SIGN LANGUAGE ALPHABET RECOGNITION

A MINI PROJECT REPORT

Submitted to University of Mumbai

In

COMPUTER ENGINEERING

Submitted By

JANVI D. PATIL

SUJAY S. POOKKATTUPARAMBIL

NIDHI S. GOSAVI

Under the guidance of

Prof. JIGNASHA SOSA

DEPARTMENT OF COMPUTER ENGINEERING

PADMABHUSHAN VASANTDADA PATIL PRATISHTHAN'S

COLLEGE OF ENGINEERING

SION, MUMBAI-400022

UNIVERSITY OF MUMBAI

2019-2020

PADMABHUSHAN VASANTDATA PATIL PRATISHTHAN’S COLLEGE OF ENGINEERING SION, MUMBAI-400022

CERTIFICATE

This is to certify that the mini project work, entitled SIGN LANGUAGE ALPHABET RECOGNITION has completed successfully in Computer Engineering by

Class: TE Division: A

Roll No: Seat No:

JANVI D. PATIL	(VU1F1718004)
SUJAY S. POOKKATTUPARAMBIL	(VU1F1718017)
NIDHI S. GOSAVI	(VU1F1718019)

Prof. JIGNASHA SOSA

Mini project Guide

Department of Computer Engineering

PADMABHUSHAN VASANTDATA PATIL PRATISHTHAN'S

COLLEGE OF ENGINEERING

SION, MUMBAI-400022

MINI PROJECT APPROVAL CERTIFICATE

This is to certify that the mini project work of semester 6, entitled SIGN LANGUAGE ALPHABET RECOGNITION is approved in Computer Engineering completed by

JANVI D. PATIL
SUJAY S. POOKKATTUPARAMBIL
NIDHI S. GOSAVI

Internal Examiner	External Examiner
Name	Name
Date	Date

Mini project Convener Head of Department Principal

ABSTRACT

Very few people understand sign language. Moreover, contrary to popular belief, it is not an international language. Obviously, this further complicates communication between the deaf community and the hearing majority. The alternative of written communication is cumbersome, because the deaf community is generally less skilled in writing a spoken language. For example, when an accident occurs, it is often necessary to communicate quickly with the emergency physician where written communication is not always possible. The purpose of this work is to contribute recognizing Indian sign language to the field of automatic sign language recognition with maximum efficiency. This paper focuses on the recognition of static gestures of ISL. The most challenging part in the design of an automatic sign language translator is the design of a good classifier that can classify the input static gestures with high accuracy. In the proposed system, design of classifier for sign languages recognition uses CNN architecture. The system trained CNNs for the classification of 24 alphabets using 3983 images. The result shows that accuracy improves as we include more data from different subjects during training.

ACKNOWLEGEMENT

First of all, we would like to express a deep sense of gratitude to our project guide **Prof. Jignasha Sosa** who gave her whole hearted support and guidance and being interested during process of our project. Without his inspiration and encouragement, the completion of project would have been difficult task. We would like to recall and thank all those who have directly and indirectly contributed to its success.

We would like to thank our Head of Department of Computer Engineering, **Prof. (Dr.) Mahavir Devmane** for supervising all the activities and for lending her full co-ordination, despite her busy schedule, in successful deployment of this project. We feel privileged to express our deepest sense of gratitude and sincere thanks to our Principal, **Dr. Alam N. Shaikh** for giving us an opportunity.

We also express our profound gratitude to Management of Padmabhushan Vasantdada Patil Pratishthan's College of Engineering for giving us the opportunity and facilitates to accomplish our project work we also express our devoted and sincere thanks to all our staff member as well as the Teachers of Information Technology Department who have offering valuable suggestion and constructive improvisation. We own a substantial share of our success to the whole faculty and staff members, who provided us with the requisite facilities to complete the project work. Lastly, we extend a sense of gratitude and love to our friends and family for their support, strength and help in everything.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction:

In this mini project we are implementing Sign Regognition Language. In our research, we look at Indian Sign Language (ISL), which is used in the India and different countries have different gestures in their sign language. There are 22 handshapes that correspond to the 26 letters of the alphabet, and you can sign the 10 digits on one hand.In this research we have used image processing techniques to distinguish the hand from background and then identify the fingertips to determine the gestures. This project understands the gestures and show it in text form.

1.2 Aim and Objectives:

This Project primarily targets on interpreting single alphabets of a sign language into text or speech, so as to facilitate the communication between deaf-mute people and ordinary people. This project will help everyone understand what people who are unable to speak are meaning. This helps them to mix in with the regular crowd.

1.3 Motivation for the Work:

People always wonder what do each of the signs mean or what does the person who is unable to speak mean. One day while returning home from college we saw some dumb people interacting each other and enjoying themselves. We started to discuss what do their actions mean and what they were trying to say. This drove us to create this project.

1.4 Need of New System:

The traditional system requires a person who knows and understands Sign Language and is able to translate it to regular speech for people to understand and for dumb and deaf people to understand our language as well. This is not always possible, hence this a new system is needed.

Chapter 2

Literature Survey

2.1 Introduction:

There are two approaches for sign recognition vision based and sensor-based gesture recognition based on past research.

Lots of study has been done on sensor-based approaches like gloves, wires, helmets etc. but due to disadvantage of wearing it continuously it is not possible to implement it, therefore further work is concentrated on Image based approaches.

The authors of [1] proposed that sign language learning method be divided into two parts. The former part is the ROI segmentation through an object detection network and the latter part is the sign language learning through a classification network.

