RAMNIRANJANJHUNJHUNWALA COLLEGE

Department of Computer Science

Ghatkopar(West),Mumbai-86

Project Report On

# HAND SIGN LANGUAGE RECOGNITION

Prepared By

Mr. Yash Kesharwani

Seat No-530

M.Sc.-II (Computer Science)

2022-2023

Project Guide

Mrs. Tejaswee Pol

RAMNIRANJAN JHUNJHUNWALA COLLEGE

Department Of Computer Science

Ghatkopar (West), Mumbai - 86

2022-2023

A project report on

# HAND SIGN LANGUAGE RECOGNITION

Towards partial fulfilment of the degree of M.Sc. (Computer Science) as prescribed by the University of Mumbai.

By

Mr. Yash Kesharwani

(Seat No-530)

Project Guide

Mrs. TEJASWEE POL

**RAMNIRANJAN JHUNJHUNWALA COLLEGE**

**Department Of Computer Science**

Ghatkopar (West), Mumbai – 86

2022-2023

# CERTIFICATE

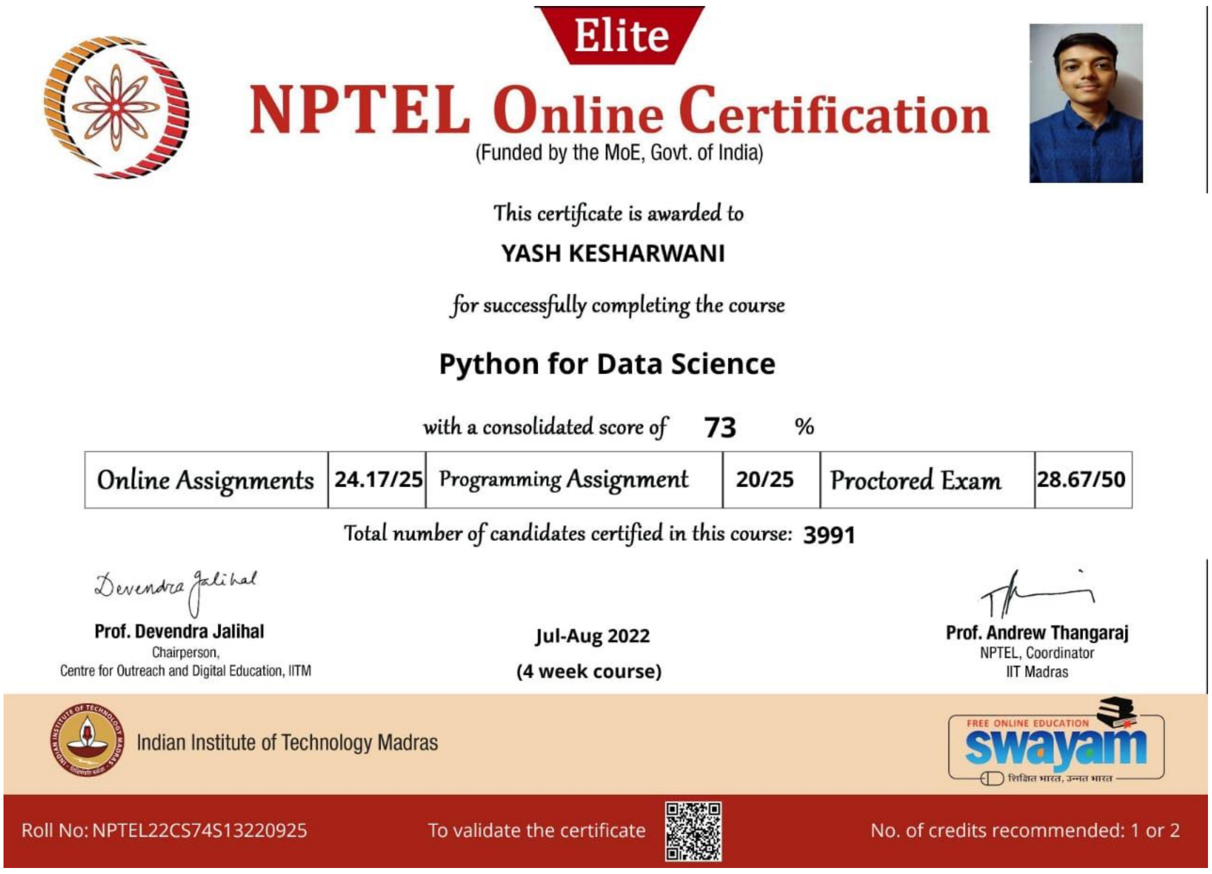
This is to certify that Mr./Miss. Yash Kesharwani(Seat No-530) has successfully completed the project titled,

“HAND SIGN LANGUAGE RECOGNITON“

Under the guidance towards the partial fulfilment of degree of Masters of Science
(Computer Science) submitted to Ramniranjan Jhunjhunwala College, Ghatkopar(W) during the academic year 2022-2023 as specified by the UNIVERSITY OF MUMBAI.

