Project Report on

Sign Decoder: A Sign Language Translator

Submitted in partial fulfillment for the award of **Post Graduate Diploma in Big Data Analytics** from **C-DAC ACTS (Pune)**

Guided by:

Mr. Himanshu Jagtap

Presented by:

Monika Dwivedi	**200940183028**
Neha Patil	**200940183031**
Saloni Sonawane	**200940183043**
Shubhangi Shelar	**200940183051**
Suraj Mahajan	**200940183054**

Centre of Development of Advanced Computing (C-DAC), Pune

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

Monika Dwivedi	**200940183028**
Neha Patil	**200940183031**
Saloni Sonawane	**200940183043**
Shubhangi Shelar	**200940183051**
Suraj Mahajan	**200940183054**

have successfully completed their project on

Sign Decoder: A Sign Language Translator

under the guidance of Mr. Himanshu Jagtap

Project Guide **Project Supervisor**

ACKNOWLEDGEMENT

This project "**Sign Decoder: A Sign Language Translator**" was a great learning experience for us and we are submitting this work to Advanced Computing Training School (CDAC ACTS). We all are very glad to mention the name of *Mr.Himanshu Jagtap* for his valuable guidance to work on this project. His guidance and support helped us to overcome various obstacles and intricacies during the course of project work.

We are highly grateful to Ms. Risha P.R. (Manager (ACTS training Centre), C-DAC), for her guidance and support whenever necessary while doing this course Post Graduate Diploma in *Big Data Analytics (PGDBDA)* through C-DAC ACTS, Pune.

Our most heartfelt thanks go to *Ms.Seema Sanjeevan* (Course Coordinator, *EDBDA*) who gave all the required support and kind coordination to provide all the necessities like required hardware, internet facility and extra Lab hours to complete the project and throughout the course up to the last day here in C-DAC ACTS, Pune.

Monika Dwivedi **200940183028**

Neha Patil **200940183031**

Saloni Sonawane **200940183043**

Shubhangi Shelar **200940183051**

Suraj Mahajan **200940183054**

Sr.No.	Content	Page No.
1	Abstract	5
2	Introduction & overview of Project	6
3	System Overview	7
4	Data Collection	12
5	Train Model	15
6	Predict Model	17
7	Output Screenshots	19
8	Conclusion	22

Chapter 1

Abstract

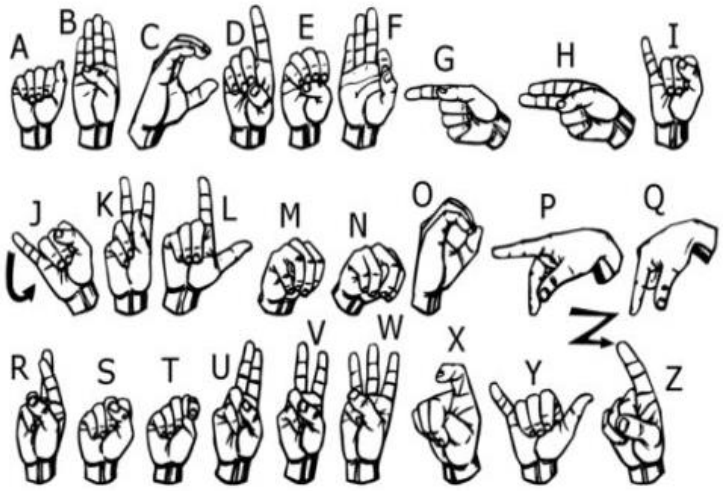
Communication has a huge impact on people's lives but not everyone can do communication. For deaf/dumb people communication becomes a huge barrier to survival in society. Sign language is mostly used for the communication of deaf and dumb people. Sign language is an efficient way for deaf people to communicate with others but normal people can't understand their sign language due to lack of knowledge. Furthermore, this will cause more problems for deaf/dumb people when they start involving in the educational, social and work environment. The goal of this paper is to design a system that is used to communicate between "normal", "deaf/dumb" people. This System has two modules, in the first module, converting the sign language into Text using deep learning techniques. In this project designing a real-time system for the recognition of some meaningful shapes made using hands. The main purpose of this project is to remove the barrier between deaf/ dumb and normal people.

Keywords— Sign Language, Deep Learning, Convolutional Neural Network (CNN)

Chapter 2

Introduction and Overview of Project

The purpose of this system to represent a real time system based on American Sign Language (ASL) recognition with greater accuracy. Sign language is the primary language of the people who are deaf or hard of hearing and also used by them who can hear but cannot physically speak. The problem arises when deaf and dumb people try to communicate using this language with the people who are unaware of this language grammar. So, it becomes necessary to develop an automatic and interactive interpreter to understand them.



The main objective of this project is to design a system that can assists the impaired people to communicate with normal people. This project also aims to meet the following objectives:

1. To develop gesture recognizing system that can recognize Sign Language and translate it into text.

2. To test the accuracy of the system.

We are going to develop following modules:

1. Hand action recognition and text generation

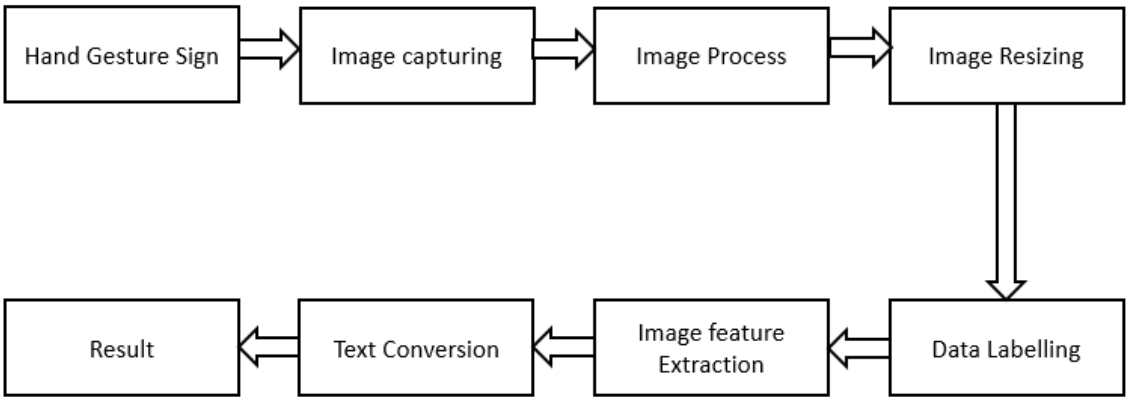
• We are going to overcome existing communication barrier by providing two-way communications for deaf and dumb peoples.

Chapter 3

System Overview

The proposed system overcomes the problem of communication between normal people and deaf/dumb people.

- Sign Language to Text Conversion: This module is the interpretation of the sign language into a text. In this part of the system, the sign gestures are given as input and it is converted to text. This module is developed for normal people to understand the sign language of the deaf/dumb people.



The first module is the interpretation of the sign language into a text. In this part of the system, the sign gestures are given as input and it is converted to voice. This module is developed for normal people to understand the sign language of the deaf/dumb people.

Once the hand gesture is recognized by the system, basic steps of image pre-processing i.e., Data cleaning, image resizing, data labelling are performed. The object detected is then forwarded to the classifier. The CNN classifier classifies or detects the sign language and then this recognized sign is then converted into the voice.

A. Data Preparation: -

1. Data Cleaning: Data cleaning plays an important part in the development of a model. Your data quality is critical to getting to the final analysis. Any data that tends to be incomplete, noisy and inconsistent can affect your outcome. The method of identifying and deleting missing or incorrect information from a record set, table or database is data cleaning. Cleaning up the data is usually the most time-consuming aspect of the process of data preparation, but it is

important to delete inaccurate data and fill in gaps. Professional data scientists typically spend a substantial amount of their time on this step.

2. Image Resizing: Image resizing refers to image scaling. Scaling is useful in many applications for image processing as well as for machine learning. It helps to reduce the number of pixels from an image and has several benefits. It may reduce a neural network's training time, as more is the number of pixels in an image, more is the number of input nodes, which in turn increases the model's complexity. It helps to zoom in images, too. Several times we have to resize the file, i.e., either shirk it or scale it up to meet the requirements for size. Image interpolation emerges when you adjust the image or distort it from the one-pixel grid to another. Image resizing is necessary when the total number of pixels needs to be increased or decreased, while remapping can occur when correcting lens distortion or rotation of an image.

3. Data Labelling: Data labelling is an important part of ML data pre-processing, especially for supervised learning, in which both input and output data are labelled as being classified to provide a learning basis for future data processing. To produce a quality algorithm, the labels used to define data features have to be accurate, unbiased and independent. A properly labelled dataset offers a ground truth that the ML model uses to test for accuracy in its predictions and continue to refine its algorithm. Labelling typically takes on a set of unlabelled data and increases every piece of that unlabelled data with meaningful informative tags.

B. Model Training: -

1. TensorFlow: Currently, Google's TensorFlow is the world's most famous deep learning library. Google's software uses machine learning to enhance the search engine, translation, image captioning and suggestions in all of its products. It is a library of symbolic maths and is also used for machine learning applications such as neural networks. TensorFlow is a computational framework for the development of a machine learning model. TensorFlow offers a variety of different toolkits that allow you to build models at your preferred abstraction level. By defining a series of mathematical operations, you may use lower-level APIs to build models. Alternatively, high-level APIs (such as tf.estimator) can be used to specify predefined architectures, such as linear regressors or neural networks.

Keras is a high-level neural networks library, written in Python and capable of running on top of either TensorFlow or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through total modularity, minimalism, and extensibility).

- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Supports arbitrary connectivity schemes (including multi-input and multi-output training).

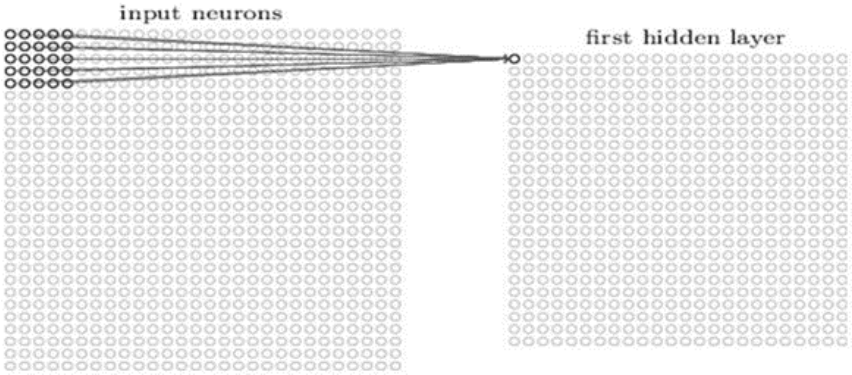
C. Convolutional Neural Network:

Convolutional Neural Networks have been successful used in image recognition and classification. It has been successfully implemented for human gesture recognition. In the area of sign language recognition, in particular, work has been done using deep CNNs, with input recognition which is sensitive to more than just pixels of images. Using cameras that sense depth and contour, the process is made much easier for each sign language action by creating characteristic depth and motion profiles.