The authors of [2] propose the acquisition of data by interfacing a camera followed by image preprocessing and image enhancement and segmentation, color-filtering, noise removal and thresholding. Then Blob-detection is applied followed by contour-detection.

The authors of [3] designed our multi stage CNN model by acquiring knowledge. The model is constructed with input layer, four convolutional layers, five rectified linear units (ReLu), two stochastic pooling layers, one dense and one SoftMax output layer.

2.2 Problem Definition:

There is no such rock-solid approach currently implemented for translation of Sign Language to English alphabets and there are hardly a few projects and papers of the same. So, the current project has to be capable of translating a few characters with some amount of accuracy.

Chapter 3

Design and Implementation

3.1 Requirement Gathering :

3.1.1 Hardware Requirements:

A computer with moderate/high specifications like high CPU and high GPU are required for the building of machine learning models/convolutional neural networks. Hence the following hardware requirements are recommend.

Quad-Core Processor	Intel Core i5 or AMD Ryzen 5 or greater
RAM	16GB
Hard Disk	40Gb Free Space
GPU	AMD RX 480 (8Gb) or Nvidia GTX 1060 (6Gb) or greater
Devices	Laptop with a webcam or a SmartPhone

Table 3.1.1:- Hardware Requirements

3.1.2 Software Requirements:

For building the machine learning models/convolutional neural networks, notebooks are required and other IDEs are required as well. Hence the following software requirements are recommend.

Operating System	Linux (Ubuntu 18.04) or Windows 10
IDEs	Jupyter Notebooks, Google Colaboratory
Programming Languages	Python
Frameworks	TensorFlow

Table 3.1.2:- Software Requirements

3.2 Implemented System

The implementation of the project can be mainly divided into the following steps:-

Dataset

Image Pre-processing

Model Building and Architecture

Model Evaluation

Predicting for new images

These steps were distributed among the group members in a way that would give the most effective results.

Dataset

For training the model, we have used a dataset with 3983 images and 24 classes. Each class has about 230 images with some classes having lesser images. It contains images of signs for each alphabet from A-Z except J and Z since both of them require temporal data as well which our current model is incapable of doing. Hence, we only have 24 classes instead of 26 classes.

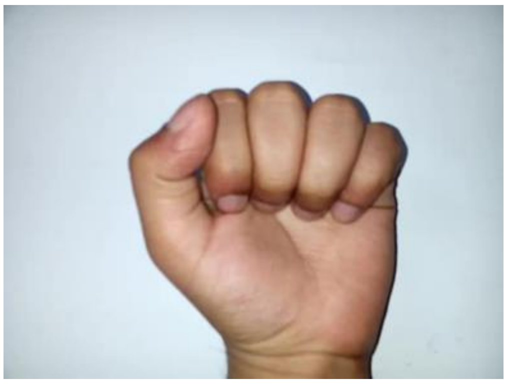


Fig 3.2.1:- Sign Language Gesture for Letter A from the dataset

Image Pre-Processing

The images cannot be directly used in training because the size of each image was 480x640 pixels which is very large and hence it would have taken a lot more time to train. Hence, we first downsize the image to 96x128 pixels. After this, we have to remove redundant information from the image which in this case would be the background of the image. There is no perfect way to do this, there would always be some drawbacks and redundancies while removing the background. To do this, the image is converted to HSV and then by specifying the pixel value range of skin in HSV only the pixels with values in between this range are converted to white while all the remaining pixels are converted to black. This can be called as Skin-Masking. The white pixels are later replaced by their original pixel values. Since it is not necessary that only skin is in the specified HSV colour range hence sometimes redundancies remain. This image is then converted into a grayscale image.

