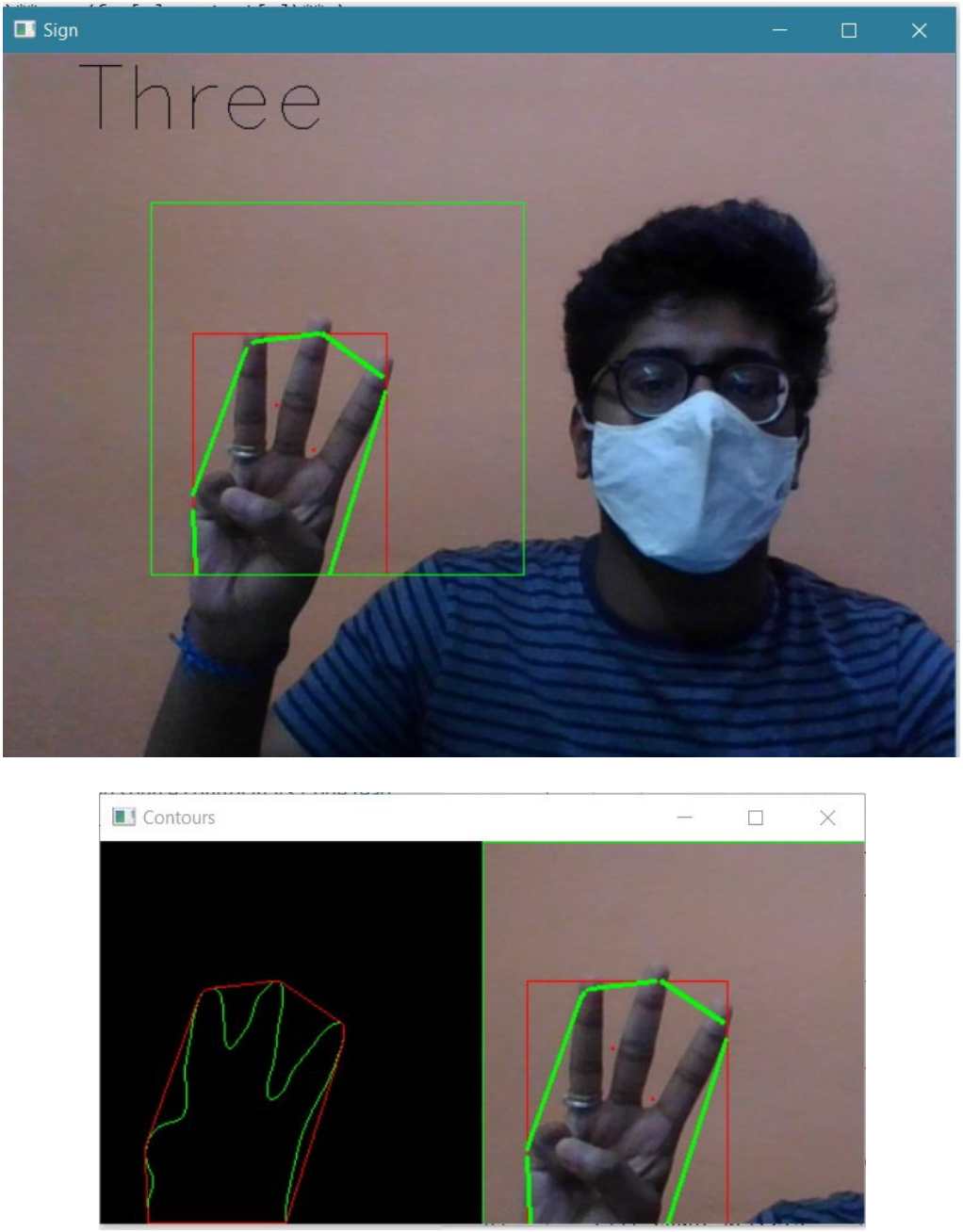

Fig-6.20. Gesture-3

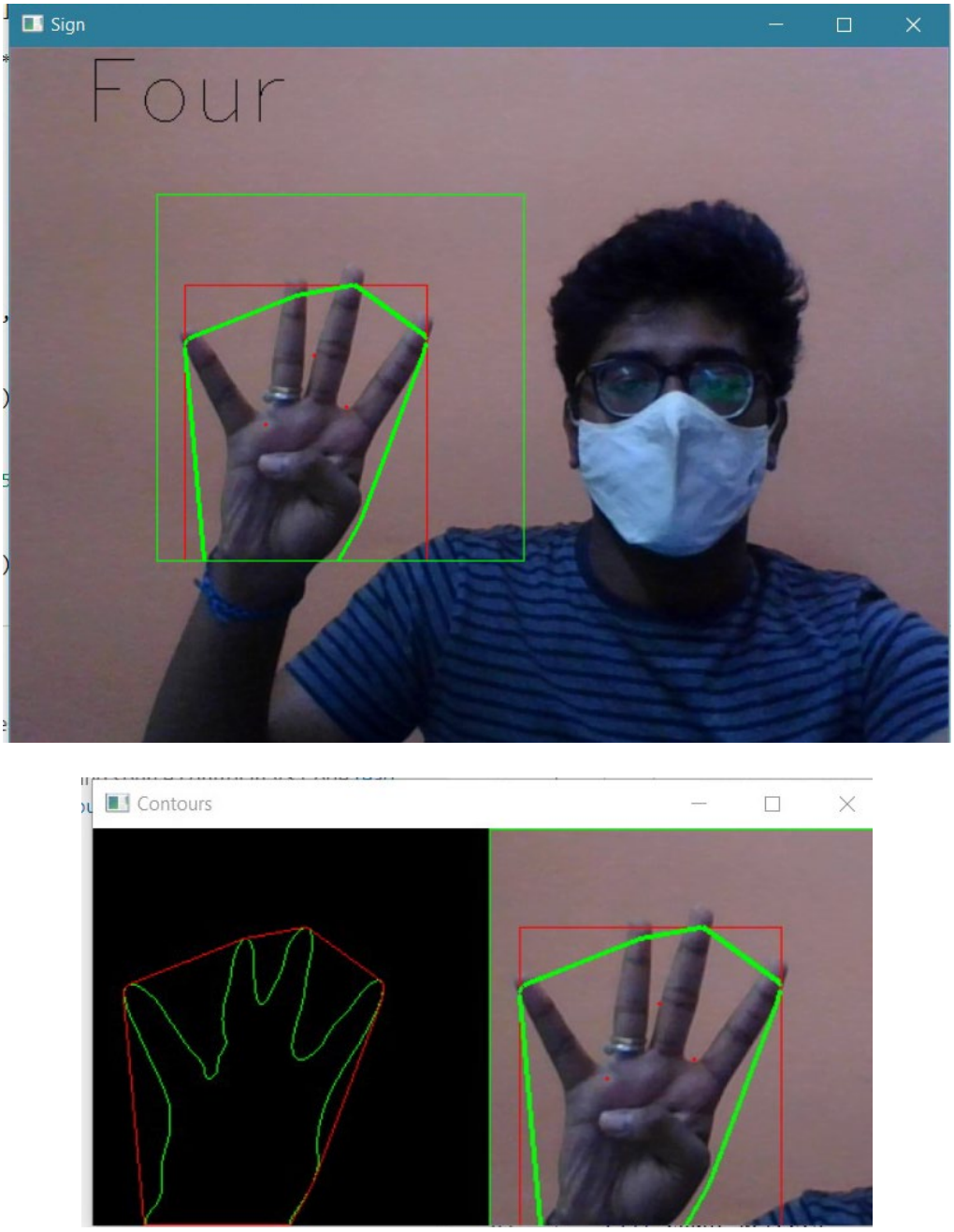

Fig-6.21. Gesture-4

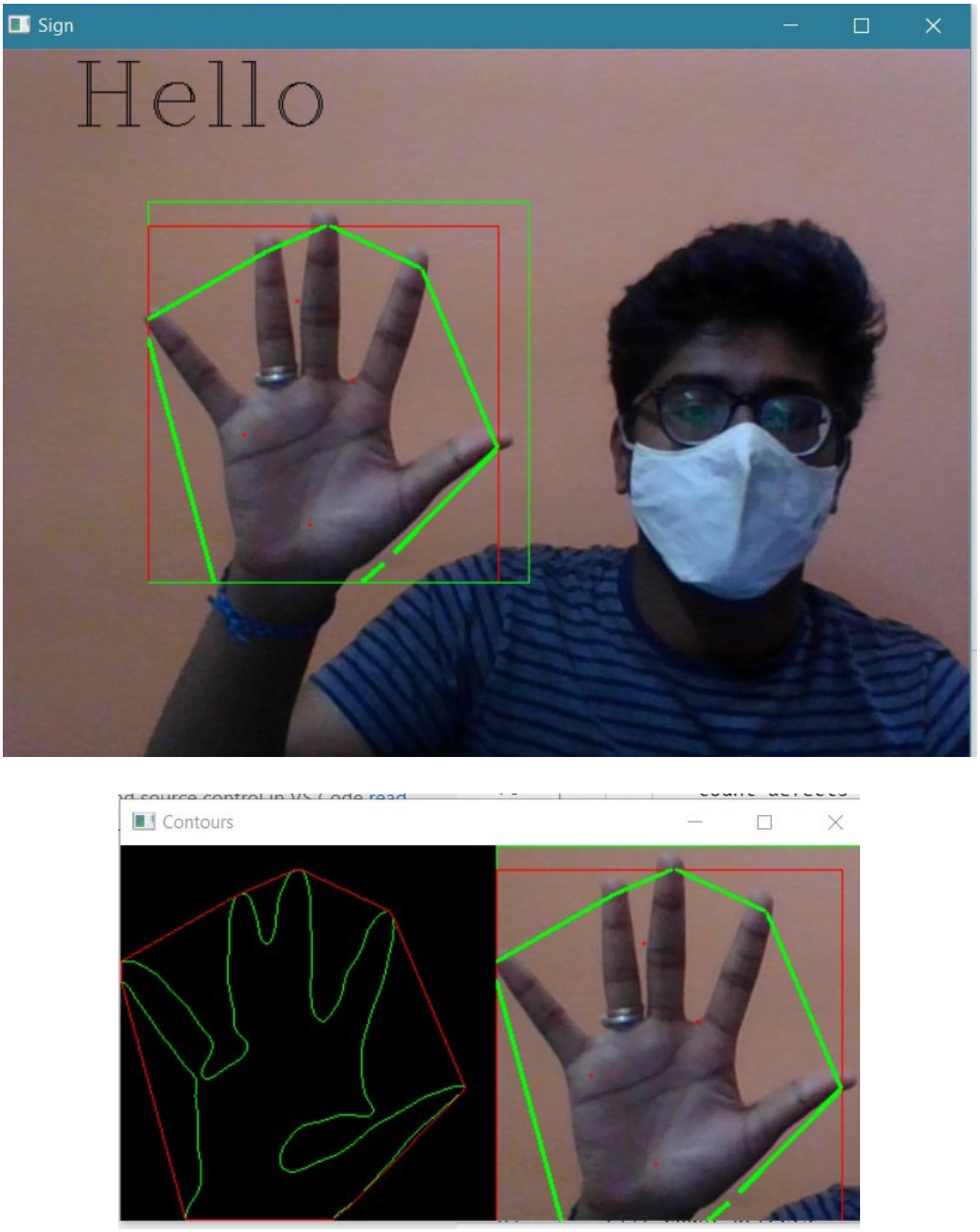

Fig-6.22. Gesture-5

Procedure:

- Capturing hand gestures from camera and apply Gaussian blur
- Change the color space to HSV from BGR.
- Make a binary image with skin tones in white and the rest in black.
- To filter out the background noise, use morphological transformations.
- Apply Gaussian Blur and Threshold to your image.
- Draw a contour of the convex hull.
- For all flaws demonstrating the meaning of hand gestures, use the cosine method to get the angle of the far point from the start and end point, i.e. the convex points (the finger tips).
- Display the meaning of hand gesture.

Metrics:

MODEL	ACCURACY
K-NEAREST NEIGHBOUR	60.79%
LOGISTIC REGRESSION	66.01%
RANDOM FOREST	81.64%
MLP CLASSIFIER	81.29%
SUPPORT VECTOR MACHINE	80.54%
ENSEMBLE	79.34%
CNN	97.39%

Table 6.1. Metrics

CHAPTER 7

RESULTS DISCUSSION

The best accuracy is achieved by deep learning algorithm which is Convolutional Neural Network (CNN) with 97.39%. The accuracies of machine learning algorithms are K-nearest neighbours-60.79%, Logistic Regression-66.01%, Random Forest-81.64%, Multi-Layer Perceptron-81.29%, Support vector machine-80.54%, Ensemble Learning-79.34%..

