HAND GESTURE RECOGNITION USING DEEP LEARING

A CAPSTONE PROJECT REPORT

Submitted in partial fulfillment of the requirement for the award of the Degree of

**BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE ENGINEERING
WITH SPECIALIZATION IN DATA ANALYTICS**

by

TEJO VINAY POTTI (17BCD7014)

Under the Guidance of

DR. USHA RANI GOGOI

SCHOOL OF COMPUTER SCIENCE ENGINEERING
VIT-AP UNIVERSITY
AMARAVATI- 522237

DECEMBER 2020

CERTIFICATE

This is to certify that the Capstone Project work titled "**HAND GESTURE RECOGNITION USING DEEP LEARNING**" that is being submitted by **TEJO VINAY POTTI (17BCD7014)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

Usha Rani Gogoi 30/3/21

Dr. Usha Rani Gogoi
Guide

The thesis is [✓ satisfactory] / unsatisfactory

Approved by

[illegible] [illegible]

PROGRAM CHAIR **DEAN**

B. Tech. CSE-DA School of Computer Science Engineering

ACKNOWLEDGEMENTS

I acknowledge with thanks the kind of patronage, guidance and inspiration I have received from my project guide, Dr. Usha Rani Gogoi and Dean CSE, Dr. Hari Seetha. I am grateful for the interest, guidance and encouragement they gave me at different stages of the capstone project. The suggestions they gave and their constructive criticism have contributed a lot to the evolution of my project. This project would not have been completed without their enormous help and worthily experience.

I am very thankful to the university for giving me this opportunity to put my theory into practice.

I also thank my parents for their encouragement which helped me in completing this project.

ABSTRACT

In this project an efficient system is presented which can automatically recognize hand gestures. The project introduces an application using computer vision, image processing and deep learning for recognition of hand gestures. An interface is created such that the computer's camera records a live video stream, takes snapshots and stores them to the system. The system is then trained on different classes of gestures at least once. After that, the test gestures are given to the model through the live stream video capture and the system tries to recognize it.

Literature survey reveals that the state-of the art systems used data gloves or markers while giving input to the system. This system has no such constraint. The user can give his hand gestures in front of the camera in its view naturally and system detects and recognizes the gesture. The implemented system can serve as an extendible foundation for future work in its applications. One of the most significant applications can be converting signs to speech. This is really important for communication with or between the people who are deaf and dumb. This system can also be applied to improve gaming experiences like for playing virtual darts.

The existing systems so far have used several known CNN architectures such as AlexNet and ResNet to get the most accurate results. Unlike the existing system, this project uses novel methods involving Augmentation and a self-developed CNN architecture that performs best specifically for hand gestures. This approach has resulted into better accuracies.

TABLE OF CONTENTS

Tables List

List of Figures

CHAPTER 1
INTRODUCTION

With constant development in technology, people are nowadays trying more to make connection with computers in many ways than before. The normal devices that are used for interaction between humans and computers like remotes, keyboards and mice are no longer the latest means for the purpose of interaction. It is because they are not flexible enough. The most flexible, updated and latest means of interaction is body language and voice commands. These are now available in many latest electronic devices.

Even though speech recognition has become popular for human computer interaction, it still has its own limitations. It cannot work well in noisy environments. Also, it is hard to tune it to the system as the same words are pronounced differently by different people. Alternate method involves interaction based on body language which is more reliable and consistent than a lot of previously used methods. It can involve various types of expressions such as emotions captured by face, body posture and gestures shown by hand. Out of all these, gestures shown by hand is the most efficient as it is a language that is common for everyone and can avoid all the limitations discussed above.

Humans use our gestures every day while interacting with different people in different scenarios. Hand gestures are used mainly in situations where ideas are being presented and to point at something. This kind of interaction is quite familiar for everyone. People also believe that the touch screen technology in different devices might be replaced by hand gestures soon [1].

Hand gestures can be applied across various applications. Some of them are sign language detection, interaction between humans and vehicles, controlling a robot, virtual mouse control and in gaming technologies for a more realistic experience. Also, any system requires physical contact to interact with them. Gestures can interpret the same functionality without physically interacting with the devices. Especially during current covid scenario, it is very important and necessary to build contactless interactions.

This project is mainly based on the concept image processing and deep learning using convolutional neural networks. In recent years, CNNs has been showing outstanding performance with image data. Because of the feature of being able to use multiple layers in the architecture, automated feature learning has become easy using CNNs. So, by using a deep learning framework, data can be hierarchically arranged from low-level to high-level in the form of features. Presently, there has been a lot of research in gesture recognition using kinect sensors[2] or using HD cameras[4] which are expensive. This project focuses on the reduced cost and improved robustness using a simple camera.

1.1 Objectives

The project's main objective is to develop a static Hand Gesture Recognition system using Deep Learning algorithms. For this the specific deliverables will include

- Development of a challenging dataset comprising of different hand gestures.
- A strong model with the right architecture optimized well for the best accuracy.
- An automatic system for Hand Gesture Recognition.

1.2 Background and Literature Survey

There has been a lot of research work in the recent years on Gesture Recognition. The objective of the work in [6] is similar to the objective of this project. They used the Americal Sign Language dataset which has about 11000 samples. They evaluated the performance of various CNN architectures to analyse and find the best architecture. However, all the architectures were existing and well known architectures such as LeNet, AlexNet, GoogleNet and ResNet. They reported the highest accuracy of 96%.

Another work proposed by 'Chonnam National University'[4] was referred here. Their model was trained on a private dataset with about 5250 sequences. Their proposed system achieved an accuracy of 92%. The main drawback of their research work was the images and sequences were captured using RGBD (Depth) cameras and Kinect sensors which are expensive.

We also refer the work done by 'Chinese Academy of Sciences'[5]. They used the Ninapro DB5 dataset which contains about 60000 samples approximately. It attained an accuracy of 98% however the method used was compact CNNs and Surface Electromyography which requires putting on armbands for collecting signal data. This is not practically feasible.

1.3 Organization of the Report

The organization of the report is as the following chapters:

- Chapter 2 contains the procedure for development of the experimental dataset.
- Chapter 3 contains the proposed system, methodology, implementation, results.
- Chapter 4 concludes the report.
- Chapter 5 consists of codes.
- Chapter 6 gives references.

CHAPTER 2

Development of Experimental Dataset

The system built is just as worth as the dataset trained on. The system has to work for anyone in general so properties like size, orientation and colour of hand differ from person to person when they are showing gestures. For this reason, it is necessary to build a challenging dataset for the model to be trained on. But, it is impossible to collect data from a huge population especially during the current lockdown. The data was initially constructed with different gestures by a single person's hand and then augmented to expand it and bring in variations to each image present, thereby making the images more complex to train a model on.

The dataset for this project has been developed on 11 different gestures with 1200 samples for each gesture resulting in a total of 13200 images for the model to work on. Figure 1 shows the 11 gestures the project focuses on.