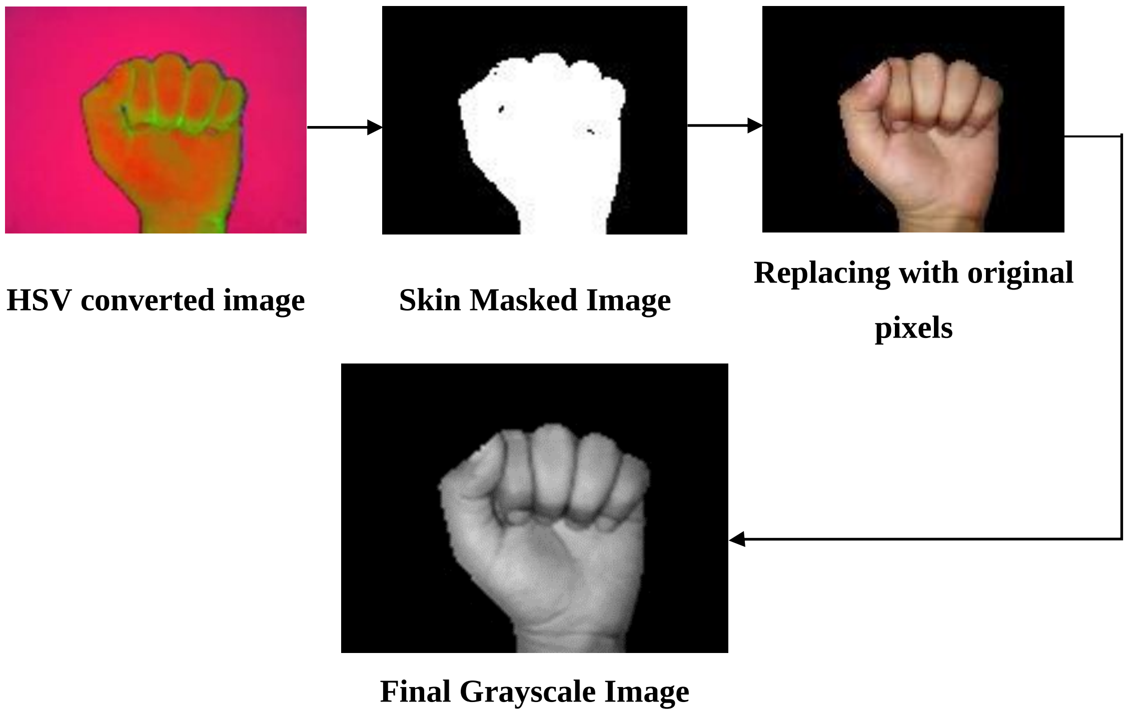
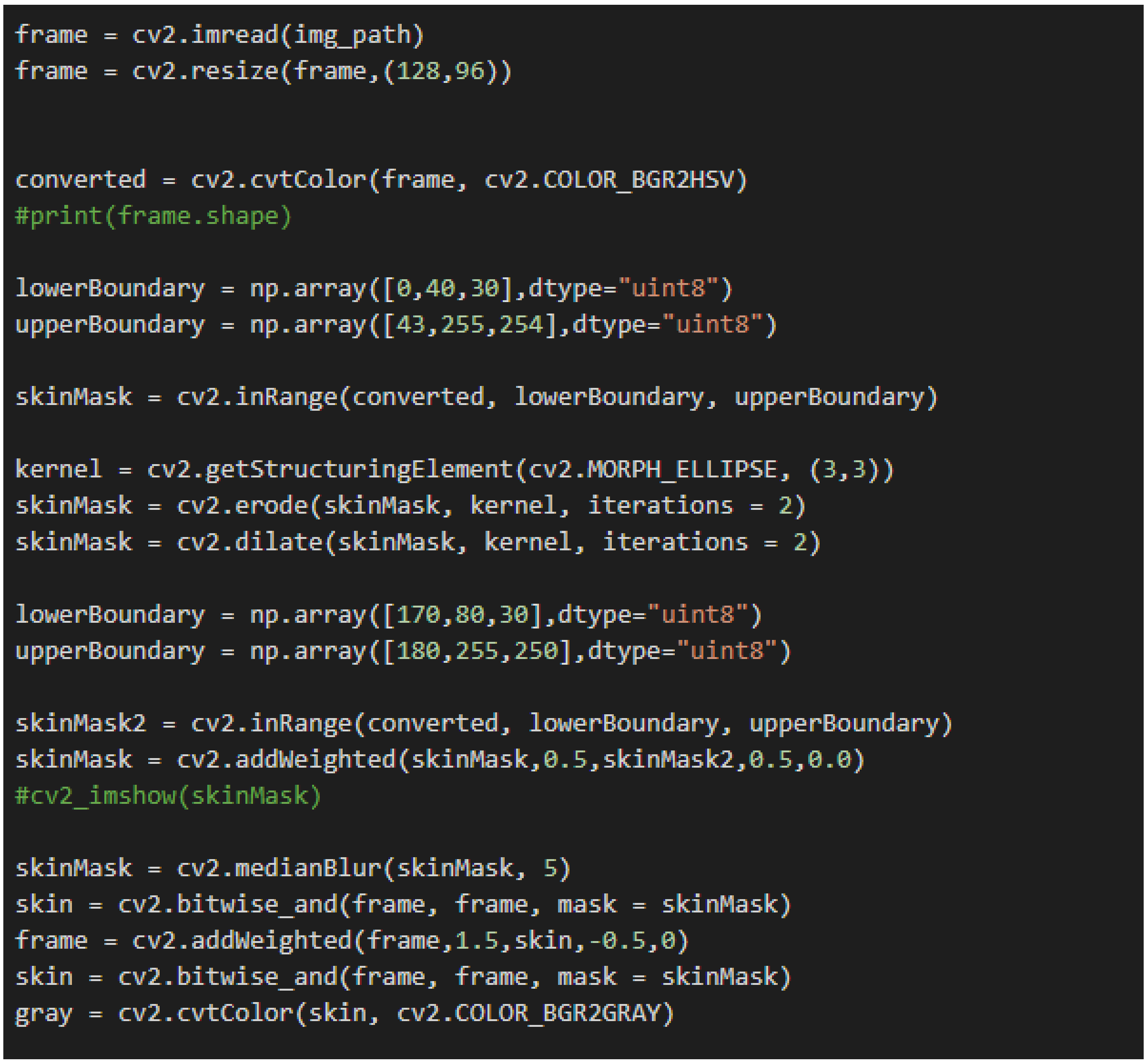


Fig 3.2.2:- Image Pre-Processing



```
frame = cv2.imread(img_path)
frame = cv2.resize(frame,(128,96))


converted = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
#print(frame.shape)

lowerBoundary = np.array([0,40,30],dtype="uint8")
upperBoundary = np.array([43,255,254],dtype="uint8")

skinMask = cv2.inRange(converted, lowerBoundary, upperBoundary)

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3,3))
skinMask = cv2.erode(skinMask, kernel, iterations = 2)
skinMask = cv2.dilate(skinMask, kernel, iterations = 2)

lowerBoundary = np.array([170,80,30],dtype="uint8")
upperBoundary = np.array([180,255,250],dtype="uint8")

skinMask2 = cv2.inRange(converted, lowerBoundary, upperBoundary)
skinMask = cv2.addWeighted(skinMask,0.5,skinMask2,0.5,0.0)
#cv2_imshow(skinMask)

skinMask = cv2.medianBlur(skinMask, 5)
skin = cv2.bitwise_and(frame, frame, mask = skinMask)
frame = cv2.addWeighted(frame,1.5,skin,-0.5,0)
skin = cv2.bitwise_and(frame, frame, mask = skinMask)
gray = cv2.cvtColor(skin, cv2.COLOR_BGR2GRAY)
```

Fig 3.2.3:- Image Pre-Processing Implementation

Model Building and Architecture

This model was trained on the dataset containing 3983 images belonging to 24 unique classes and 985 images were used to evaluate the model. The model was built using the TensorFlow Framework. The created CNN model (Convolutional Neural Network) is used for image classification.