Four main layer working approach of CNN explained below: -

a. Convolutional Layer:

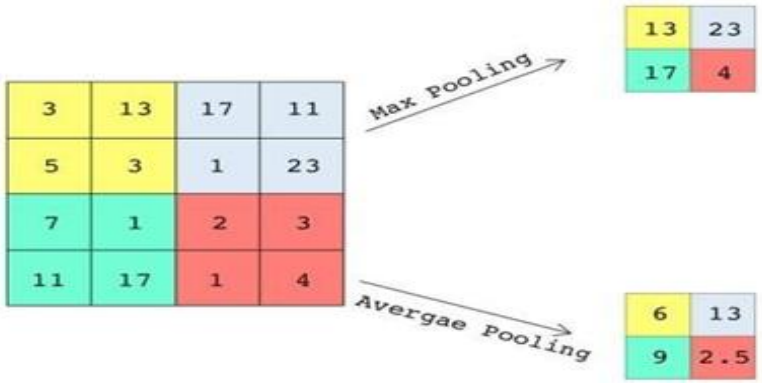
Convolution is the first layer where features are extracted from an input image. Convolution maintains the relation between pixels by using small squares of input data to learn image features. It is a mathematical operation that involves two inputs such as a matrix of images and a filter or kernel. The convolution operation involving a defined filter over the input image and recording the results you can get an output image that tells you where the vertical edges. The convolution operation works by taking the filter and laying it over there starting in one corner, laying this over and multiplying each element by the corresponding element in the filter and summing them the result will be placed in the first corner of the filter. There is the sum of the element-wise multiplication of all numbers. This process continues until the pattern is complete.



b. Pooling Layer:

Pooling layers are used to reduce the size of the representation. This layer will decrease the number of parameters when the images are too large. Max Pooling is the

most common type of pooling layer used; the less common Average Pooling is sometimes seen in very deep neural networks. The operation of Max Pooling involves taking the maximum value while the average pooling, the average.

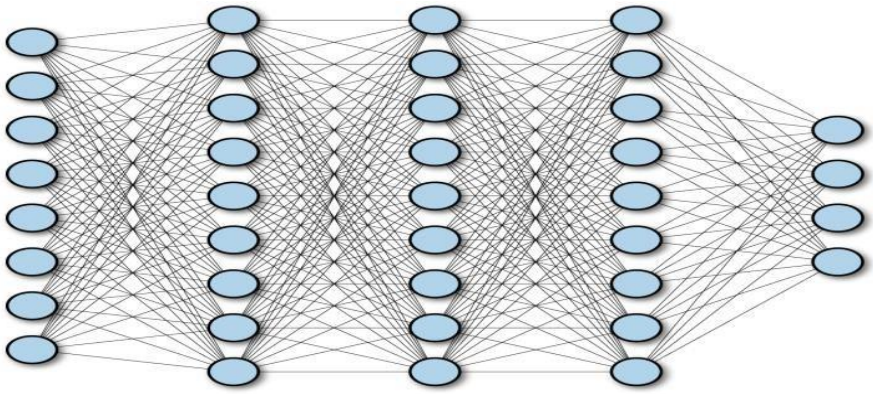


c. Flattening:

Flattening transforms the data into a 1-dimensional array to be passed to the next layer. Create a single long feature vector, we flatten the output of the convolution layers. And it's connected to the final model of classification, which is called a fully connected layer. In other words, we are placing all the pixel data in one line and making connections to the end layer.

d. Fully Connected Layer:-

Fully connected layers are those where each node is connected to every previous and every next node. The number of nodes is irrelevant for the full connectedness. Fully Connected Layer is the actual component in a Deep Neural Network that does discriminative learning. It is a simple multilayer perceptron that can learn weights to identify a class of objects.



Chapter 4

Data Collection

Data collection is the process of gathering and measuring information on targeted variables in an established system, which then enables one to answer relevant questions and evaluate outcomes. Data collection is a research component in all study fields, including physical and social sciences, humanities, and business. While methods vary by discipline, the emphasis on ensuring accurate and honest collection remains the same. The goal for all data collection is to capture quality evidence that allows analysis to lead to the formulation of convincing and credible answers to the questions that have been posed.

In [1]:

```
import cv2
import numpy as np
import os

# Create the directory structure
if not os.path.exists("/Users/salonisonawane/Documents/runkr/data"):
    os.makedirs("/Users/salonisonawane/Documents/runkr/data")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/test")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/a")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/b")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/c")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/d")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/e")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/f")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/g")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/h")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/i")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/j")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/k")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/l")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/m")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/n")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/o")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/p")
    os.makedirs("/Users/salonisonawane/Documents/runkr/data/train/q")
```

```
cap = cv2.VideoCapture(0)

while True:
    _, frame = cap.read()
    # Simulating mirror image
    frame = cv2.flip(frame, 1)

    # Getting count of existing images
    count = {'a': len(os.listdir(directory+"/a")),
             'b': len(os.listdir(directory+"/b")),
             'c': len(os.listdir(directory+"/c")),
             'd': len(os.listdir(directory+"/d")),
             'e': len(os.listdir(directory+"/e")),
             'f': len(os.listdir(directory+"/f")),
             'g': len(os.listdir(directory+"/G")),
             'h': len(os.listdir(directory+"/H")),
             'i': len(os.listdir(directory+"/I")),
             'j': len(os.listdir(directory+"/J")),
             'k': len(os.listdir(directory+"/K")),
             'l': len(os.listdir(directory+"/L")),
             'm': len(os.listdir(directory+"/M")),
             'n': len(os.listdir(directory+"/N")),
             'o': len(os.listdir(directory+"/O")),
             'p': len(os.listdir(directory+"/P")),
```

```
    cv2.putText(frame, "p : "+str(count['p']), (10, 240), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "q : "+str(count['q']), (10, 250), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "r : "+str(count['r']), (10, 260), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "s : "+str(count['s']), (10, 270), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "t : "+str(count['t']), (10, 280), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "u : "+str(count['u']), (10, 290), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "v : "+str(count['v']), (10, 300), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "w : "+str(count['w']), (10, 310), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "x : "+str(count['x']), (10, 320), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "y : "+str(count['y']), (10, 330), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "z : "+str(count['z']), (10, 340), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    # Coordinates of the ROI
    x1 = int(0.5*frame.shape[1])
    y1 = 10
    x2 = frame.shape[1]-10
    y2 = int(0.5*frame.shape[1])
    # Drawing the ROI
    # The increment/decrement by 1 is to compensate for the bounding box
    cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (255,0,0) ,1)
    # Extracting the ROI
    roi = frame[y1:y2, x1:x2]
    roi = cv2.resize(roi, (64, 64))

    cv2.imshow("Frame", frame)
```