Guide
Mrs. Tejaswee Pol

Course Coordinator
Department Of Computer Science
Prof. Mrs. Vaidehi Deshpande



Elite

NPTEL Online Certification

(Funded by the MoE, Govt. of India)

*This certificate is awarded to*

**YASH KESHARWANI**

*for successfully completing the course*

**Python for Data Science**

*with a consolidated score of* **73** %

| | | | | | |
|---|---|---|---|---|---|
| Online Assignments | 24.17/25 | Programming Assignment | 20/25 | Proctored Exam | 28.67/50 |

*Total number of candidates certified in this course:* **3991**

Devendra Jalihal

**Prof. Devendra Jalihal**
Chairperson,
Centre for Outreach and Digital Education, IITM

**Jul-Aug 2022**
**(4 week course)**

[illegible]

**Prof. Andrew Thangaraj**
NPTEL, Coordinator
IIT Madras

Indian Institute of Technology Madras

FREE ONLINE EDUCATION swayam शिक्षित भारत, उन्नत भारत

Roll No: NPTEL22CS74S13220925 To validate the certificate No. of credits recommended: 1 or 2

## Acknowledgement

The development of this project is the result of the cumulative efforts along with the productive input of many of our teachers and friends at Ramniranjan Jhunjhunwala College, Ghatkopar.

It is my pleasure to express sincere thanks to Hon. Principal Dr. Himanshu Dawda sir who have always been the source of inspiration.

I would like to convey special thanks to Prof. Mrs. Vaidehi Deshpande, Head of Department (Computer Science) for her kind and helpful support regarding the project and the course.

I wish to thank to our project guide Prof. Mrs. Tejaswee Pol for her valuable suggestions regarding the selection of project and for frequently collaborating with this project. Her guidance not only help me in trying fruitful but also transform the whole process of learning and implementing into an enjoyable experience.

I am grateful and thankful to all teaching and non-teaching staff of computer science department who shared their years of experience, excellent support, and blossoms of suggestion with me for developing the project.

Finally heartfelt appreciation to those countless friends who have contributed greatly to the success of my project

**Yash Rajkishor Kesharwani**

# INDEX

# HAND SIGN LANGUAGE DETECTION

# 1: PROJECT INTRODUCTION

## 1.1 Introduction

The goal of this project was to build a neural network able to classify which letter of the American Sign Language (ASL) alphabet is being signed, given an image of a signing hand. This project is a first step towards building a possible sign language translator, which can take communications in sign language and translate them into written and oral language. Such a translator would greatly lower the barrier for many deaf and mute individuals to be able to better communicate with others in day to day interactions

.

This goal is further motivated by the isolation that is felt within the deaf community. Loneliness and depression exists in higher rates among the deaf population, especially when they are immersed in a hearing world. Large barriers that profoundly affect life quality stem from the communication disconnect between the deaf and the hearing. Some examples are information deprivation, limitation of social connections, and difficulty integrating in society .

Most research implementations for this task have used depth maps generated by depth camera and high resolution images. The objective of this project was to see if neural networks are able to classify signed ASL letters using simple images of hands taken with a personal device such as a laptop webcam. This is in alignment with the motivation as this would make a future implementation of a real time ASL-to-oral/written language translator practical in an everyday situation.

## 1.2 Objectives

- The objective of hand sign language detection is to enable communication between individuals who use sign language as their primary mode of communication and those who
- do not understand it.

- Human interpreters may make errors or have limitations in interpreting complex or nuanced signs, but technology can provide a more consistent and reliable interpretation.

- This can help to create a more inclusive and welcoming environment for people who are deaf or hard of hearing.

- Hand sign language detection can also facilitate the development of new technologies that enable better communication and access for people who use sign language. For example, sign language recognition technology can be integrated into devices such as smartphones or smart speakers to enable real-time translation of sign language into spoken language or text.

# 2: PROJECT BACKGROUND STUDY

## 2.1 Background

Hand sign language detection is a technology that aims to enable computers to recognize and interpret human hand gestures, which can be used to communicate with people who use sign language as their primary means of communication. It is part of a larger field of research known as computer vision, which involves teaching computers to interpret and understand visual information. Hand sign language detection technology has a wide range of potential applications, from improving communication for the deaf and hard of hearing to enhancing human-robot interactions and improving the usability of virtual and augmented reality interfaces. It can also be used to develop new ways of teaching sign language to people who are hearing, by providing real-time feedback on their hand gestures and encouraging them to improve their accuracy and fluency.

### 2.1.2 Problem Formulation

Overall, the problem formulation for a hand sign language detection project involves collecting and preprocessing data, extracting relevant features, selecting and training a suitable machine learning model, and deploying the system in a real-world application. The goal is to create an accurate and reliable system that can improve communication and accessibility for people who are deaf or hard of hearing.

### 2.1.3 Existing System

Leap Motion uses infrared cameras to detect hand gestures.

Myo Armband uses EMG sensors to control devices with hand gestures.

OpenPose is an open-source software library for human pose estimation, SignAll is a sign language recognition system, and Hand Talk is a mobile app that interprets Brazilian Sign Language gestures into Portuguese text and speech.

Overall, these existing systems demonstrate the potential of hand sign language detection technology for improving accessibility and communication for people who are deaf or hard of hearing. However, there is still room for improvement in terms of accuracy, real-time performance, and ease of use, and ongoing research and development in this area is important to address these challenges.