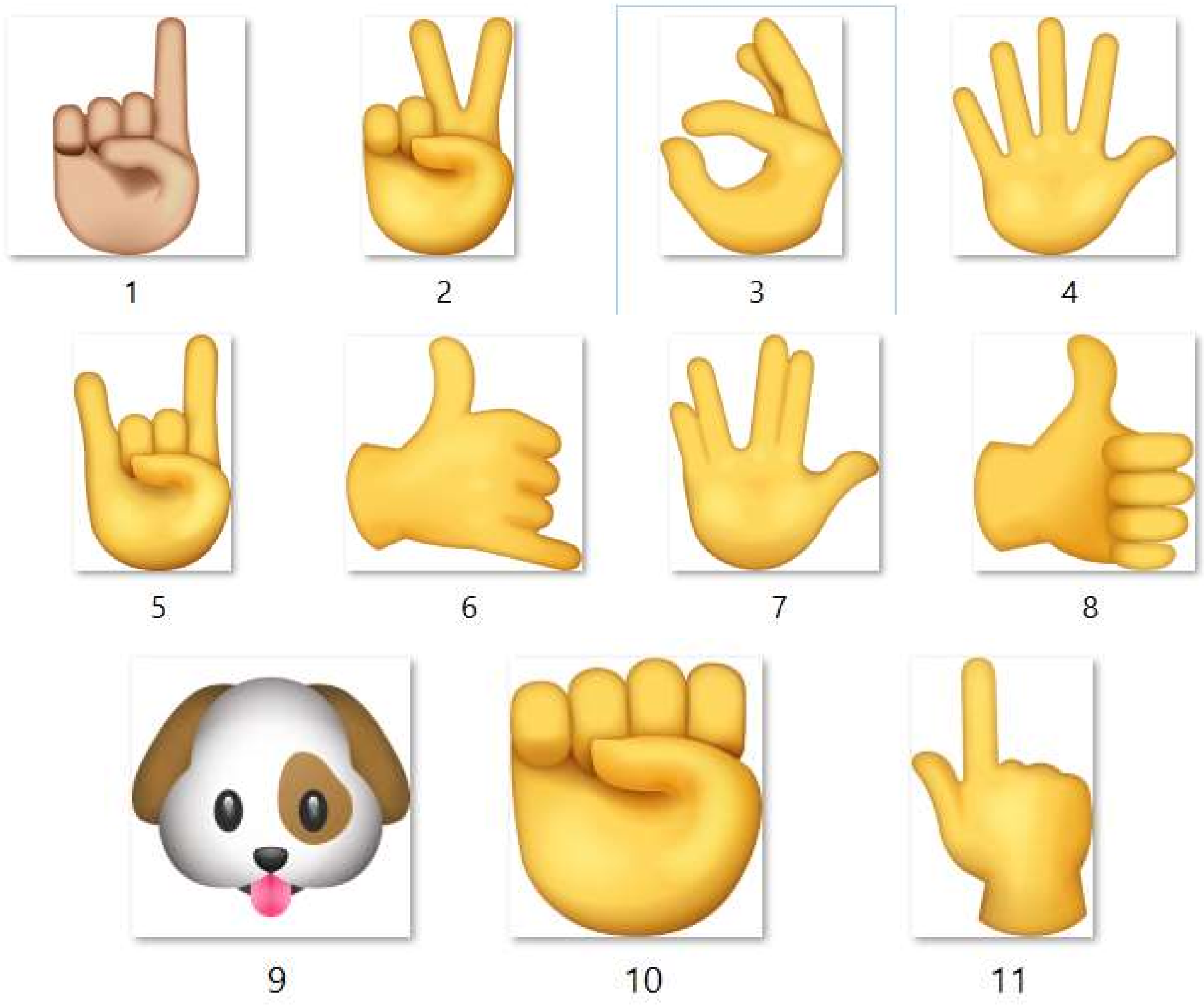


Figure 1 Target Gestures

2.1 Pre-processing

2.1.1 Segmentation of Region of Interest: Hand Region

Data pre-processing is the important step before building the model. For this project, pre-processing has been done at two stages. The pre-processing was done while capturing images for the dataset and also for increasing the dataset size and making it more challenging.

While capturing the images, it is important to transform the raw data into useful and efficient data before saving the images for the dataset. For this, each frame getting captured was converted from BGR to HSV colour format[7], then extracted only the hand part by giving the pixel range

Hue calculation:

$$H = \begin{cases} 0^\circ & \Delta = 0 \\ 60^\circ \times \left(\frac{G'-B'}{\Delta} mod6\right) & , C_{max} = R' \\ 60^\circ \times \left(\frac{B'-R'}{\Delta} + 2\right) & , C_{max} = G' \\ 60^\circ \times \left(\frac{R'-G'}{\Delta} + 4\right) & , C_{max} = B' \end{cases}$$

Saturation calculation:

$$S = \begin{cases} 0 & , C_{max} = 0 \\ \frac{\Delta}{C_{max}} & , C_{max} \neq 0 \end{cases}$$

Value calculation:

$$V = Cmax$$

for skin colour ([2, 50, 60], [25, 150, 255])[8] and then applied this mask using a bitwise_and operator which gives the resultant image as a black and white image. Later, the colour format is changed to grayscale and applied Gaussian blur to smoothen the image. In the Gaussian Blur operation[9], convolution takes place on the image with a filter called the Gaussian filter and not the box filter. It is a low-pass filter that reduces or removes the components having high frequency.

The Gaussian filter is linear, so is used to reduce noise and blur images. Here, the Gaussian filter is used for the same purpose. It will reduce contrast and blur the edges. Then the image is dilated to improve the foreground i.e., the hand region of the image and is gone through a morphological transformation to remove white noise or outliers in the image. Then a threshold is set and contours are extracted and saved as an image.

Figure 2 shows the sample dataset image of the corresponding gesture shown in Figure 1. These images were captured from the camera using computer vision and then pre-processed before saving the image. The pre-processing was done such that it provides good results when applied with a CNN architecture.

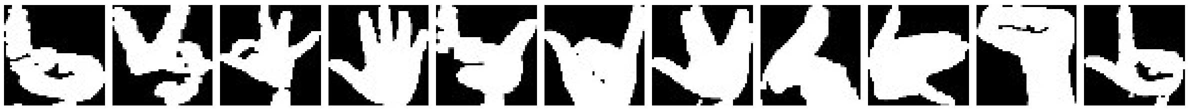


Figure 2 Captured Images for Dataset

2.1.2 Data Augmentation

The next step involves expansion of the created image dataset. The created image dataset had images of complete similarity for each gesture class. This similarity will not allow us to build a generalized model and will go linear with any simple CNN model to give 100% accuracy. So, data augmentation was done to make the image dataset more challenging and sophisticated. For this, a data generator class was created to rotate, zoom, shift width, shift height and perform horizontal flips randomly on different images in a varied range. Data augmentation was also used to increase the dataset size by almost 3 times to make the data more complex and increase generalizability. Now, this image data requires a complex CNN architecture to give good accuracy.

Then, this expanded gesture dataset was converted into a CSV file holding the pixel values of the images with target variable as the gesture class it belongs to.

Figure 3 shows the sample dataset image of the corresponding gestures after performing data augmentation on the images in Figure 2.

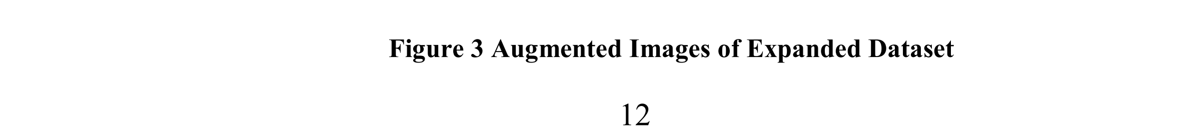


Figure 3 Augmented Images of Expanded Dataset

CHAPTER 3
HAND GESTURE RECOGNITION

3.1 Proposed System

The main objective of this project is to identify hand and recognize which hand emoji or hand sign is trying to be shown. The project was built using Computer Vision, Image Processing and Deep Neural Networks. This deep learning model has been built on Jupyter Notebooks using Python.

The project was built in 3 phases. First being preparing the hand gesture data of developed database. Second being creating the model and training the model. Third being the application layer of the model to obtain the desired results.

Phase 1: The first phase involves the processing of collected hand gesture data to cancel background noise and pickup only the gesture area and store these frames as image files in jpeg format. Then, convert this image data into a CSV pixel format dataset, to pass different filters through certain functions.

Phase 2: This phase involves dividing the processed dataset into training data and testing data, then building a sequential model using Convolutional Neural Networks from the Keras neural network library which runs on top of TensorFlow. Convolution Neural Networks will be used to classify different hand gestures and segregate them into different emoji classes. Then, will feed all this to a model class, identify the best optimizer, run it through multiple epochs to reach the best accuracy possible and save the model. Also, run this model on a benchmark dataset to check if the model is working well.

Phase 3: This is the application phase where the input is taken at runtime (livestream) and the captured hand gesture is passed to the model to get it classified and mapped to the hand gesture of its class as output. Then, the output gesture is overlayed to place it on top of the frame where hand is being recognized.

3.2 Workflow of the Project

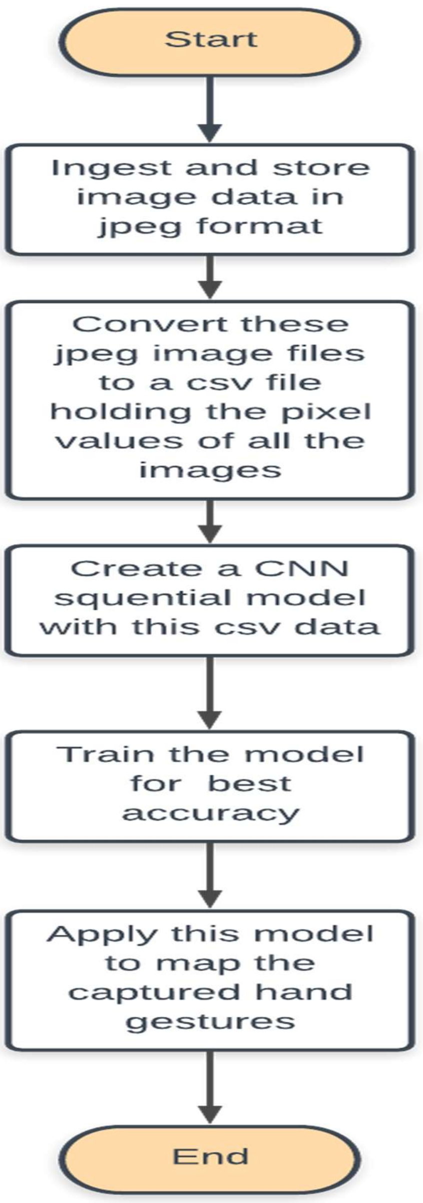

Start
Ingest and store image data in jpeg format
Convert these jpeg image files to a csv file holding the pixel values of all the images
Create a CNN squential model with this csv data
Train the model for best accuracy
Apply this model to map the captured hand gestures
End

Figure 4 Project Workflow

3.3 Engineering Standards

The main engineering standards that will be used to build this project are:

- **Convolutional Neural Networks**: It is a type of deep neural networks mainly applied on visual data types like images and videos to analyse them. It is a neural network used with the concept of layers called convolutional layers. CNNs are used mainly for classification and segmentation. It is also in image processing and for other correlated data. For this project, CNN will be used to classify the images into different classes representing the gesture.