Computer Vision

Computer vision is a process by which we can understand the images and videos how they are stored and how we can manipulate and retrieve data from them. Computer Vision is the base or mostly used for Artificial Intelligence. Computer-Vision is playing a major role in self-driving cars, robotics as well as in photo correction apps.

OpenCV

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as Numpy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

- **cv2.rectangle()** method is used to draw a rectangle on any image.

  Syntax: cv2.rectangle(image, start_point, end_point, color, thickness)
  Parameters:
  image: It is the image on which rectangle is to be drawn.
  start_point: It is the starting coordinates of rectangle. The coordinates are represented as tuples of two values i.e. (**X** coordinate value, **Y** coordinate value).
  end_point: It is the ending coordinates of rectangle. The coordinates are represented as tuples of two values i.e. (**X** coordinate value, **Y** coordinate value).
  color: It is the color of border line of rectangle to be drawn. For **BGR**, we pass a tuple. eg: (255, 0, 0) for blue color.
  thickness: It is the thickness of the rectangle border line in **px**. Thickness of **-1 px** will fill the rectangle shape by the specified color.
  Return Value: It returns an image.

- **cv2.cvtColor()** method is used to convert an image from one color space to another.

  Syntax: cv2.cvtColor(src, code[, dst[, dstCn]])
  Parameters:
  src: It is the image whose color space is to be changed.
  code: It is the color space conversion code.
  dst: It is the output image of the same size and depth as src image. It is an optional parameter.
  dstCn: It is the number of channels in the destination image. If the parameter is 0 then the number of the channels is derived automatically from src and code. It is an optional parameter.
  Return Value: It returns an image.

Chapter 5

Train Model

The process of training an ML model involves providing an ML algorithm (that is, the *learning algorithm*) with training data to learn from. The term *ML model* refers to the model artifact that is created by the training process.

The training data must contain the correct answer, which is known as a *target* or *target attribute*. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict), and it outputs an ML model that captures these patterns.

You can use the ML model to get predictions on new data for which you do not know the target. For example, let's say that you want to train an ML model to predict if an email is spam or not spam. You would provide Amazon ML with training data that contains emails for which you know the target (that is, a label that tells whether an email is spam or not spam). Amazon ML would train an ML model by using this data, resulting in a model that attempts to predict whether new email will be spam or not spam.

To train an ML model, you need to specify the following:

- Input training data source
- Name of the data attribute that contains the target to be predicted
- Required data transformation instructions
- Training parameters to control the learning algorithm

In [2]:
```
# Importing the Keras libraries and packages
import tensorflow
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, MaxPool2D
# Step 1 - Building the CNN

# Initializing the CNN
classifier = keras.Sequential()

# First convolution layer and pooling
classifier.add(tensorflow.keras.layers.Conv2D(32, (3, 3), input_shape=(64, 64, 1), activation='relu'))
classifier.add(tensorflow.keras.layers.MaxPool2D(pool_size=(2, 2)))
# Second convolution layer and pooling
classifier.add(tensorflow.keras.layers.Conv2D(32, (3, 3), activation='relu'))
# input_shape is going to be the pooled feature maps from the previous convolution layer
classifier.add(tensorflow.keras.layers.MaxPool2D(pool_size=(2, 2)))

# Flattening the layers
classifier.add(tensorflow.keras.layers.Flatten())

# Adding a fully connected layer
classifier.add(tensorflow.keras.layers.Dense(units=128, activation='relu'))
classifier.add(tensorflow.keras.layers.Dense(units=26, activation='softmax')) # softmax for more than 2

# Compiling the CNN
classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']) # categorical_crossentropy for more t


# Step 2 - Preparing the train/test data and training the model

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
```

```
# Step 2 - Preparing the train/test data and training the model

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

training_set = train_datagen.flow_from_directory('D:/eDBDA_ProjectGroup11/data/train',
                                                 target_size=(64, 64),
                                                 batch_size=5,
                                                 color_mode='grayscale',
                                                 class_mode='categorical')

test_set = test_datagen.flow_from_directory('D:/eDBDA_ProjectGroup11/data/test',
                                            target_size=(64, 64),
                                            batch_size=5,
                                            color_mode='grayscale',
                                            class_mode='categorical')
classifier.fit_generator(
        training_set,
        steps_per_epoch=None, # No of images in training set
        epochs=10,
        validation_data=test_set,
        validation_steps=30)# No of images in test set


# Saving the model
model_json = classifier.to_json()
with open("D:/eDBDA_ProjectGroup11/model-bw.json", "w") as json_file:
    json_file.write(model_json)
classifier.save_weights('D:/eDBDA_ProjectGroup11/model-bw.h5')
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