### 2.1.4 Proposed System

The proposed system should collect a large dataset of hand gesture samples from different signers and under various conditions, and preprocess the data to remove noise and distortions and extract relevant features. he proposed system should be able to recognize hand gestures in real-time, using a camera or sensor to track the position, orientation, and motion of the hand. It should also detect different hand shapes and movements that correspond to specific sign language gestures.

### 2.1.5 Unique Features of the System

Multi-modal sensor fusion provides more accurate and robust hand gesture recognition in challenging environments.

Personalized adaptation improves accuracy and reduces errors caused by individual variations in hand shapes.

Continuous learning improves system performance and keeps up-to-date with new sign language developments.

Interactive feedback can help users improve their signing accuracy and consistency, as well as their engagement and motivation in using the system.

Cloud-based processing can offload the computational load from the local device, providing more powerful and scalable processing capabilities, real-time collaboration and communication, and access to additional language translation and other resources.

### 2.1.6 Data Sources

Public Sign Language Datasets: There are several public datasets available for sign language recognition research, such as the American Sign Language (ASL) dataset, the Chinese Sign Language (CSL) dataset, and the German Sign Language (DGS) dataset. These datasets contain a large number of video clips or image sequences of sign language gestures, along with their corresponding annotations or labels.

Custom Collected Datasets: A custom dataset can be collected specifically for the hand sign language detection project, using a camera or sensor to capture hand gestures from different signers and under different environmental conditions. This dataset can include a wide variety of sign language gestures, and can be annotated or labeled by expert sign language interpreters.

Motion Capture Data: Motion capture technology can be used to collect precise and detailed data on hand movements and gestures, which can be used to train and evaluate hand sign language detection models. This data can be collected using specialized motion capture systems or wearable devices, and can be processed and annotated using motion capture software.

### 2.1.7 Read & Preprocess the Dataset

The dataset must be read and preprocessed before training a hand sign language detection model.

The dataset needs to be read and loaded into the system's memory, which can include image sequences, video clips, or other formats.

Data cleaning is necessary to remove outliers and noise that can affect the performance of a model, such as removing frames or sequences with low-quality or inconsistent data or signers with poor signing proficiency.

Data augmentation can be used to increase the diversity and size of a dataset by applying random transformations to the data, such as rotation, translation, or scaling.

Data Labeling is necessary to indicate the sign language gesture being performed in each frame or sequence, either manually by expert interpreters or using automated methods such as motion tracking or hand shape classification.

Feature extraction is the process of extracting relevant features from data to train a hand sign language detection model. It can involve extracting hand shape features or motion features, such as velocity or acceleration.

### 2.1.8 Extract, Resize & Normalize Frames

Extracting, resizing, and normalizing frames is an important step in preparing the dataset for training a hand sign language detection model. For video datasets, the frames need to be extracted from the videos, resized to a fixed size, normalized to a common scale, and data augmentation applied to the frames to increase the diversity and size of the dataset. Finally, preprocessed frames need to be saved in a format that can be used for training the hand sign language detection model. Careful attention should be given to these steps when preparing the dataset for training a hand sign language detection model.

### 2.1.9 Dataset Creation

Data normalization is necessary to ensure the model can handle different lighting and environmental conditions, such as adjusting brightness and contrast or using statistical techniques.

Collecting, labeling, and preprocessing data for hand sign language detection.

Data collection involves recording videos, images, or motion capture systems to capture hand sign language gestures.

Data labeling is necessary to indicate sign language gestures in each frame or sequence, either manually or using automated methods.

Data must be preprocessed to remove outliers and noise that can affect the performance of the model, such as removing frames or sequences with low-quality or inconsistent data.

Data augmentation can increase the diversity and size of a dataset by applying random transformations to the data.

Feature extraction is the process of extracting hand shape and motion features from data to train a hand sign language detection model.

## 2.2 Deep Learning and CNN Model

### 2.2.1 Deep Learning

Deep learning is an unofficial name for a group of machine learning algorithms, usually consisted of multi-layer neural networks. Yan Lecun in 1988 introduced backpropagation in multi-layer convolutional neural networks and made it possible to create deep structures. However, mostly because of the process limitations, they did not become a mainstream tool in machine learning until recently. I have used a deep neural network to create a probability distribution for different emotions given every segment. Deep neural networks can be defined as any network with more than one hidden layer. With this definition, they cover all the deep structures such as convolutional, long short-term memory, and generative adversarial networks. However, in this text, we have Sensors 2021, 21, 1249 11 of 27 separated deep artificial neural networks in the sense of only using linear, fully connected layers from other deep networks incorporating any additional nonlinear layer. Thus, in the sense of our definition, deep artificial neural networks have better capabilities in modeling problems than their single-layer siblings. However, their modeling is still limited to nonlinear polynomial functions, and the cost of such ability is the exponential increase in the number of the variables to train; hence more process and more memory.