- **Activation Functions**: It is the function used to learn complex patterns in the data in the network by adding it to the neural network. The activation function takes in the aggregated values at each layer, applies its specific function and gives the output which acts as either the final output or input for next layer in the network. There are different activation functions such as Sigmoid, Tanh, Relu, Leaky Relu and Softmax. This project will be using the softmax activation function for the output layer is classified into multiple classes. For the remaining layers, a single activation is considered which will be the activation function which provides the best accuracy for the model.
- **Optimizer**: Optimizer also called the learning algorithm is used to improve loss after each iteration up the neural network. It will change the weights and learning rate in order to reduce the loss and provide the most accurate results possible.
- **Computer Vision**: Computer vision helps by training computers in interpreting and understanding the visual world. It enables computers to understand the content of images and videos. In this project, Computer Vision is used to capture live gestures being shown and also for creating the dataset by saving frames as images while capturing the gestures.

3.4 Developing the CNN Model

Building a complex dataset was done and the next phase of the project was to build a model with promising accuracy. Figure 4 shows the base architecture followed for this project. The flow or the sequence of the layers would be the same but the number of layers of batch normalization, dropout layers and kernel sizes, number of filters were twitched and tested for optimization.

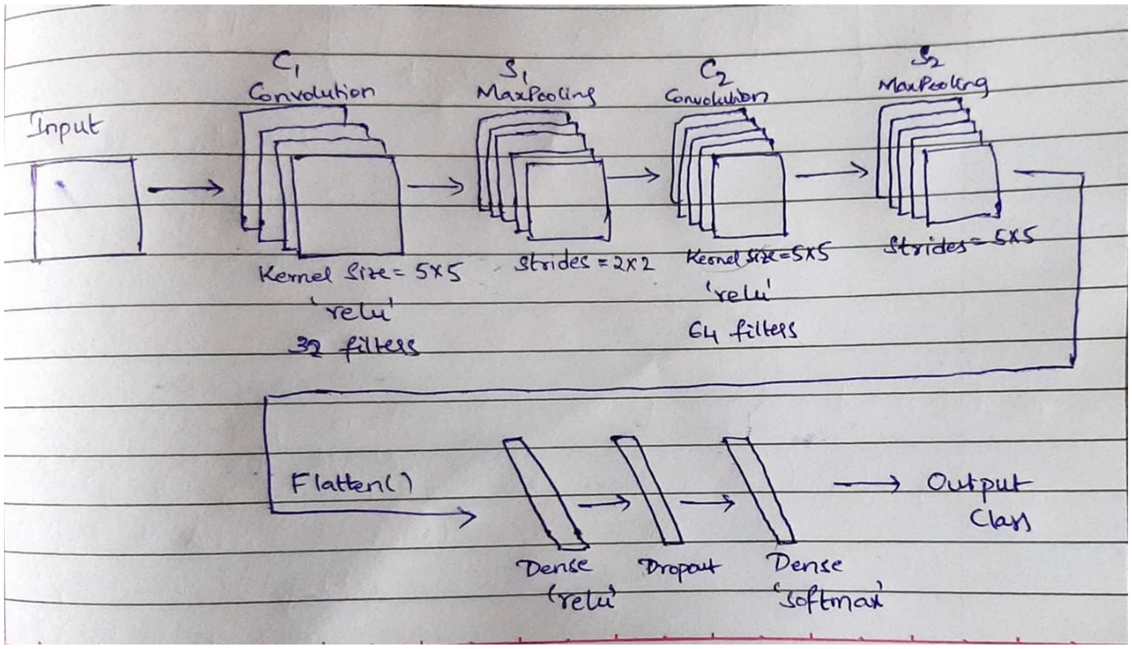
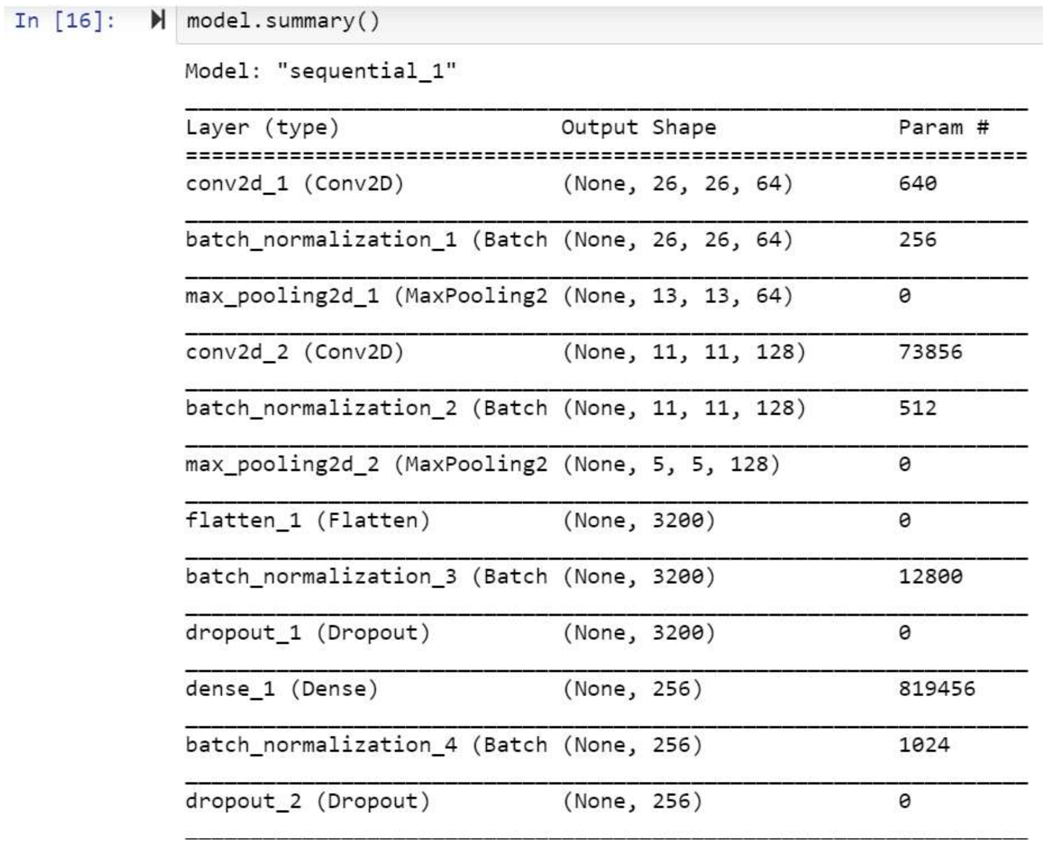


Figure 5 CNN Model Base Architecture



```
In [16]:  model.summary()

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 64)        640
_________________________________________________________________
batch_normalization_1 (Batch (None, 26, 26, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 128)       73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 11, 11, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3200)              0
_________________________________________________________________
batch_normalization_3 (Batch (None, 3200)              12800
_________________________________________________________________
dropout_1 (Dropout)          (None, 3200)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               819456
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)               1024
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
```

Figure 6 CNN Model Initial Architecture

The problem with the base architecture was, it did not have enough and appropriate bath normalization, dropout layers which either lead to overfitting or very poor performance of the model. So, on this architecture, more layers were built according to the data and the performance got better. After lots of changes to the base architecture, the architecture in Figure 5 was the first one to cross 90% accuracy. It consists of 2 layers of convolution and max pooling combination with batch normalization layers after every complex operation layer. Dropout layers were also added after flatten operation. Batch normalization helps in increasing the stability of the neural network. It takes in the output of the previous activation layer and normalizes it by subtracting the mean of batch and dividing by the standard deviation of batch. Dropout helps in preventing the model from overfitting. It works by taking random neurons and killing it that is by setting the output of neurons to 0 at every update that happens during training model.