```
classifier.save_weights('D:/eDBDA_ProjectGroup11/model-bw.h5')
```

```
Found 16221 images belonging to 26 classes.
Found 0 images belonging to 26 classes.
Epoch 1/10
3245/3245 [==============================] - 177s 54ms/step - loss: 0.6400 - accuracy: 0.8250
Epoch 2/10
3245/3245 [==============================] - 90s 28ms/step - loss: 0.0356 - accuracy: 0.99020s - loss: 0.0356 - accuracy
Epoch 3/10
3245/3245 [==============================] - 71s 22ms/step - loss: 0.0185 - accuracy: 0.9951
Epoch 4/10
3245/3245 [==============================] - 73s 23ms/step - loss: 0.0148 - accuracy: 0.9955
Epoch 5/10
3245/3245 [==============================] - 75s 23ms/step - loss: 0.0091 - accuracy: 0.9968
Epoch 6/10
3245/3245 [==============================] - 74s 23ms/step - loss: 0.0092 - accuracy: 0.9969
Epoch 7/10
3245/3245 [==============================] - 107s 33ms/step - loss: 0.0111 - accuracy: 0.9967
Epoch 8/10
3245/3245 [==============================] - 94s 29ms/step - loss: 0.0047 - accuracy: 0.9983
Epoch 9/10
3245/3245 [==============================] - 80s 25ms/step - loss: 0.0052 - accuracy: 0.9983
Epoch 10/10
3245/3245 [==============================] - 70s 21ms/step - loss: 0.0061 - accuracy: 0.9980
```

In [ ]:

Chapter 5

Predict Model

"Prediction" refers to the output of an algorithm after it has been trained on a historical dataset and applied to new data when forecasting the likelihood of a particular outcome, such as whether or not a customer will churn in 30 days. The algorithm will generate probable values for an unknown variable for each record in the new data, allowing the model builder to identify what that value will most likely be. If classification is about separating data into classes, prediction is about fitting a shape that gets as close to the data as possible.

```
import numpy as np
from tensorflow.keras.models import model_from_json
import operator
import cv2
import sys, os

# Loading the model
json_file = open("model-bw.json", "r")
model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(model_json)
# Load weights into new model
loaded_model.load_weights("model-bw.h5")
print("Loaded model from disk")

cap = cv2.VideoCapture(0)

# Category dictionary
categories = {0 : 'a', 1: 'b', 2: 'c', 3: 'd', 4: 'e', 5: 'f', 6: 'g', 7: 'h', 8: 'i', 9: 'j', 10: 'k', 11: 'l', 12: 'm', 13

while True:
    _, frame = cap.read()
    # Simulating mirror image
    frame = cv2.flip(frame, 1)

    # Got this from collect-data.py
    # Coordinates of the ROI
    x1 = int(0.5*frame.shape[1])
```

```
    # Simulating mirror image
    frame = cv2.flip(frame, 1)

    # Got this from collect-data.py
    # Coordinates of the ROI
    x1 = int(0.5*frame.shape[1])
    y1 = 10
    x2 = frame.shape[1]-10
    y2 = int(0.5*frame.shape[1])
    # Drawing the ROI
    # The increment/decrement by 1 is to compensate for the bounding box
    cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (255,0,0) ,1)
    # Extracting the ROI
    roi = frame[y1:y2, x1:x2]

    # Resizing the ROI so it can be fed to the model for prediction
    roi = cv2.resize(roi, (64, 64))
    roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    _, test_image = cv2.threshold(roi, 120, 255, cv2.THRESH_BINARY)
    cv2.imshow("test", test_image)
    # Batch of 1
    result = loaded_model.predict(test_image.reshape(1, 64, 64, 1))
    prediction = {'a': result[0][0],
                  'b': result[0][1],
                  'c': result[0][2],
                  'd': result[0][3],
                  'e': result[0][4],
                  'f': result[0][5],
                  'g': result[0][6],
                  'h': result[0][7]
```

```
                  't': result[0][19],
                  'u': result[0][20],
                  'v': result[0][21],
                  'w': result[0][22],
                  'x': result[0][23],
                  'y': result[0][24],
                  'z': result[0][25],
                 }
    # Sorting based on top prediction
    prediction = sorted(prediction.items(), key=operator.itemgetter(1), reverse=True)

    # Displaying the predictions
    cv2.putText(frame, prediction[0][0], (10, 120), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.imshow("Frame", frame)

    interrupt = cv2.waitKey(10)
    if interrupt & 0xFF == 27: # esc key
        break