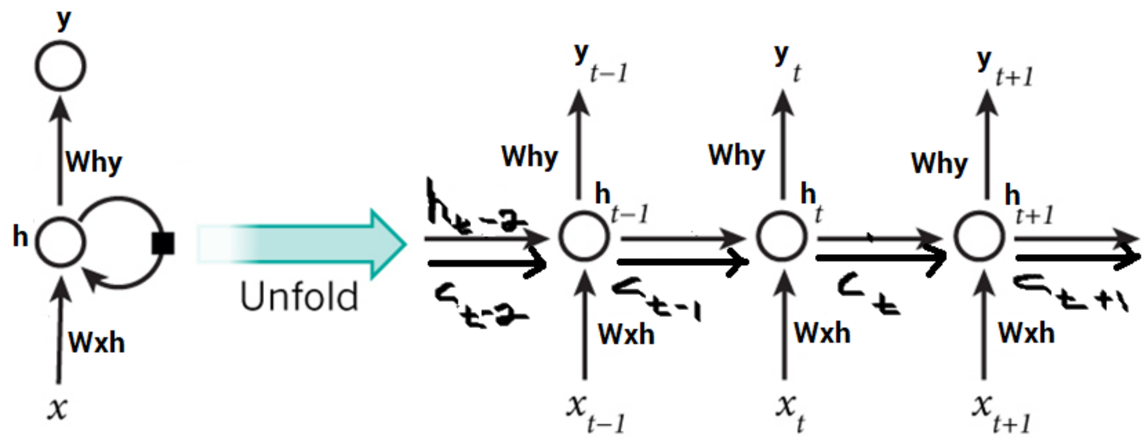
### 2.2.2 LSTM (Long Short-Term Memory)

An LSTM module has a cell state and three gates which provides them with the power to selectively learn, unlearn or retain information from each of the units. The cell state in LSTM helps the information to flow through the units without being altered by allowing only a few linear interactions. Each unit has an input, output and a forget gate which can add or remove the information to the cell state. The forget gate decides which information from the previous cell state should be forgotten for which it uses a sigmoid function. The input gate controls the information flow to the current cell state using a point-wise multiplication operation of ‘sigmoid’ and ‘tanh’ respectively. Finally, the output gate decides which information should be passed on to the next hidden state

## LSTM Model Architectures

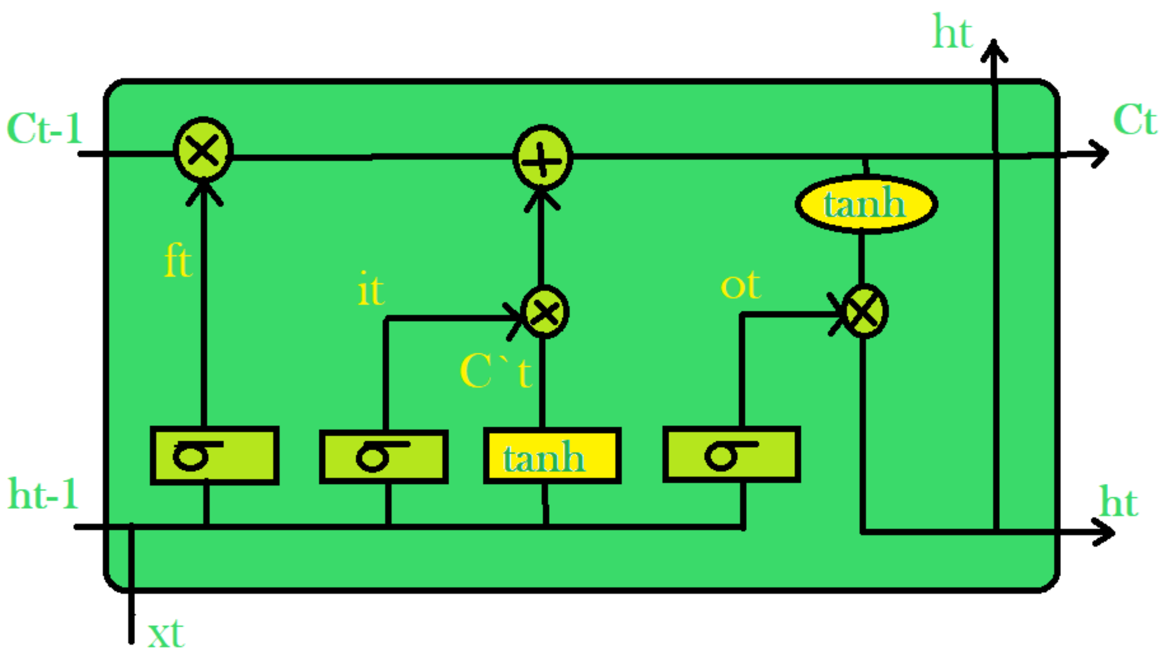
The basic difference between the architectures of RNNs and LSTMs is that the hidden layer of LSTM is a gated unit or gated cell. It consists of four layers that interact with one another in a way to produce the output of that cell along with the cell state. These two things are then passed onto the next hidden layer. Unlike RNNs which have got the only single neural net layer of tanh, LSTMs comprises of three logistic sigmoid gates and one tanh layer. Gates have been introduced in order to limit the information that is passed through the cell. They determine which part of the information will be needed by the next cell and which part is to be discarded. The output is usually in the range of 0-1 where ‘0’ means ‘reject all’ and ‘1’ means ‘include all’.