The architecture of the model consists of two Convolutional 2D layers, two Max Pooling 2D layers, one Flatten layer and two Dense layers.

All the pre-processed images are sent to the input layer.

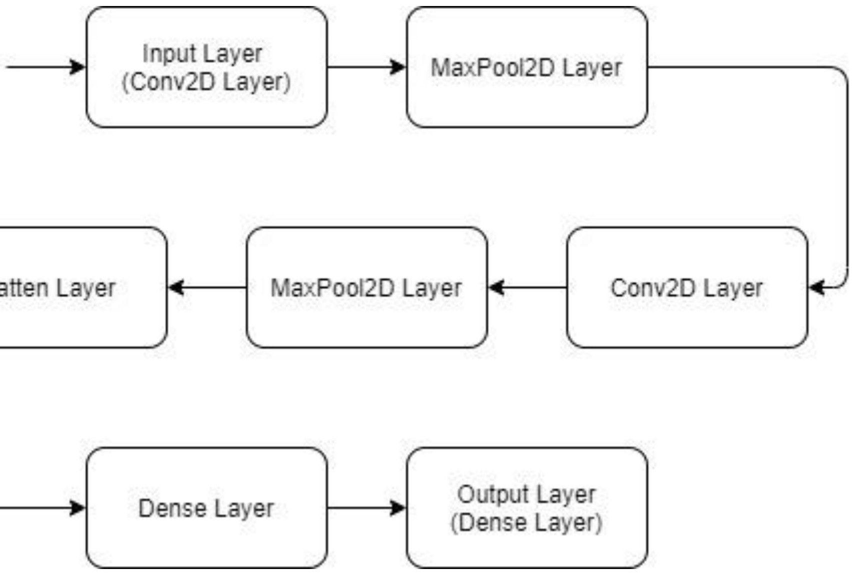
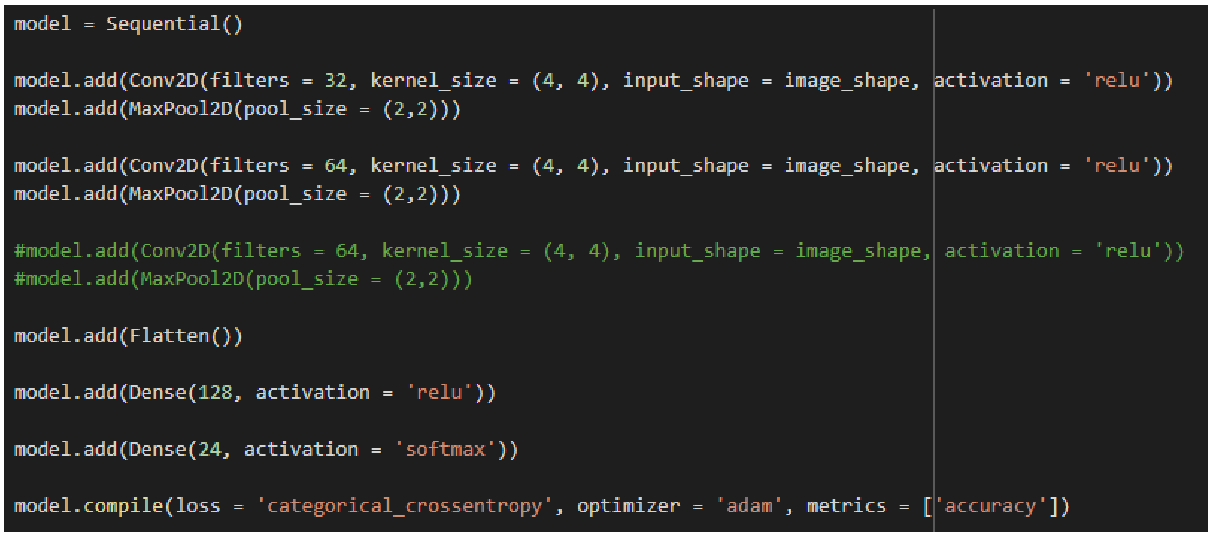