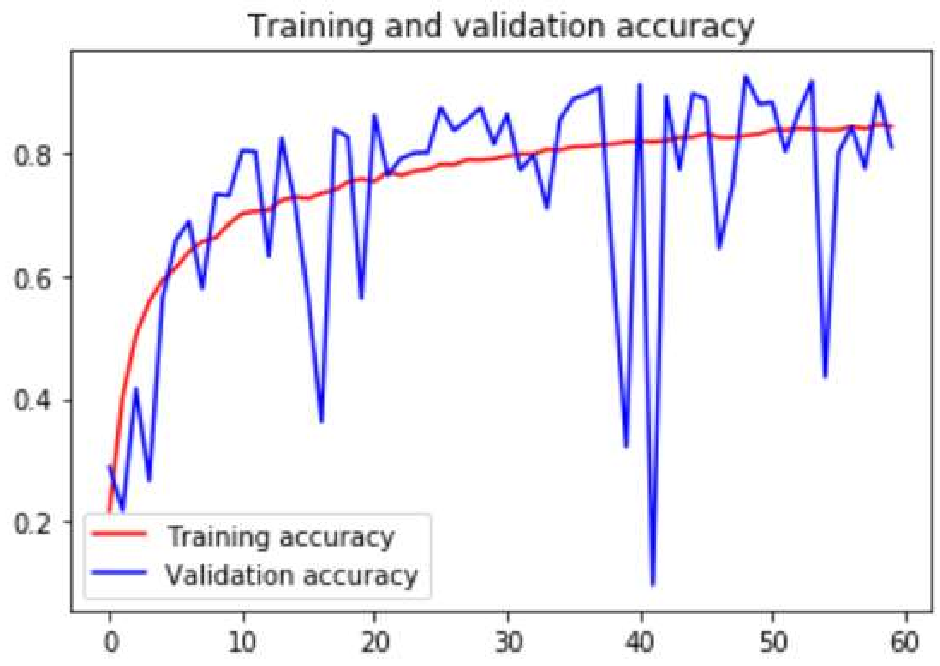
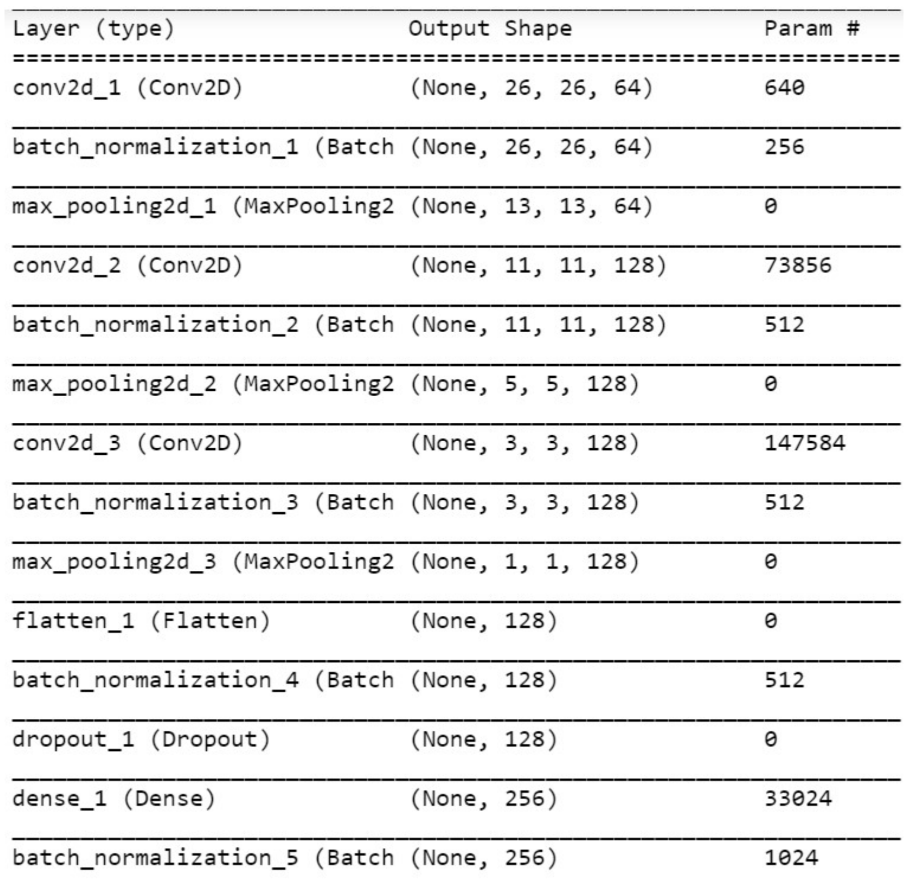


Figure 7 Initial Architecture Accuracy Plot

An accuracy of approx. 92% was obtained from the above model (Table 1). This accuracy is just not enough in practical prospect and should at least be 95%. Also, in the train test accuracy plot in Figure 6, we can see that there are a lot of downward spikes in the validation accuracy even after 50 epochs and the graph is not quite stabilizing by the end.

In order to optimize the model for a more stable curve and better accuracy, the above model had to be improved more. Batch normalization decreases overfitting because it has little effects of regularization. Just like dropout, each hidden layer's activations are added with some noise when batch normalization is used. So, less dropout should be used when batch normalization is used so that we do not lose a lot of information. So, for the updated model, an extra layer of convolution and max pooling was added, a dropout layer was removed and an extra batch normalization was added leaving only one dropout layer in the model to make sure that important information is not lost. Figure 7 shows the updated final model which has shown the best accuracy.