## Hidden layers of LSTM :



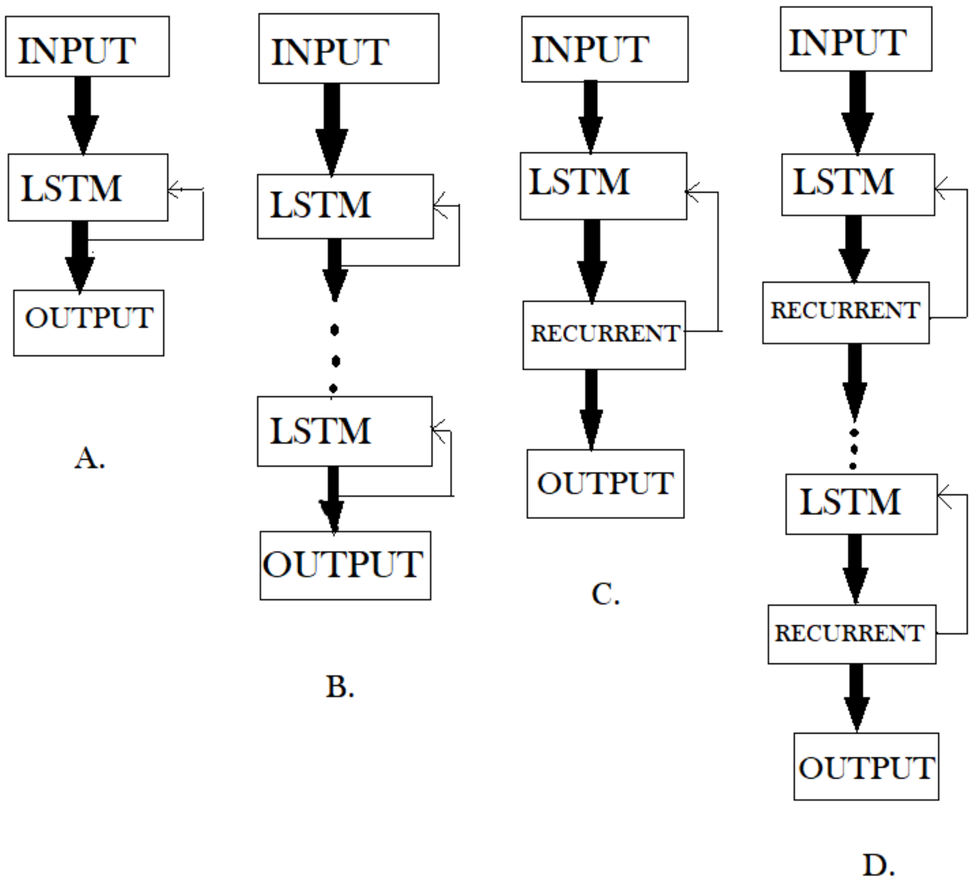
Each LSTM cell has three inputs h_{t-1} ,C_{t-1} and x_t and two outputs h_t and C_t . For a given time t, h_t is the hidden state, C_t is the cell state or memory, x_t is the current data point or input. The first sigmoid layer has two inputs–h_{t-1} and x_t where h_{t-1} is the hidden state of the previous cell. It is known as the forget gate as its output selects the amount of information of the previous cell to be included. The output is a number in [0,1] which is multiplied (point-wise) with the previous cell state C_{t-1} .