Fig 3.2.4 :- Model Architecture



```
model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (4, 4), input_shape = image_shape, activation = 'relu'))
model.add(MaxPool2D(pool_size = (2,2)))

model.add(Conv2D(filters = 64, kernel_size = (4, 4), input_shape = image_shape, activation = 'relu'))
model.add(MaxPool2D(pool_size = (2,2)))

#model.add(Conv2D(filters = 64, kernel_size = (4, 4), input_shape = image_shape, activation = 'relu'))
#model.add(MaxPool2D(pool_size = (2,2)))

model.add(Flatten())

model.add(Dense(128, activation = 'relu'))

model.add(Dense(24, activation = 'softmax'))

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

Fig 3.2.5 :- Model Architecture Implementation

Model Evaluation

The model was evaluated using the testing data consisting of 983 images. The model is giving an accuracy of around 79%. This accuracy is based on images similar to the images used for training.

Predicting for New Images

New images cannot be directly sent for prediction. They must first undergo the pre-processing step. Then the image can be sent to the model for prediction. The accuracy of the prediction for new images completely depends on how well the image gets pre-processed.

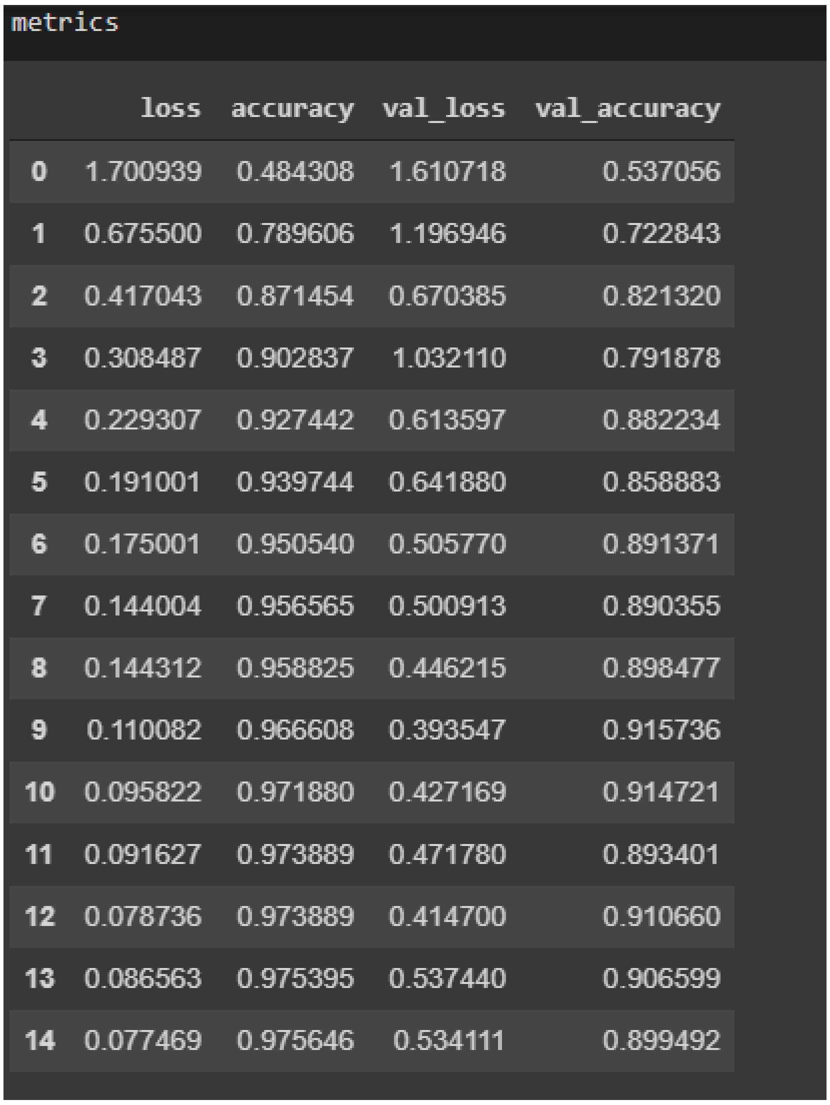
Chapter 4

Results and Discussion

4.1 Results:

The Recognition is good for well pre-processed image. But for images with large redundancies the prediction is completely different.

The results of the model evaluation are decent.



```
metrics
```

	loss	accuracy	val_loss	val_accuracy
0	1.700939	0.484308	1.610718	0.537056
1	0.675500	0.789606	1.196946	0.722843
2	0.417043	0.871454	0.670385	0.821320
3	0.308487	0.902837	1.032110	0.791878
4	0.229307	0.927442	0.613597	0.882234
5	0.191001	0.939744	0.641880	0.858883
6	0.175001	0.950540	0.505770	0.891371
7	0.144004	0.956565	0.500913	0.890355
8	0.144312	0.958825	0.446215	0.898477
9	0.110082	0.966608	0.393547	0.915736
10	0.095822	0.971880	0.427169	0.914721
11	0.091627	0.973889	0.471780	0.893401
12	0.078736	0.973889	0.414700	0.910660
13	0.086563	0.975395	0.537440	0.906599
14	0.077469	0.975646	0.534111	0.899492

Fig 4.1 :- Result/Accuracy of the model

As seen in the above image the validation accuracy of the model at the last epoch is 0.899492 or 89.9492 %.

4.2 Screenshots:

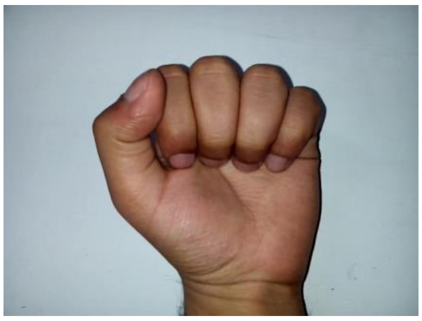


Fig 4.2.1(a):- Sign Language gesture of A

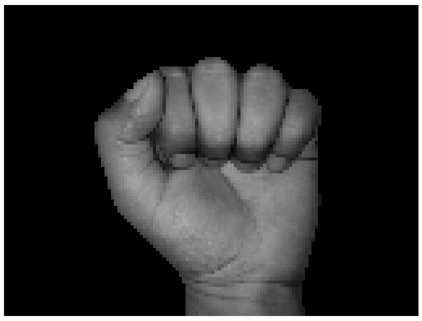
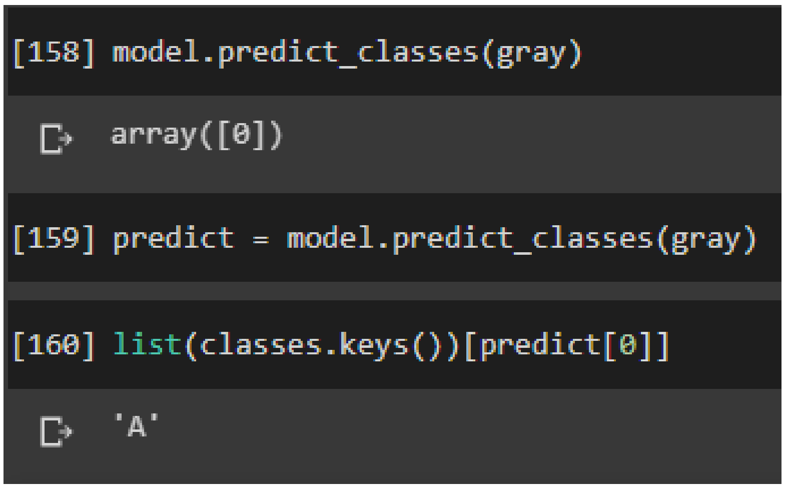


Fig4.2.1(b):-Image after Pre-processing of A