cap.release()
cv2.destroyAllWindows()
```

- **cv2.VideoCapture():**

  we can capture a video from the camera. It lets you create a video capture object which is helpful to capture videos through webcam and then you may perform desired operations on that video.

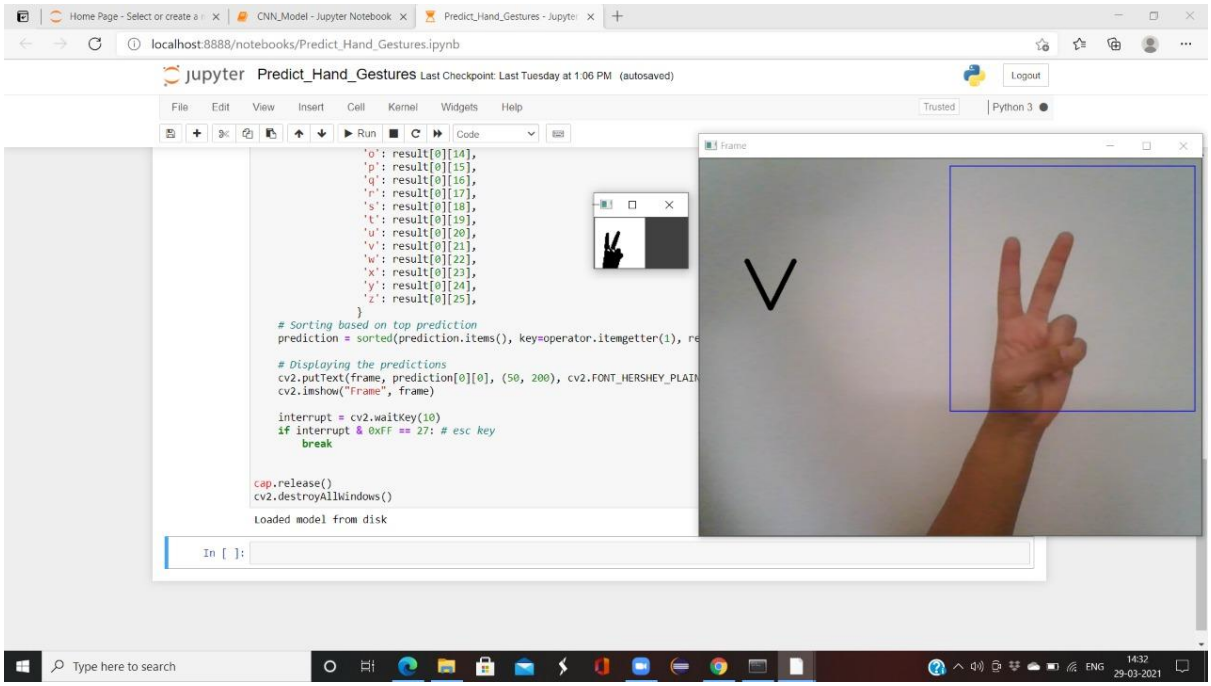
- **cv2.flip():**

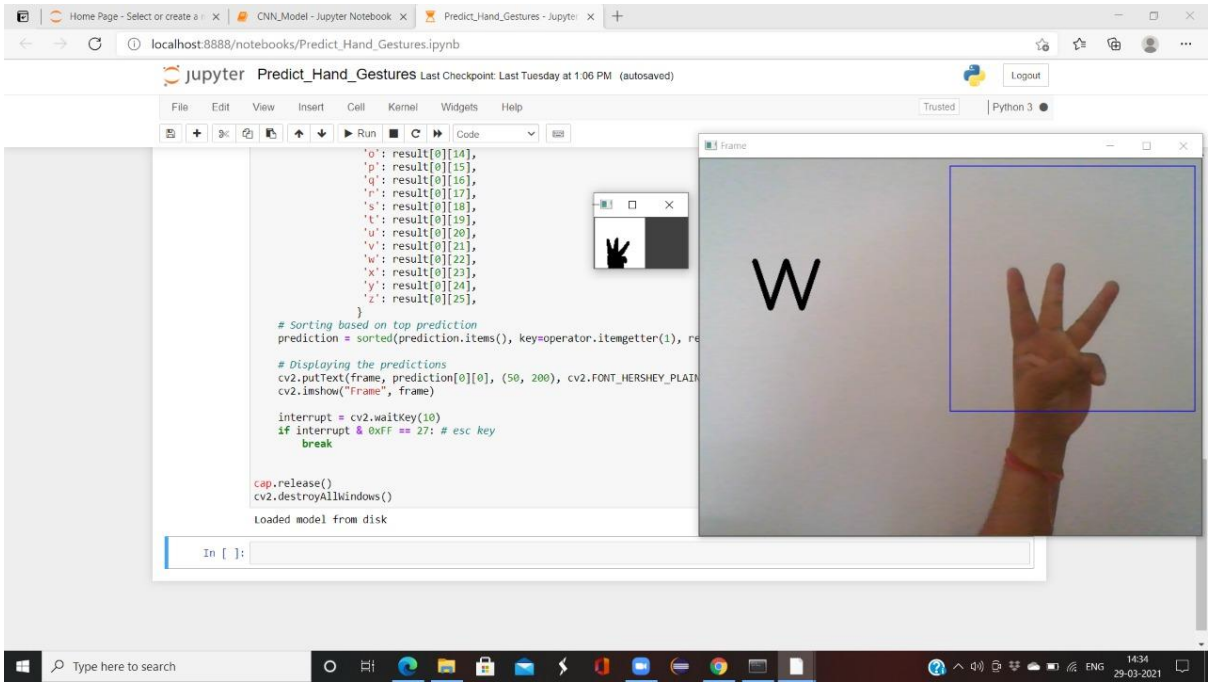
  method is used to flip a 2D array. The function cv:flip flips a 2D array around vertical, horizontal, or both axes.

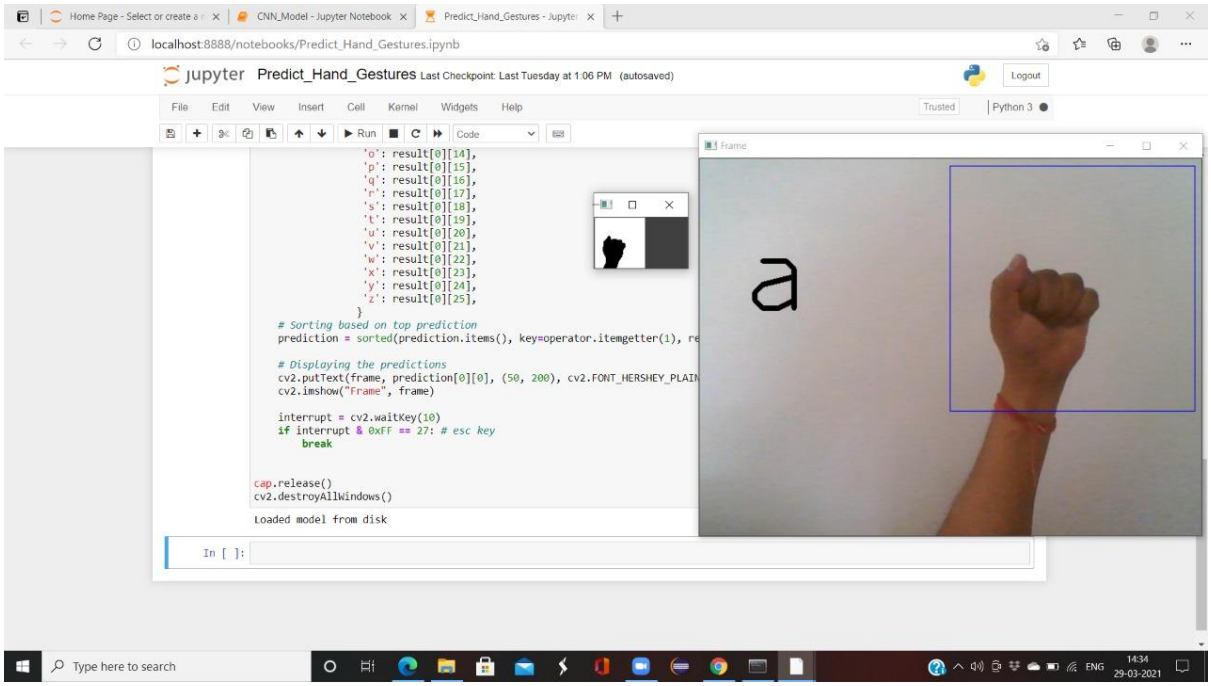
- **cv2.imshow()**:
  method is used to display an image in a window. The window automatically fits to the image size.

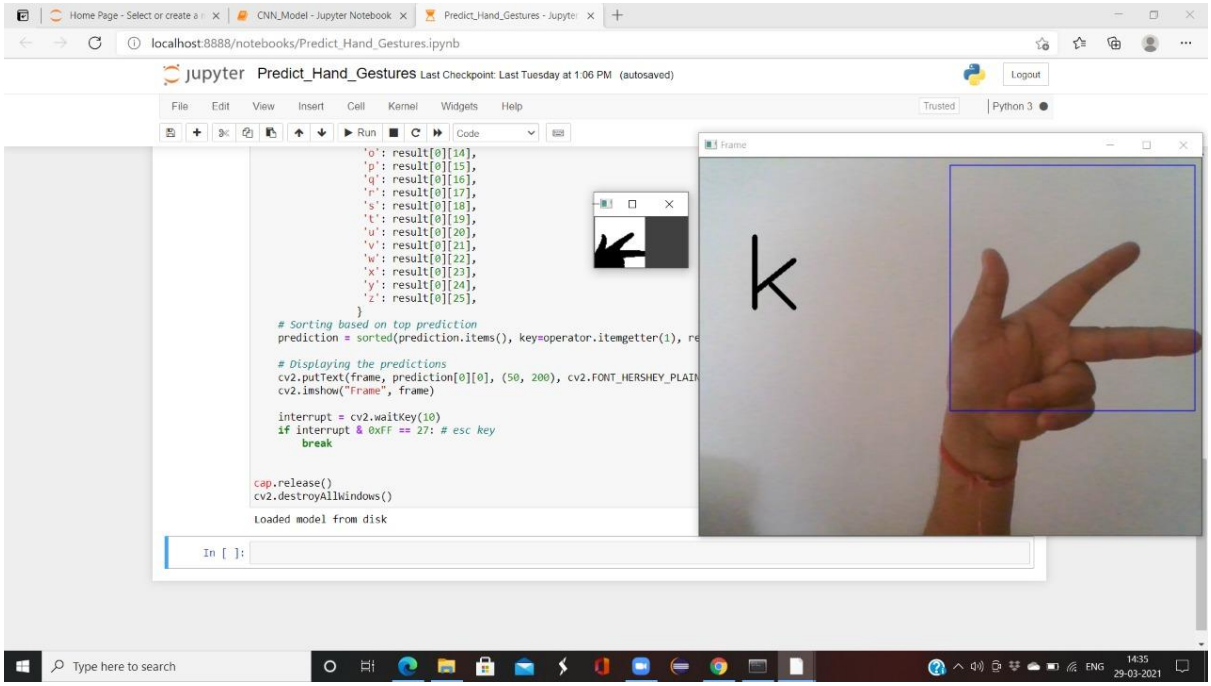
Chapter 6

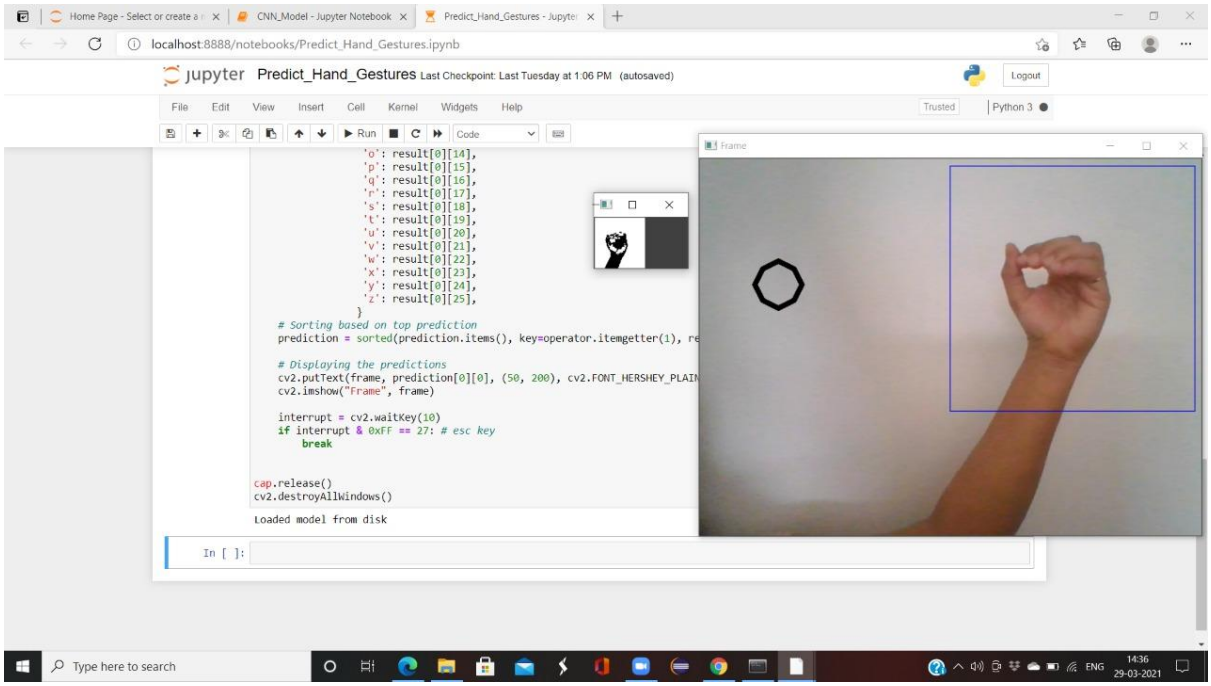
Output Screenshot






Home Page - Select or create a
CNN_Model - Jupyter Notebook
Predict_Hand_Gestures - Jupyter
localhost:8888/notebooks/Predict_Hand_Gestures.ipynb
jupyter Predict_Hand_Gestures Last Checkpoint: Last Tuesday at 1:06 PM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
'o': result[0][14],
'p': result[0][15],
'q': result[0][16],
'r': result[0][17],
's': result[0][18],
't': result[0][19],
'u': result[0][20],
'v': result[0][21],
'w': result[0][22],
'x': result[0][23],
'y': result[0][24],
'z': result[0][25],
}
# Sorting based on top prediction
prediction = sorted(prediction.items(), key=operator.itemgetter(1), re
# Displaying the predictions
cv2.putText(frame, prediction[0][0], (50, 200), cv2.FONT_HERSHEY_PLAIN
cv2.imshow("Frame", frame)