The final model obtained an accuracy of 97.28% (Table 1). Also, from the accuracy and loss plots in Figure 8 and Figure 9, it is quite evident that the curves stabilize well and there are no high downward spikes. The validation curve smoothens by the end of 60 epochs. Those little variations in the validation accuracy/loss is because of the generalizability obtained from the data augmentation performed on the image dataset.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 64)        640
_________________________________________________________________
batch_normalization_1 (Batch (None, 26, 26, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 128)       73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 11, 11, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 128)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 128)         147584
_________________________________________________________________
batch_normalization_3 (Batch (None, 3, 3, 128)         512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 1, 1, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               33024
_________________________________________________________________
batch_normalization_5 (Batch (None, 256)               1024
```

Figure 8 Final CNN Model Architecture

MODEL	ACCURACY
Base Architecture	68%
Initial Model	91%
Final Model	97%

Table 1 Model Accuracies

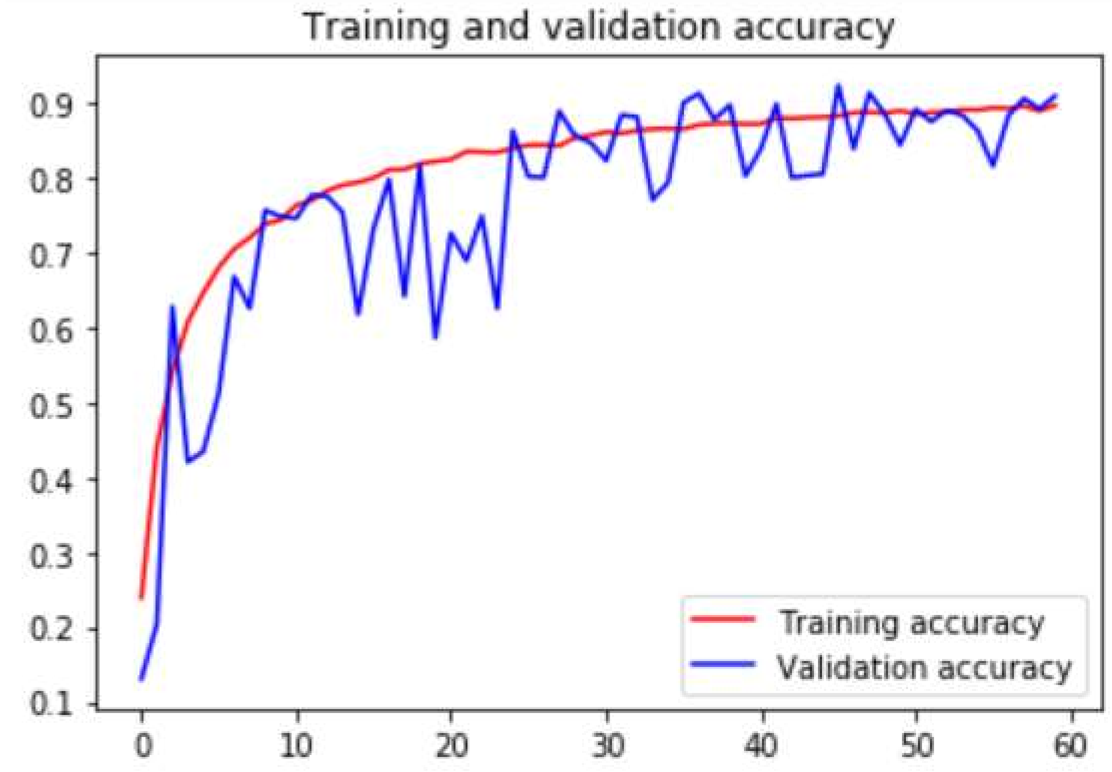


Figure 9 Final Model Accuracy Plot

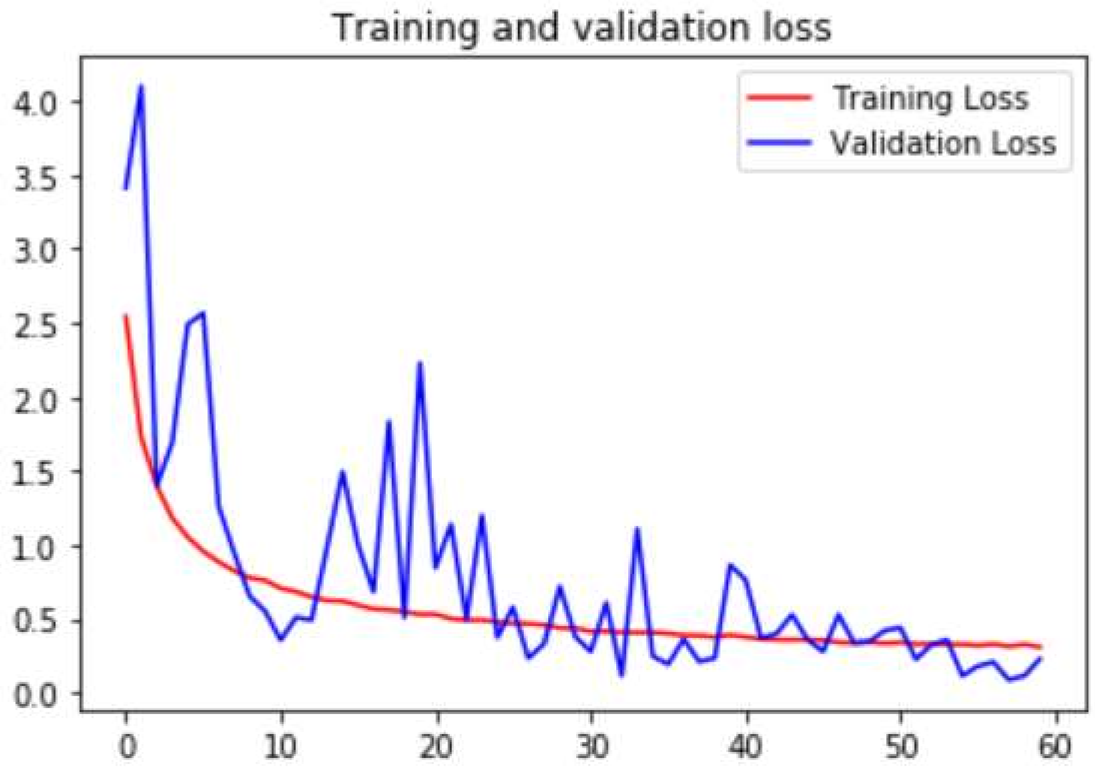


Figure 10 Final Model Loss Plot

3.5 Evaluating the Model

The final model performed very well on our developed dataset. Now to evaluate the robustness of the proposed model in hand gesture recognition, it is necessary to evaluate its efficiency on a benchmark dataset. For this, the American Sign Language (ASL) dataset was considered. It is called Sign Language MNIST on Kaggle[10]. The ASL database has 24 letter classes (except the letters 'J' and 'Z' as these classes are not static). Each case of train and test have a label from 0 to 25 which are mapped to the corresponding alphabet that is from A to Z (except for 9 which is J and 25 which is Z as they aren't static gestures). The number of train and test cases are 27,455 and 7172 respectively. Figure 10 shows a sample of the gestures used in the dataset.

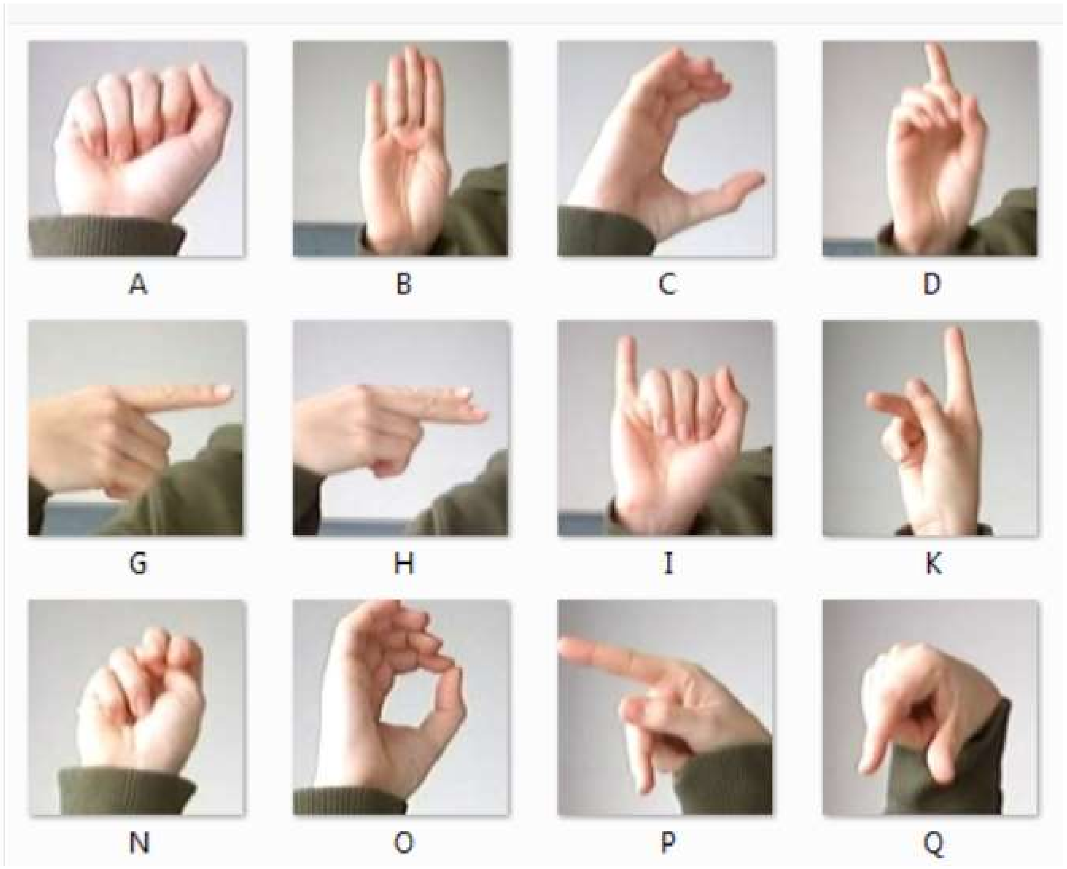
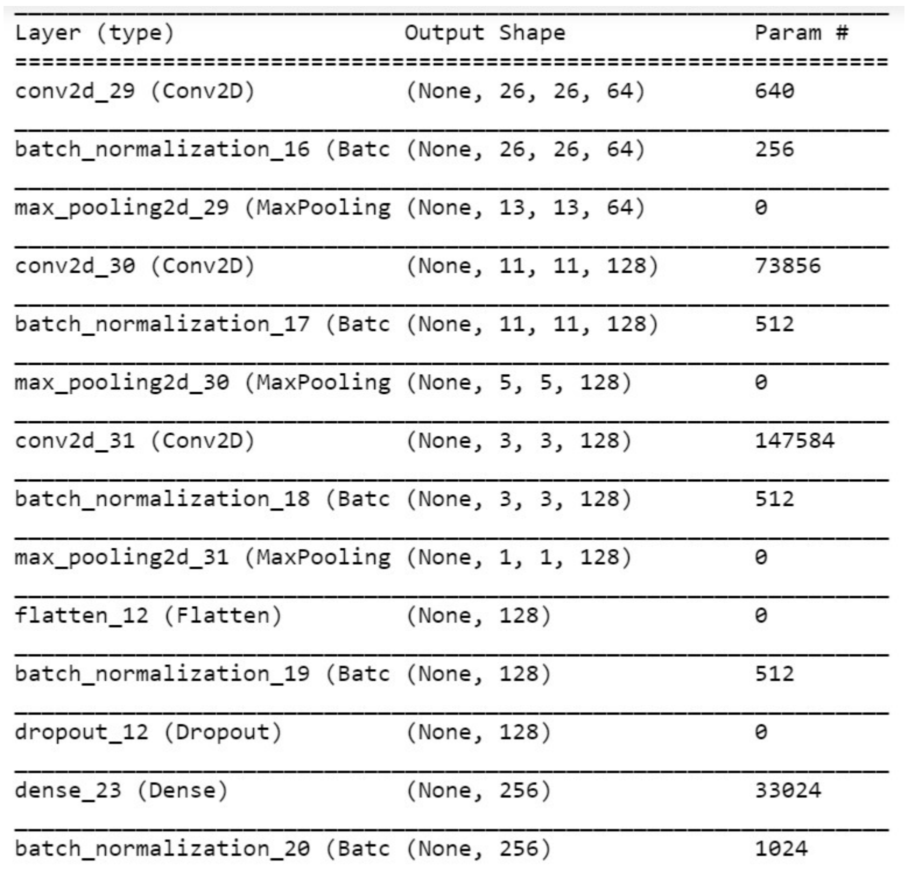
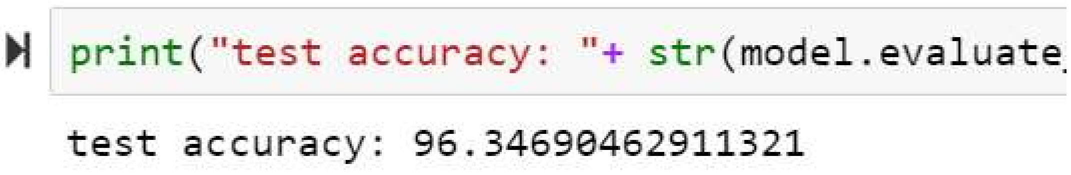


Figure 11 ASL Gestures



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_29 (Conv2D)           (None, 26, 26, 64)        640       
_________________________________________________________________
batch_normalization_16 (Batc (None, 26, 26, 64)        256       
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 13, 13, 64)        0         
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 11, 11, 128)       73856     
_________________________________________________________________
batch_normalization_17 (Batc (None, 11, 11, 128)       512       
_________________________________________________________________
max_pooling2d_30 (MaxPooling (None, 5, 5, 128)         0         
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 3, 3, 128)         147584    
_________________________________________________________________
batch_normalization_18 (Batc (None, 3, 3, 128)         512       
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 1, 1, 128)         0         
_________________________________________________________________
flatten_12 (Flatten)         (None, 128)               0         
_________________________________________________________________
batch_normalization_19 (Batc (None, 128)               512       
_________________________________________________________________
dropout_12 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_23 (Dense)             (None, 256)               33024     
_________________________________________________________________
batch_normalization_20 (Batc (None, 256)               1024      
```