```
[158] model.predict_classes(gray)

      array([0])

[159] predict = model.predict_classes(gray)

[160] list(classes.keys())[predict[0]]

      'A'
```

Fig 4.2.1(c):- Prediction of the above image

The first image is a regular image taken on a camera without any kind of processing. It represents the character ‘A’. Then after pre-processing the image the background is mostly converted to black and then the result is predicted. In this case the model successfully predicted the gesture of ‘A’.

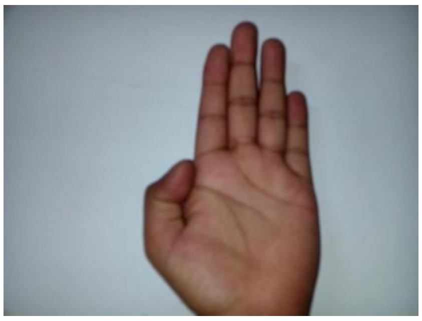
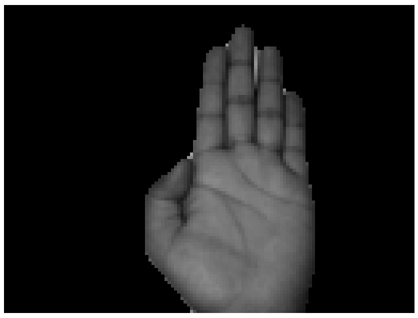
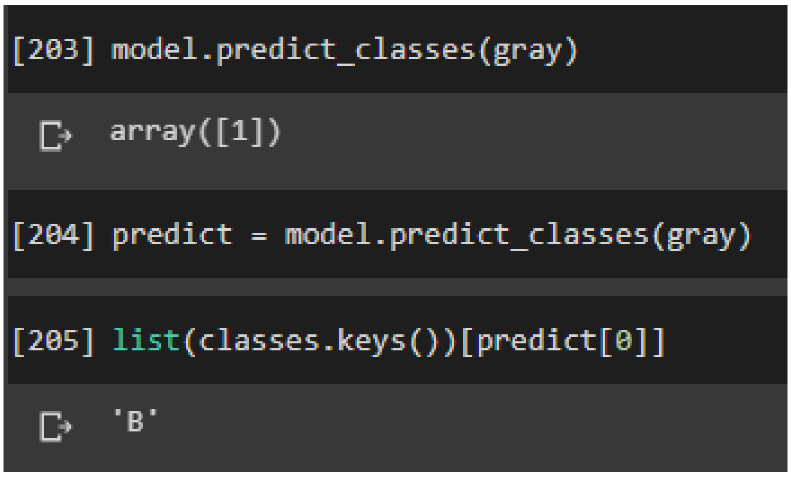



Fig 4.2.2(a):- Sign Language gesture of B **Fig4.2.2(b):-Image after Pre-processing of B**



```
[203] model.predict_classes(gray)

      array([1])

[204] predict = model.predict_classes(gray)

[205] list(classes.keys())[predict[0]]

      'B'
```

Fig 4.2.2(c):- Prediction of the above image

The first image represents the character ‘B’. Then after pre-processing the image the background is mostly converted to black and then the result is predicted. In this case the model successfully predicted the gesture of ‘B’.

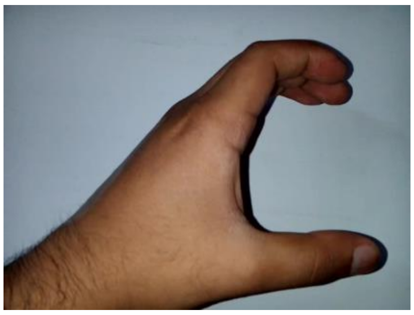


Fig 4.2.3(a):- Sign Language gesture of C

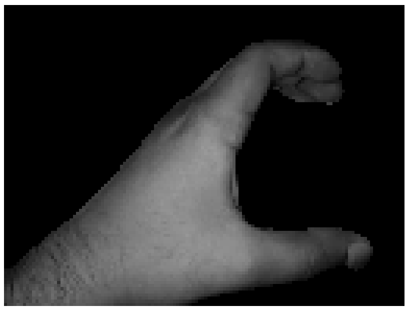
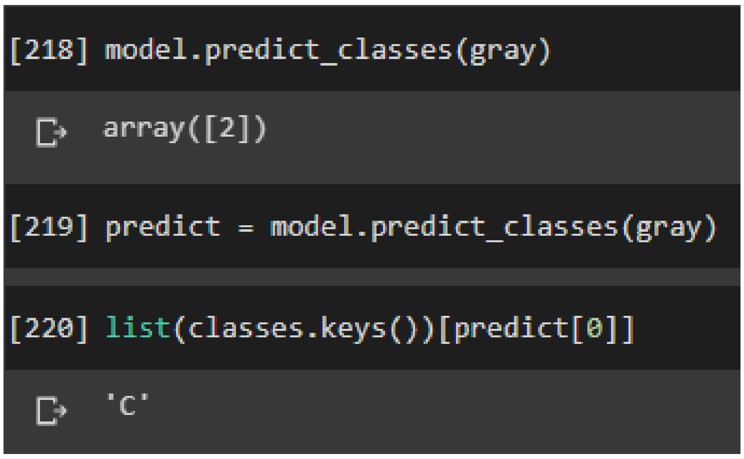


Fig4.2.3(b):-Image after Pre-processing of C