interrupt = cv2.waitKey(10)
if interrupt & 0xFF == 27: # esc key
break
cap.release()
cv2.destroyAllWindows()
Loaded model from disk
In [ ]:
Frame
a
Type here to search
ENG
14:34
29-03-2021


Home Page - Select or create a
CNN_Model - Jupyter Notebook
Predict_Hand_Gestures - Jupyter
localhost:8888/notebooks/Predict_Hand_Gestures.ipynb
jupyter Predict_Hand_Gestures Last Checkpoint: Last Tuesday at 1:06 PM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
'o': result[0][14],
'p': result[0][15],
'q': result[0][16],
'r': result[0][17],
's': result[0][18],
't': result[0][19],
'u': result[0][20],
'v': result[0][21],
'w': result[0][22],
'x': result[0][23],
'y': result[0][24],
'z': result[0][25],
}
# Sorting based on top prediction
prediction = sorted(prediction.items(), key=operator.itemgetter(1), re
# Displaying the predictions
cv2.putText(frame, prediction[0][0], (50, 200), cv2.FONT_HERSHEY_PLAIN
cv2.imshow("Frame", frame)
interrupt = cv2.waitKey(10)
if interrupt & 0xFF == 27: # esc key
break
cap.release()
cv2.destroyAllWindows()
Loaded model from disk
In [ ]:
Frame
k
Type here to search
ENG
14:35
29-03-2021


Home Page - Select or create a
CNN_Model - Jupyter Notebook
Predict_Hand_Gestures - Jupyter
localhost:8888/notebooks/Predict_Hand_Gestures.ipynb
jupyter Predict_Hand_Gestures Last Checkpoint: Last Tuesday at 1:06 PM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
'o': result[0][14],
'p': result[0][15],
'q': result[0][16],
'r': result[0][17],