Figure 12 CNN Model on ASL Dataset



```
print("test accuracy: "+ str(model.evaluate

test accuracy: 96.34690462911321
```

Figure 13 CNN Model Accuracy on ASL Dataset

On applying the same architecture (Figure 11) on the ASL dataset, an accuracy of 96.34% (Figure 12) was achieved which is as good as the accuracy achieved in one of recent papers on the same dataset. So, this shows that the final model is a good model and generalizes well with other data as well. Also, the accuracy and loss plots on train and validation shows no sudden extreme spikes and is quite smooth throughout (as shown in Figure 13 and Figure 14).

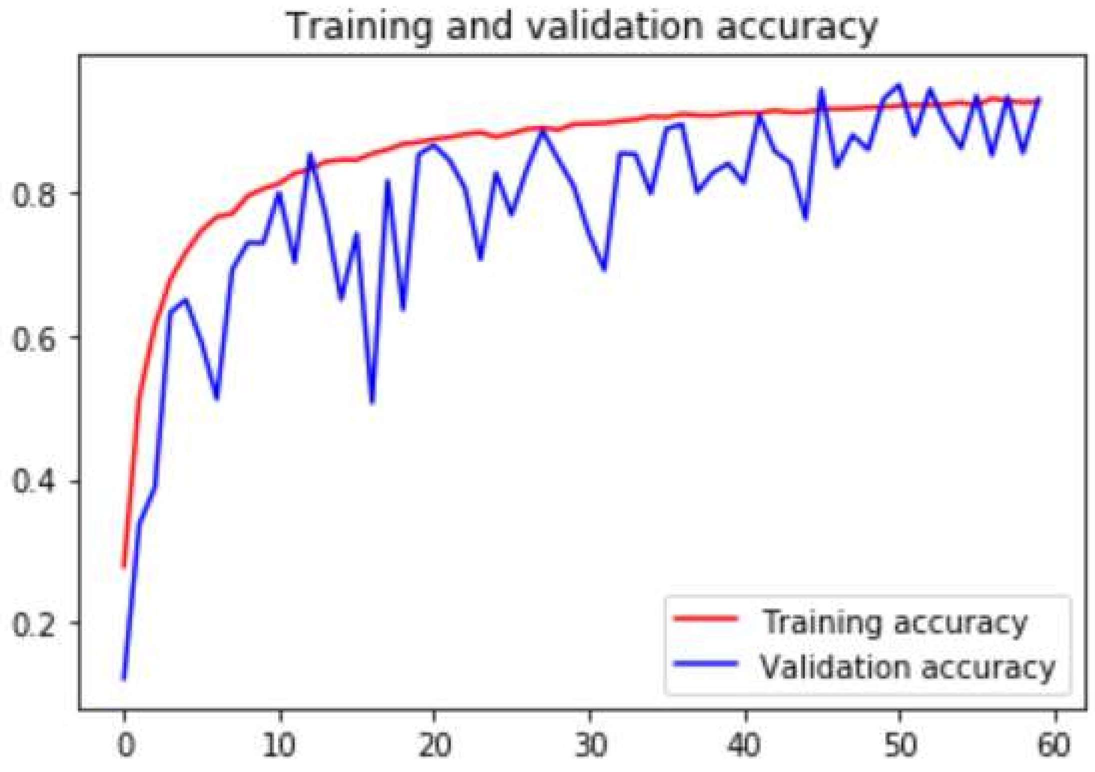


Figure 14 Final Model Accuracy Plot on ASL Dataset

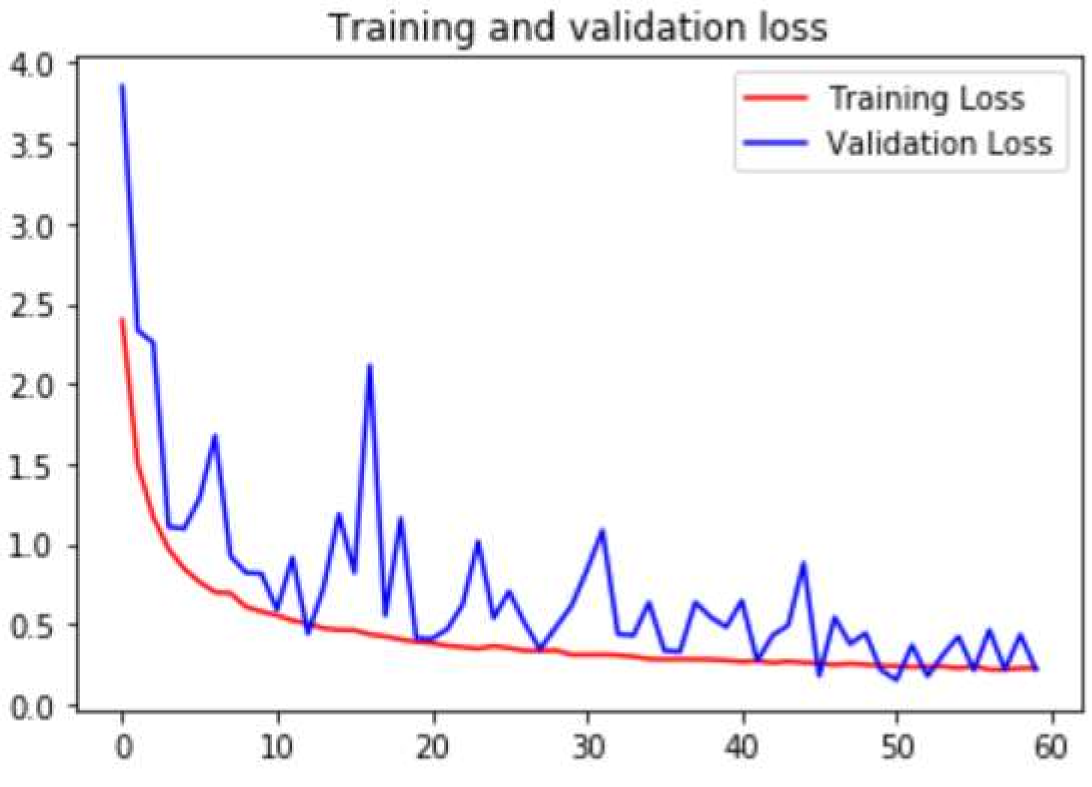


Figure 15 Final Model Loss Plot on ASL Dataset

3.6 Building the Application

The next and the final phase of the project is to build an interface to show the efficiency of the model. The cv2 package was used to start capturing and taking in frames and is pre-processed similar to how image dataset was created. From there, each frame is sent to the model created and the output class is mapped to the corresponding gesture. Then a function to overlay was developed which was used to overlay the target gesture on top of the hand being shown on the screen during live capture. Figure 15 is a screenshot of the working sample.