**Conventional LSTM:**



The second sigmoid layer is the input gate that decides what new information is to be added to the cell. It takes two inputs h_{t-1} and x_t . The tanh layer creates a vector C_t of the new candidate values. Together, these two layers determine the information to be stored in the cell state. Their point-wise multiplication ( i_t ? C_t ) tells us the amount of information to be added to the cell state. The result is then added with the result of the forget gate multiplied with previous cell state ( f_t* C_{t-1} ) to produce the current cell state C_t . Next, the output of the cell is calculated using a sigmoid and a tanh layer. The sigmoid layer decides which part of the cell state will be present in the output whereas tanh layer shifts the output in the range of [-1,1]. The results of the two layers undergo point-wise multiplication to produce the output ht of the cell.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_6 (LSTM)                (None, 30, 64)            442112
_________________________________________________________________
lstm_7 (LSTM)                (None, 30, 128)           98816
_________________________________________________________________
lstm_8 (LSTM)                (None, 64)                49408
_________________________________________________________________
dense_6 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_7 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_8 (Dense)              (None, 3)                 99
=================================================================
Total params: 596,675
Trainable params: 596,675
Non-trainable params: 0
_________________________________________________________________
```



## 2.3 Libraries

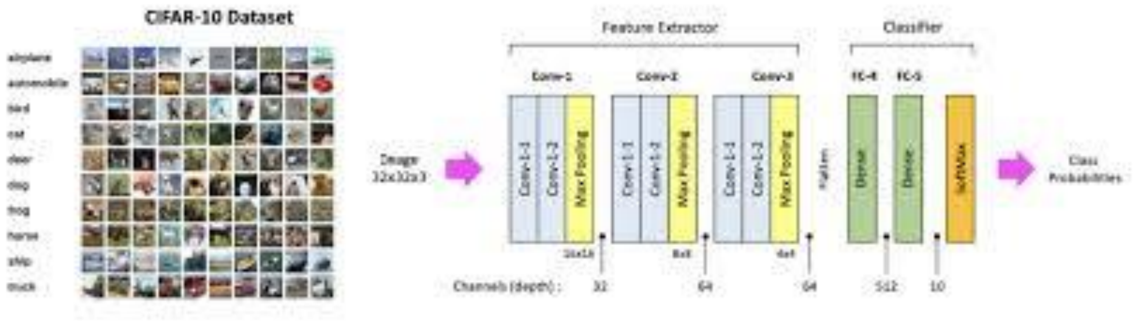
### 2.3.1 TensorFlow

TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow. The focus of Keras is the idea of a model. The main type of model is called a Sequence which is a linear stack of layers. You create a sequence and add layers to it in the order that you wish for the computation to be performed. Once defined, you compile the model which makes use of the underlying framework to optimize the computation to be performed by your model. In this you can specify the loss function and the optimizer to be used. Once compiled, the model must be fit to data. This can be done one batch of data at a time or by firing off the entire model training regime. This is where all the compute happens. Once trained, you can use your model to make predictions on new data.

Let us first try to understand what the word TensorFlow mean! TensorFlow is basically a software library for numerical computation using data flow graphs where:

- nodes in the graph represent mathematical operations.
- edges in the graph represent the multidimensional data arrays (called tensors) communicated between them. (Please note that tensor is the central unit of data in TensorFlow).

Consider the diagram given below



### 2.3.2 Keras

Keras is a minimalist Python library for deep learning that can run on top of TensorFlow. It was developed to make implementing deep learning models as fast and easy as possible for research and development. It runs on Python 2.7 or 3.5 and can seamlessly execute on GPUs and CPUs given the underlying frameworks. It is released under the permissive MIT license.