- Applied Multi-Layer perceptron (MLP) on the MNIST data set and obtained an accuracy of 81.29%.
- Scaling down of data has been done and increase in accuracy of models compared to research papers.
- Performed Ensemble learning

CHAPTER 8

CONCLUSION & FUTURE WORK

This project's main goal is to recognize sign language (hand gestures). On the sign language MNIST dataset, we've been able to successfully introduce and get accuracies for machine learning and deep learning methods. We've also used OpenCV and gaussian blur techniques to recognize the shape of a hand gesture through a camera and deduce its meaning. On the MNIST dataset, the convolutional neural network (CNN) had the maximum accuracy of 97.39%.Sign language recognition benefits are : Deaf and dumb people, like anyone else, can communicate their feelings and thoughts with or without words, Gestures can be used to control robots and operate various devices, We can change the channels on the television using hand gestures, Virtual reality interaction uses hand gestures to manipulate virtual actions using one or two hands for 2D and 3D interaction displays, Geometry architecture for dimensionality: Easily view various 3D designs from various angles (Eg : AutoCad).

We used OpenCV in conjunction with image processing techniques to deduce the meaning of sign language. To achieve more exact and efficient recognition of hand gestures with accurate meanings, we will employ more image processing techniques and recognition of more number of hand gestures using OpenCV will be done.

REFERENCES

[1] Singha, Joyeeta & Das, Karen. (2013). Indian Sign Language Recognition Using Eigen Value Weighted Euclidean Distance Based Classification Technique. International Journal of Advanced Computer Science and Applications. 4. 10.14569/IJACSA.2013.040228.

[2] De Souza, Cesar & Pizzolato, Ednaldo. (2013). Sign Language Recognition with Support Vector Machines and Hidden Conditional Random Fields: Going from Fingerspelling to Natural Articulated Words. 7988. 84-98. 10.1007/978-3-642-39712-7_7.

[3] Mrs. Dipali Rojasara, Dr. Nehal G. Chitaliya, " Indian sign language recognition", International journal of engineering research and technology (IJERT), ISSN: 2278-0181, Vol:2, Issue 10 October 2013.

[4] Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 370–376, Baltimore, Maryland, USA, June 23-25 2014. c 2014 Association for Computational Linguistics.

[5] Yang, H.-D. Sign Language Recognition with the Kinect Sensor Based on Conditional Random Fields. *Sensors* **2015**, *15*, 135-147.

[6] Sanil Jain(12616) K.V.Sameer Raja(12332) Mentor - Prof.Amitabha Mukerjee,Indian Sign Language Character Recognition, Indian Institute of Technology, Kanpur March 16, 2015

[7] Fatmi, Rabeet & Rashad, Sherif & Integlia, Ryan & Hutchison, Gabriel. (2017). American Sign Language Recognition using Hidden Markov Models and Wearable Motion Sensors.

[8] Hakim, N.L.; Shih, T.K.; Kasthuri Arachchi, S.P.; Aditya, W.; Chen, Y.-C.; Lin, C.-Y. Dynamic Hand Gesture Recognition Using 3DCNN and LSTM with FSM Context-Aware Model. *Sensors* **2019**, *19*, 5429.

[9] Hosain, Al Amin, Panneer Selvam Santhalingam, Parth Pathak, Jana Kosecka, and Huzefa Rangwala. "Sign Language Recognition Analysis using Multimodal Data." Statistics 2019.1909 (2019).

[10] Yang, L.; Chen, J.; Zhu, W. Dynamic Hand Gesture Recognition Based on a Leap Motion Controller and Two-Layer Bidirectional Recurrent Neural Network. Sensors 2020, 20, 2106

[11] https://www.kaggle.com/datamunge/sign-language-mnist

[12] Deepika Tewari and Sanjay Kumar Srivastava. A Visual Recognition of Static Hand Gestures in Indian Sign Language based on Kohonen Self- Organizing Map Algorithm,