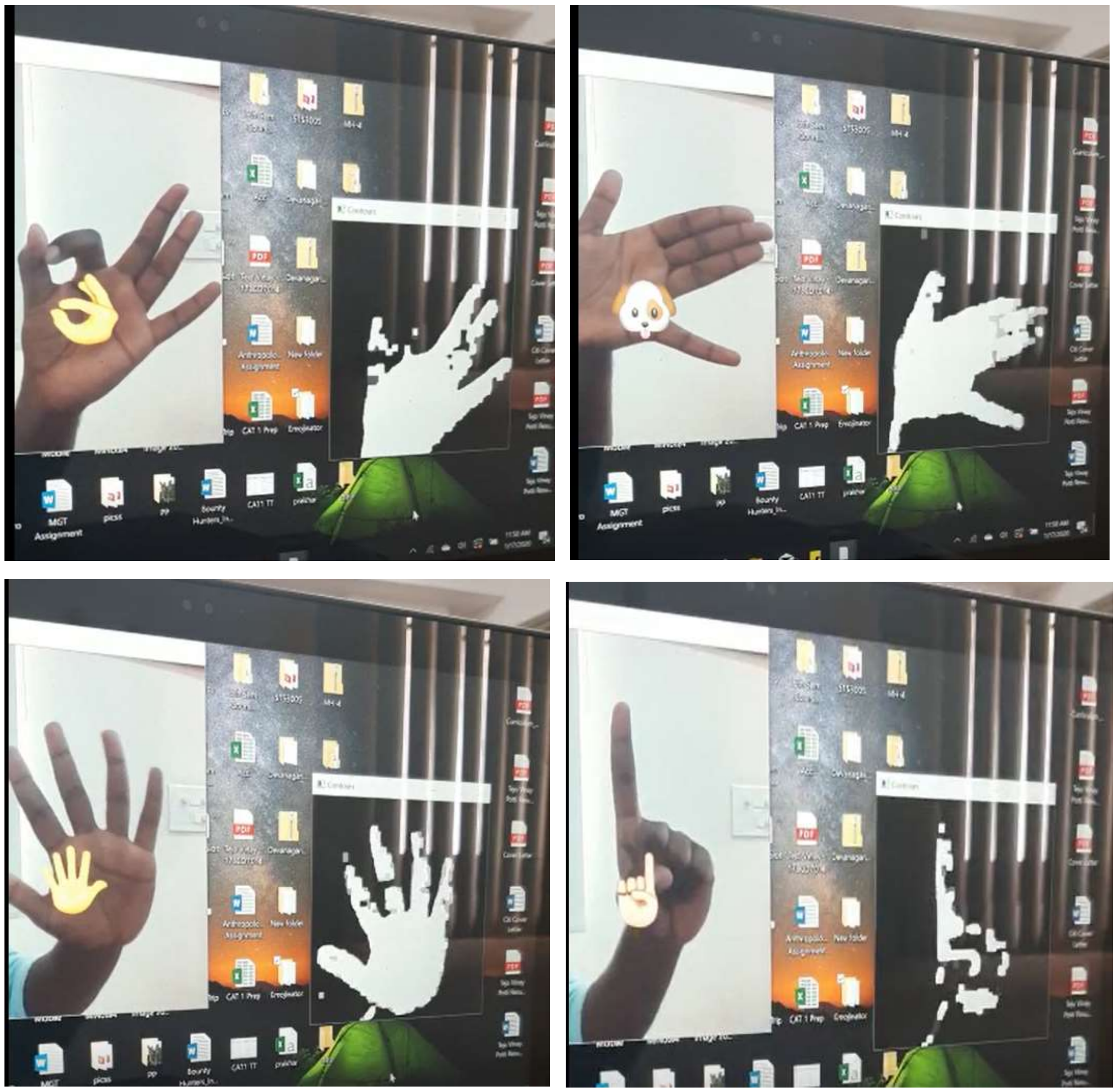


Figure 16 Working Samples

CHAPTER 4

CONCLUSION AND FUTURE WORK

Covid has brought a lot of change in the way we live. People are adapting to touchless contactless interactions. So, it is very important from now on to be ready to face any future pandemics and at the same time be extra safe while interacting. The need of the hour is to be able to implement contactless technologies.

Hand gesture recognition can play a very important role at this kind of situations. It has many applications in different domains. Some of its main applications include converting signs to speech, video gaming, robot control, virtual mouse control and a lot more. This gesture recognition can be expanded and even be implemented at public places where people need to press buttons or tap on screens.

Lot can be done in this area. There is a large scope which could be ventured, and new designs or system could be made to improve the conditions and efficiency of the recognition systems and by using AI we can figure out if in near future any of the component might need attention.

CHAPTER 5

APPENDIX

Creating Image Dataset Code

```
import numpy as np
import cv2
import os

def fol(f):
   if not os.path.exists(f):
      os.mkdir(f)

def store_im(id_gest):
   total=1200
   c=cv2.VideoCapture(0)
   xx,yy,ww,hh=300,50,350,350

   fol("gestures/"+ str(id_gest))
   p=0
   fc=False
   fr=0
   while True:
      re,fr=c.read()
      fr=cv2.flip(fr, 1)
      h=cv2.cvtColor(fr,cv2.COLOR_BGR2HSV)
      m2=cv2.inRange(h,np.array([2, 50, 60]),np.array([25, 150, 255]))
      r=cv2.bitwise_and(fr,fr,mask=m2)
      gr=cv2.cvtColor(r,cv2.COLOR_BGR2GRAY)
      med=cv2.GaussianBlur(gr,(5,5),0)
      kernSq=np.ones((5,5),np.uint8)
      dil=cv2.dilate(med,kernSq,it=2)
      op=cv2.morphologyEx(dil,cv2.MORPH_CLOSE,kernSq)
      re,th=cv2.threshold(op,30,255,cv2.THRESH_BINARY)
      th=th[yy:yy+h,xx:xx+ww]
      con = cv2.findContours(th.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)[1]

      if len(con)>0:
         co=max(con,key=cv2.conArea)
         if cv2.conArea(co)>10000 and fr>50:
            xx1, yy1, ww1, hh1 = cv2.boundingRect(co)
            pic += 1
            sim = th[yy1:yy1 + hh1, xx1:xx1 + ww1]
            sim = cv2.resize(sim, (50, 50))
            cv2.putText(fr, "Cap...",(10,40),cv2.FONT_HERSHEY_TRIPLEX,3,(125,255,255))
            cv2.imwrite("gestures/"+str(id_gest)+"/"+str(p)+".jpg",sim)
         cv2.rectangle(fr,(xx,yy),(xx+ww,yy+hh),(0,255,0),3)
```

```
        cv2.putText(fr,str(p),(10, 200),cv2.FONT_HERSHEY_TRIPLEX,1,(126, 125, 255))
        cv2.imshow("Capturing the given ges", fr)
        cv2.imshow("t", th)
        kp=cv2.waitKey(1)
        if kp==ord('c'):
            if fc==False:
                fc=True
            else:
                fc=False
                fr=0
        if fc==True:
            fr +=1
        if p==total:
            break
id_gest=input("Enter the ges number. : ")
st_im(id_gest)
```

Data Augmentation (Expansion) Code

```
import numpy as np
from tensorflow.keras.preprocessing.image import *
import os

aug = ImageDataGenerator(
    rotation_range=40,
    zoom_range=0.2,
    width_shift_range=0.1,
    height_shift_range=0.3,
    shear_range=0.2,
    horizontal_flip = True)

P=os.getcwd()
dp=P+'/gestures'
ddl=os.listdir(dp)
print(P)
print(dp)
print(ddl)

for ds in ddl:
    il=os.listdir(dp+'/'+ds)
    print ('Dataset Loaded -'+'{}\n'.format(ds))
    for i in il:
        ip=dp+'/'+ds+'/'+i
        i=load_img(ip)
        i=img_to_array(i)
        i=np.expand_dims(i,axis=0)
        t=0
        ig=aug.flow(i, batch_size=1, save_to_dir=PATH+'/gesture_exp/'+dataset,save_prefix="im",
save_format="jpg")
```

```
    for i in im_gen:
        t += 1
        if t == 5:
            break
```

Image Dataset to CSV

```
from scipy.misc import imread

root_dir = os.getcwd()
img_dir = os.path.join(root_dir, 'gesture_exp')

pixels = np.array(['pixel_{:04d}'.format(x) for x in range(7500)])
flag = True

for char_name in sorted(os.listdir(img_dir)):
    char_dir = os.path.join(img_dir, char_name)
    img_df = pd.DataFrame(columns=pixels)

    for img_file in sorted(os.listdir(char_dir)):
        image = pd.Series(imread(os.path.join(char_dir, img_file)).flatten(), index=pixels)
        img_df = img_df.append(image.T, ignore_index=True)

    img_df = img_df.astype(np.uint8)
    img_df['emoji'] = char_name

    img_df.to_csv('data_exp.csv', index=False, mode='a', header=flag)
    flag=False


    print('=', end='')
```