Keras was developed and maintained by using four guiding principles:

1) **Modularity**: A model can be understood as a sequence or a graph alone. All the concerns of a deep learning model are discrete components that can be combined in arbitrary ways.
2) **Minimalism**: The library provides just enough to achieve an outcome, no frills and maximizing readability.
3) **Extensibility**: New components are intentionally easy to add and use within the framework, intended for researchers to trial, and explore new ideas.
4) **Python**: No separate model files with custom file formats. Everything is native Python.

### How to Install Keras

Keras is relatively straightforward to install if you already have a working Python and SciPy environment. You must also have an installation of TensorFlow on your system already. sudo pip install keras

We can summarize the construction of deep learning models in Keras as follows:

**Define your model**. Create a sequence and add layers.

**Compile your model**. Specify loss functions and optimizers.

**Fit your model**. Execute the model using data.

**Make predictions**. Use the model to generate predictions on new data

### 2.3.3 OpenCV

OpenCV (Open-Source Computer Vision Library: http://opencv.org) is an opensource library that includes several hundreds of computer vision algorithms. OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

**Core functionality (core)** - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.

**Image Processing (imgproc)** - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.

**Video Analysis (video)** - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.

**Camera Calibration and 3D Reconstruction (calib3d)** - basic multipleview geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.

**2D Features Framework (features2d)** - salient feature detectors, descriptors, and descriptor matchers.

**Object Detection (objdetect)** - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).

**High-level GUI (highgui)** - an easy-to-use interface to simple UI capabilities.

**Video I/O (videoio)** - an easy-to-use interface to video capturing and video codec

### 2.3.4 NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

### 2.3.5 Jupyter Notebook

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It is used for data science, scientific computing, and machine learning projects.

The Jupyter Notebook interface is based on a web browser and provides an interactive environment for running code in multiple languages, including Python, R, and Julia. Each notebook consists of a series of cells that can contain code, text, or multimedia elements such as images and videos. You can execute code in the cells, visualize data using charts and graphs, and include explanatory text to document your work.

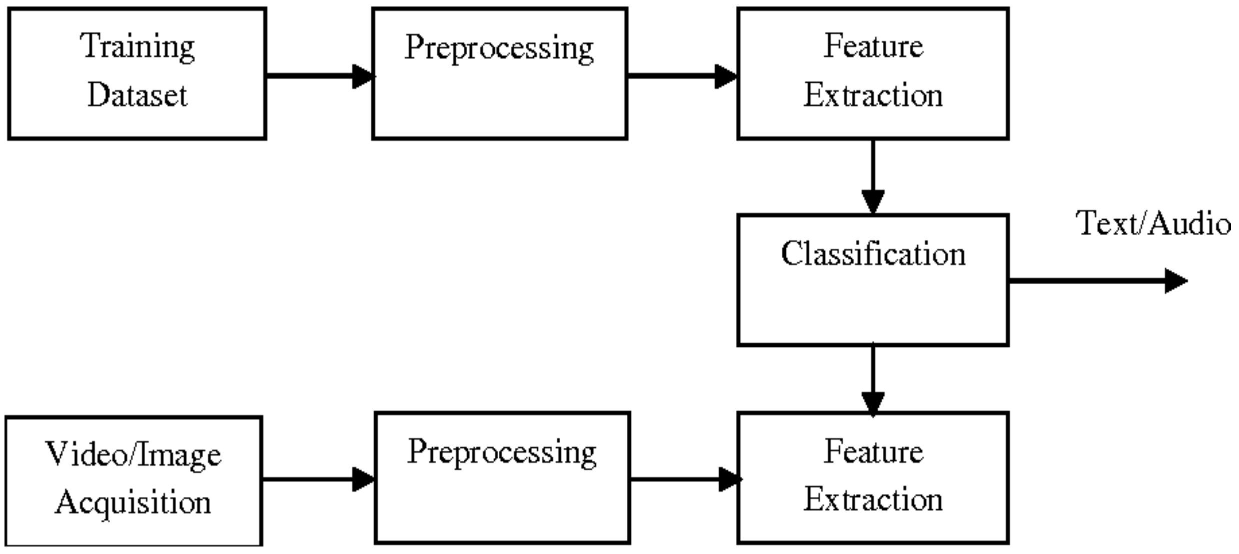
One of the key benefits of Jupyter Notebook is that it allows you to create reproducible research by saving your code, output, and explanations in a single document. This makes it easier to share your work with colleagues, collaborators, and the wider research community.