's': result[0][18],
't': result[0][19],
'u': result[0][20],
'v': result[0][21],
'w': result[0][22],
'x': result[0][23],
'y': result[0][24],
'z': result[0][25],
}
# Sorting based on top prediction
prediction = sorted(prediction.items(), key=operator.itemgetter(1), re
# Displaying the predictions
cv2.putText(frame, prediction[0][0], (50, 200), cv2.FONT_HERSHEY_PLAIN
cv2.imshow("Frame", frame)
interrupt = cv2.waitKey(10)
if interrupt & 0xFF == 27: # esc key
break
cap.release()
cv2.destroyAllWindows()
Loaded model from disk
In [ ]:
Frame
Type here to search
ENG
14:36
29-03-2021

Chapter 7

Conclusion & Future Work

- ## Conclusion

System is based on the standard American Sign Language (ASL) and is used by the daily conversation. In this work, we have gone through an automatic sign language gesture recognition system in real-time, using different tools. Although our proposed work expected to recognized the sign language and convert it into the text, there's still a lot of scope for possible future work.

- ## Future Work

1.In Future, Our System will be able to convert text into speech. System can also get extension with sentence formation.

2. We are planning to achieve higher accuracy even in case of complex backgrounds by trying out various background subtraction algorithms. We are also thinking of improving the pre-processing to predict gestures in low light conditions with a higher accuracy.

3.We are also planning to add another module for converting sign to text/speech.