CNN Model Code

```
from keras import *
traindata=pd.read_csv('d_exp_tr.csv')
trainlabel=traindata['label'].values
traindata.drop('label',inplace=True,axis=1)
trainimages = traindata.values
trainimages=trainimages.reshape(-1,28,28,1)
testdata = pd.read_csv('data_exp_te.csv')
testlabel=testdata['label'].values
testdata.drop('label',inplace=True,axis=1)
testimages = testdata.values
testimages=testimages.reshape(-1,28,28,1)
```

```
traingen=ImageDataGenerator(rotation_range=30,zoom_range=0.3,width_shift_range=0.15,heigh
t_shift_range=0.15,
shear_range=0.3,horizontal_flip=True,rescale=1/255.0,validation_split=0.2)
traindata_generator = traingen.flow(trainimages,trainlabel,subset='training')
validationdata_generator = traingen.flow(trainimages,trainlabel,subset='validation')
testgen=ImageDataGenerator(rescale=1/255.0)
testdata_generator = testgen.flow(testimages,testlabel)

m=Sequential([])
m.add(Conv2D(64,(3,3),activation="relu",input_shape=(28,28,1)))
m.add(BatchNormalization())
m.add(MaxPooling2D(2,2))

m.add(Conv2D(128,(3,3),activation="relu"))
m.add(BatchNormalization())
m.add(MaxPooling2D(2,2))
m.add(Conv2D(128,(3,3),activation="relu"))
m.add(BatchNormalization())
m.add(MaxPooling2D(2,2))
m.add(Conv2D(128,(3,3),activation="relu"))
m.add(BatchNormalization())
m.add(MaxPooling2D(2,2))
m.add(Flatten())
m.add(BatchNormalization())
m.add(Dropout(0.4))
m.add(Dense(256,activation="relu"))
m.add(BatchNormalization())
m.add(Dense(26,activation="softmax"))
m.compile(loss="sparse_categorical_crossentropy",optimizer='adam',metrics=['accuracy'])
#filepath = "handEmo.h5"
m.summary()

class CB(cb):
  def endEp(self, epoch, logs={}):
    if(logs.get('accuracy')>0.985):
      print("\nReached 98.5% accuracy so cancelling training!")
      self.model.stop_training=True
cb=CB()

h=model.fit(traindata_generator,epochs=60,validation_data=validationdata_generator,callbacks=[
cb])
a=h.h['accuracy']
va=h.h['val_accuracy']
l=h.h['loss']
vlos=h.h['val_loss']
eps=range(len(a))
p.plot(eps,a,'r',label='Acc Tr')
p.plot(eps,va,'b',label='Acc Val')
p.title('Tr and Val Accuracy')
```

```
p.legend()
p.figure()
p.plot(eps,l,'r',label='Loss Tr')
p.plot(eps,vlos,'b',label='Loss Val')
p.title('Tr and Val Loss')
p.legend()
p.show()
print("Acc Test: "+ str(model.evaluate_generator(testdata_generator)[1]*100))
```

Model Evaluation on Benchmark Dataset Code

```
from keras import *
td=pd.read_csv('sign_mnist_tr.csv')
trainlabel=td['label'].values
traindata.drop('label',inplace=True,axis=1)
trainimages = traindata.values
trainimages=trainimages.reshape(-1,28,28,1)
testdata = pd.read_csv('sign_mnist_test.csv')
testlabel=testdata['label'].values
testdata.drop('label',inplace=True,axis=1)
testimages = testdata.values
testimages=testimages.reshape(-1,28,28,1)

traingen=ImageDataGenerator(rotation_range=30,zoom_range=0.3,width_shift_range=0.15,heigh
t_shift_range=0.15,shear_range=0.3,horizontal_flip=True,rescale=1/255.0,validation_split=0.2)
traindata_generator=traingen.flow(trainimages,trainlabel,subset='training')
validationdata_generator=traingen.flow(trainimages,trainlabel,subset='validation')
testgen=ImageDataGenerator(rescale=1/255.0)
testdata_generator=testgen.flow(testimages,testlabel)

m = Sequential()
m.add(Conv2D(64,(3,3),activation="relu",input_shape=(28,28,1)))
m.add(BatchNormalization())
m.add(MaxPooling2D(2,2))
m.add(Conv2D(128,(3,3),activation="relu"))
m.add(BatchNormalization())
m.add(MaxPooling2D(2,2))
m.add(Conv2D(128,(3,3),activation="relu"))
m.add(BatchNormalization())
m.add(MaxPooling2D(2,2))
m.add(Flatten())
m.add(BatchNormalization())
m.add(Dropout(0.2))
m.add(Dense(256,activation="relu"))
m.add(BatchNormalization())
m.add(Dense(26,activation="softmax"))
m.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
m.summary()
```

```
h=model.fit(traindata_generator,epochs=60,validation_data=validationdata_generator,callbacks=[
cb])

a=h.h['Acc']
va=h.h[‘Acc Val’]
l=h.h['Loss']
valos=h.h['Loss Val']
eps=range(len(a))
p.plot(eps,acc,'r',label='Acc Tr')
p.plot(eps,va,'b',label='Acc Val')
p.title('Acc Tr and Val')
p.legend()
p.figure()
p.plot(eps,l,'r',label='Loss Tr')
p.plot(eps,valos,'b',label='Loss Val')
p.title('Loss Tr and Val')
p.legend()
p.show()
print("test accuracy: "+ str(model.evaluate_generator(testdata_generator)[1]*100))
```

Result/Application Code

```
from keras.models import load_model
m = load_model('handEmo.h5')

def getEmo():
   emo_fol = 'handEmo/'
   ems = []
   for em in range(len(os.listdir(emo_fol))):
      emos.append(cv2.imread(emo_fol+str(em)+'.png', -1))
      print(em)
   return emos

def pred(m, im):
   proc = keras_process_image(im)
   probability = model.predict(processed)[0]
   cl = list(pred_probability).index(max(probability))
   return max(probability,cl)

def im_pro(im):
   im = cv2.resize(im, (50, 50))
   im = np.array(im, dtype=np.float32)
   im = np.reshape(im, (-1, 50, 50, 1))
   return im

def overlay(im, em, xx,yy,ww,hh):
   em = cv2.resize(em, (ww, hh))
   try:
      im[yy:yy+hh,xx:xx+ww] = blendTrans(im[yy:yy+hh,xx:xx+ww],em)
   except:
      pass
   return im

def blendTrans(f_im, o_t_img):
   o_im = o_t_img[:,:,:3]
   om = o_t_img[:,:,3:]



   bgm = 255 - om
   om=cv2.cvtColor(om,cv2.COLOR_GRAY2BGR)
   bgm=cv2.cvtColor(bgm,cv2.COLOR_GRAY2BGR)
   fp=(f_im*(1/255.0))*(bgm*(1/255.0))
   op=(o_im*(1/255.0))*(om*(1/255.0))
   return np.uint8(cv2.addWeighted(fp,255.0,op,255.0,0.0))
emos = get_emos()
c=cv2.VideoCapture(0)
xx,yy,ww,hh=300,50,325,325

while (c.isOpened()):
   #repeat code as in preprocessing
   if len(cons) > 0:
      con = max(cons, key=cv2.contourArea)
      if cv2.contourArea(con)>2500:
          xx,yy,ww1,hh1 = cv2.boundingRect(con)
          n_im = th[yy:yy+hh1,xx:xx+ww1]
          n_im = cv2.resize(n_im,(50, 50))
          pred_probability,pred_cl = predict(m, n_im)
          print(pred_cl, pred_probability)
          im = overlay(im, emos[pred_cl],400,250,90,90)

   xx,yy,ww,hh=300,50,325,325
   cv2.imshow("Frame", im)
   cv2.imshow("Contours", th)
   pk=cv2.waitKey(10)
   if pk==27:
      break
```

CHAPTER 6

REFERENCES

[1] Gesture Recognition Technology (engineersgarage.com)

[2] Kinect (ict.ac.cn)

[3] Vladimir I. Pavlovic, "Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review. " IEEE Transactions on Pattern Analysis and Machine Intelligence 19.7 (1997) pavlovic97pami.pdf (rutgers.edu)

[4] Tran, Dinh-Son, et al. "Real-time hand gure spotting and recognition using RGB-D camera and 3D convolutional neural network." Applied Sciences 10.2 (2020): 722.

[5] Chen, Lin, et al. "Hand gesture recognition using compact CNN via surface electromyography signals." Sensors 20.3 (2020): 672.

[6] Pinto, Raimundo F., et al. "Static hand gesture recognition based on convolutional neural networks." Journal of Electrical and Computer Engineering 2019 (2019).

[7] RGB to HSV conversion | Color conversion (rapidtables.com)

[8] Tutorial: Skin Detection Example using Python and OpenCV (pyimagesearch.com)

[9] OpenCV: Smoothing Images

[10] Sign Language MNIST | Kaggle

[11] Transparent overlays with OpenCV - PyImageSearch

[12] Real-time object detection with deep learning and OpenCV - PyImageSearch

[13] An Intuitive Explanation of Convolutional Neural Networks – the data science blog (ujjwalkarn.me)

[14] Building powerful image classification models using very little data (keras.io)

[15] Transfer Learning in Keras with Computer Vision Models (machinelearningmastery.com)

BIODATA

Name	: Tejo Vinay Potti
Mobile Number	: 9885735065
E-mail	: tejo.vinaypotti@vitap.ac.in
Permanaent Address	: A 708, Vertex Sadguru Krupa, Nizampet Rd, Hyd