```
[218] model.predict_classes(gray)

      array([2])

[219] predict = model.predict_classes(gray)

[220] list(classes.keys())[predict[0]]

      'C'
```

Fig 4.2.3(c):- Prediction of the above image

The first image represents the character ‘C’. Then after pre-processing the image the background is mostly converted to black and then the result is predicted. In this case the model successfully predicted the gesture of ‘C’.

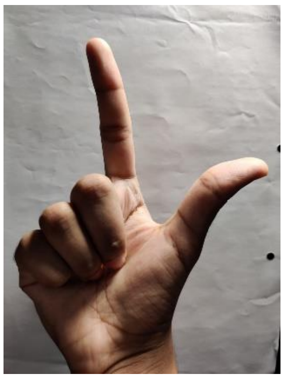


Fig 4.2.4(a) : Sign Language gesture of L

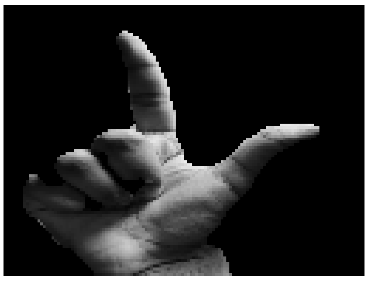
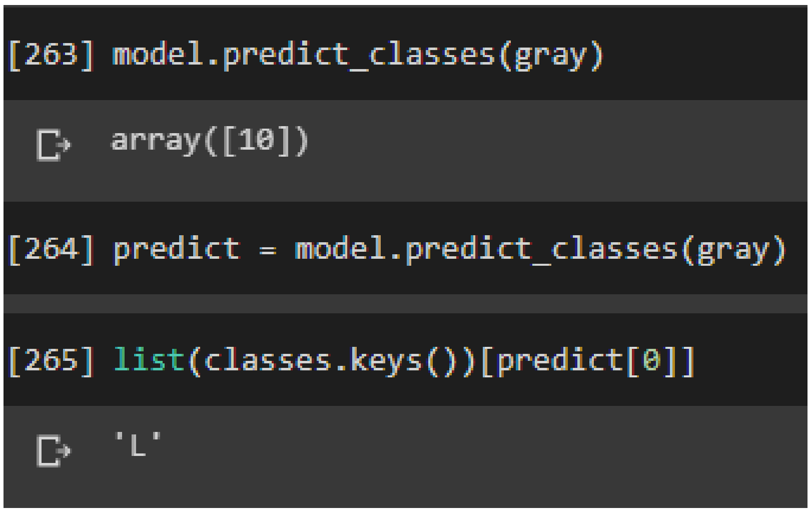


Fig4.2.4(b):-Image after Pre-processing of L



```
[263] model.predict_classes(gray)

      array([10])

[264] predict = model.predict_classes(gray)

[265] list(classes.keys())[predict[0]]

      'L'
```

Fig 4.2.4(c):- Prediction of the above image

The first image represents the character ‘L’. Then after pre-processing the image the background is mostly converted to black and then the result is predicted. In this case the model successfully predicted the gesture of ‘L’.

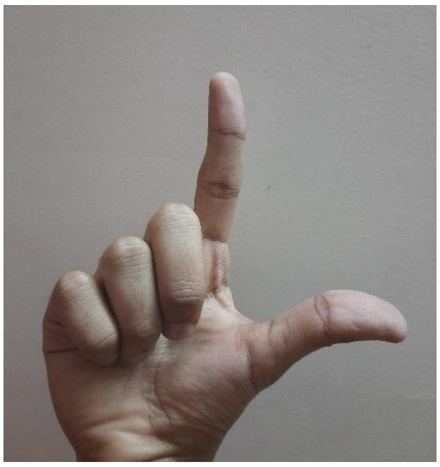


Fig 4.2.5(a) : Sign Language gesture of L

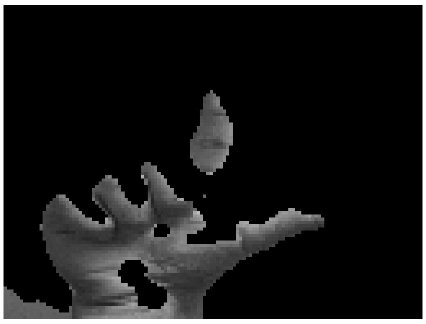
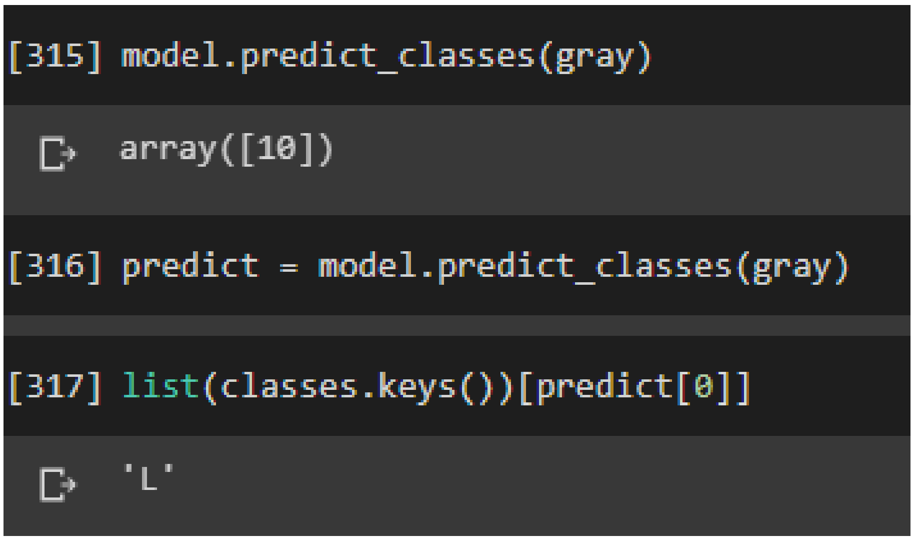


Fig4.2.5(b):-Image after Pre-processing of L



```
[315] model.predict_classes(gray)

      array([10])

[316] predict = model.predict_classes(gray)

[317] list(classes.keys())[predict[0]]

      'L'
```

Fig 4.2.5(c):- Prediction of the above image

The first image represents the character ‘L’. Then after pre-processing the image the background is mostly converted to black and then the result is predicted. As it can be seen that the pre-processing of the image is not proper as the background is quite similar to the skin colour and hence when converted to HSV, it falls in the HSV range specified. But still, in this case the model successfully predicted the gesture of ‘L’.

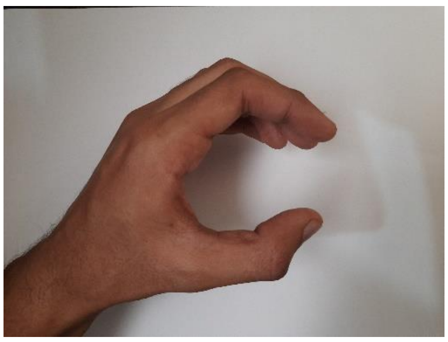


Fig 4.2.6(a) : Sign Language gesture of C

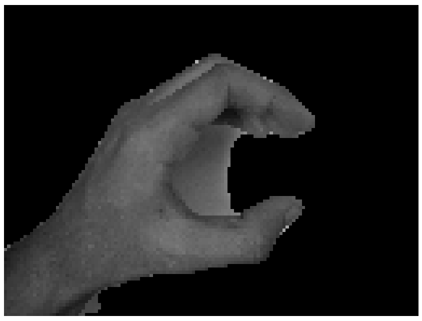
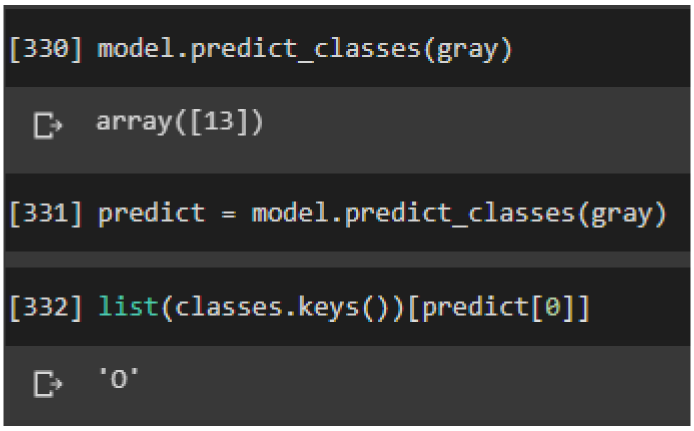


Fig4.2.6(b):-Image after Pre-processing of C


```
[330] model.predict_classes(gray)

      array([13])

[331] predict = model.predict_classes(gray)

[332] list(classes.keys())[predict[0]]

      'O'
```

Fig 4.2.6(c):- Prediction of the above image

The first image represents the character ‘C’. Then after pre-processing the image the background is mostly converted to black and then the result is predicted. As it can be seen that the pre-processing of the image is not proper as the shadow of the image is falling in the HSV range specified. Hence it doesn’t get converted to black. In this case the model incorrectly predicted the gesture of ‘C’ as character ‘O’.

Chapter 5

Conclusion

5.1 Conclusion:

As seen from the above screenshots/examples if the pre-processing of the image has some major redundancies then the answer may or may not vary from the desired result.

Hence we can conclude that to obtain good results we must use an image with a plain background preferably white background.

5.2 Future Scope:

The Sign Language Recognition System has a really broad scope which can be widely expanded to real time detection and be used as a direct translator.

It can be used for various other gesture detections as well.

Hence the Sign Language Recognition System can have a big scope in the future.

Chapter 6

References

[1] Sunmok Kim, YanghoJi, and Ki-Baek Lee. “An effective sign language learning with object detection based ROI segmentation.” (2018).

[2] Ashish S. Nikam and Aarti G. Ambedkar. “Sign Language Recognition Using Image Based Hand Gesture Recognition Techniques.” (2016).

[3] G.Anantha Rao, K.Syamala, P.V.V.Kishore, A.S.C.S.Sastry. “Deep Convolutional Neural Networks for Sign Language Recognition.” (2018).