Jupyter Notebook also supports a range of popular libraries and frameworks for data science and machine learning, such as NumPy, Pandas, Matplotlib, and Scikit-learn. You can install these libraries directly into your Jupyter Notebook environment and use them to analyze and visualize data, build machine learning models, and deploy your applications.

Overall, Jupyter Notebook is a powerful and flexible tool for data science and machine learning projects that allows you to create interactive, reproducible, and shareable documents that combine code, data, and explanations in a single notebook.

# 3: BLOCK DIAGRAM STUDY

## 3.1 BLOCK DRIAGRAM

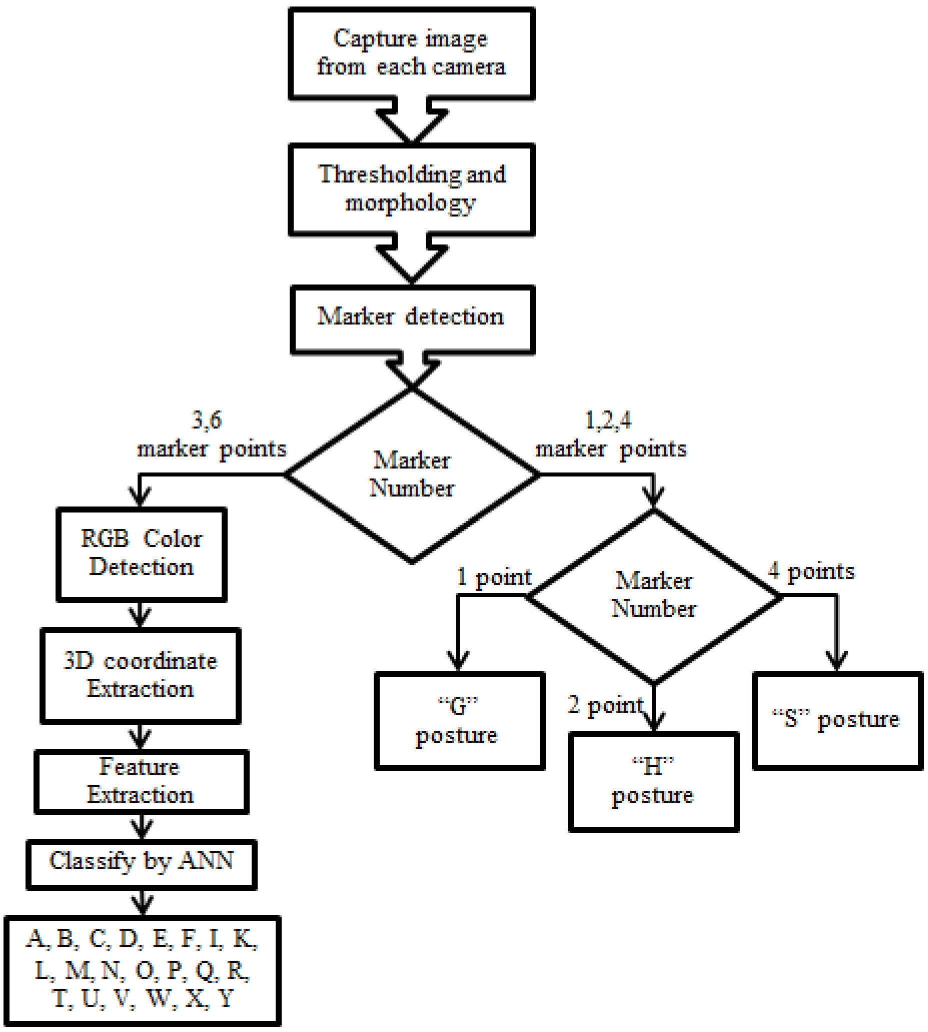


## 3.2 Block Diagram Description

- We are taking the video frames, extracting the features from the video meaning what factor or the component of the video effect the hand sign detection.
- So feature enhancement means doing some preprocessing operations which can help us in detecting the better gesture.
- So for the classifier training we are using the neural network like Convolutional Neural Networks (LSTM) and Deep Learning Method to classify the hand gesture and then the activity recognition and display the activity of a person.
- So for the classifier training we are using the neural network like Convolutional Neural Networks (CNNs)and Deep Learning Method to classify the activity and then the activity recognition and display the activity of the actor.

# 4: FLOW CHART

## 4.1 FLOW CHART DIAGRAM.



## 4.2 FLOW CHART DESCRIPRTION

Multi-layer system architecture using LSTM for sign recognition in video frames.

- The feature vectors are extracted using InceptionResNetV2 and passed to the model. Here, the video frames are classified into objects with InceptionResNet-2; then, the task is to create key points stacked for video frames

- The first layer of the neural network is composed of a combination of LSTM. This composition can be used to capture the semantic dependencies in a more effective way;

- The dropout is used to reduce overfitting and improve the model's generalization ability;

- The final output is obtained through the 'softmax' function.

The LSTM methods are used because of their ability to remember previous inputs with the help of gates. The corresponding word is learned on the basis of the previous activation and current input feature, and the weights are adjusted accordingly. The training dataset is further split into 80:20 ratios, sorting the data into training and validation datasets. The validation dataset can be useful to test the learning achieved by the model after each epoch. By proceeding in this way, the training process may be monitored and validated, and it is possible to check whether the training was completed correctly. K-fold cross-validation is used with ten folds to confirm that the model is performing accurately on the dataset. Each model was trained with the help of 10 different combinations of the training set and cross-validation set.

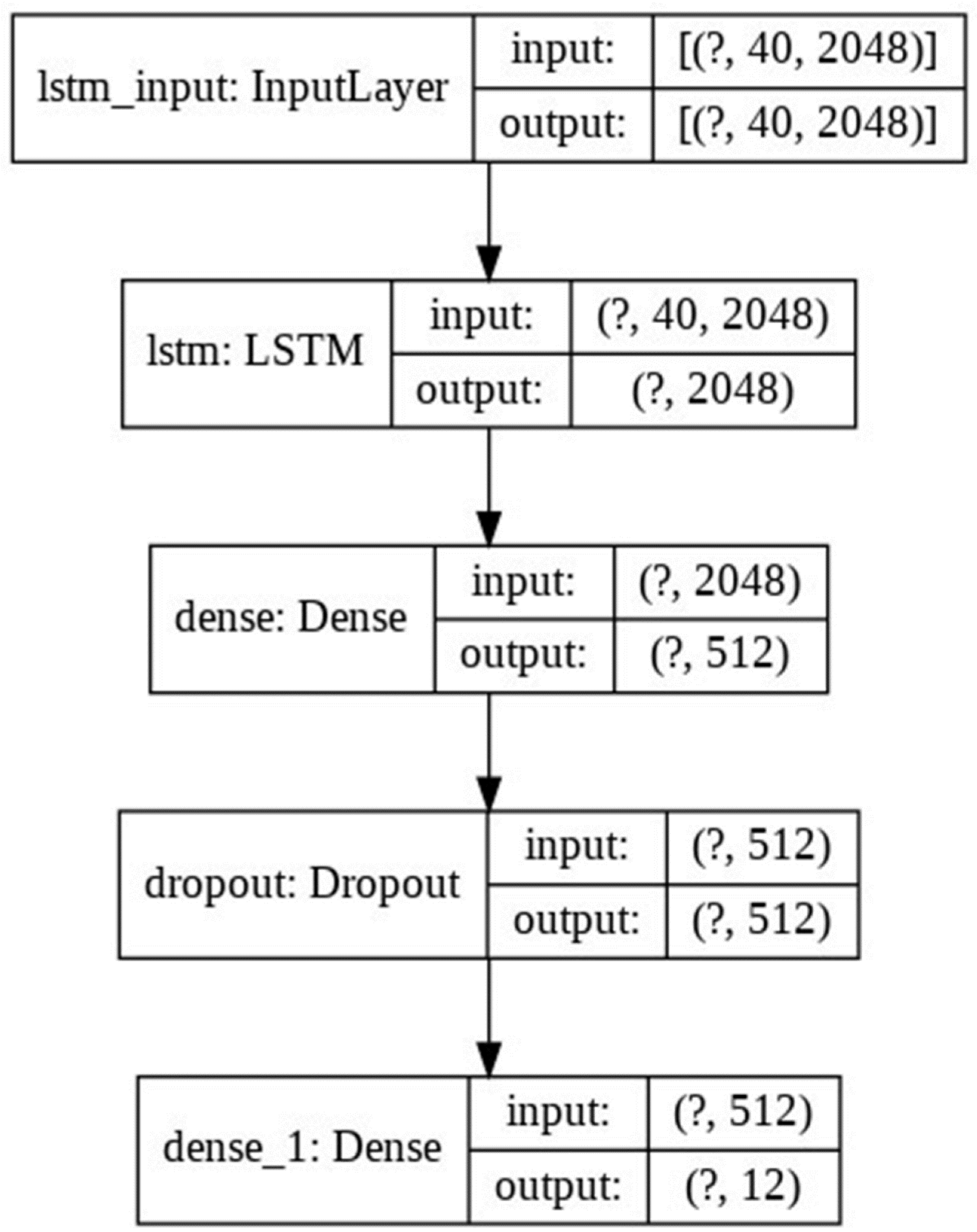
Hochreiter and Schmidhuber proposed long short-term memory (a development of RNN) to overcome the previously described shortcomings of RNNs by adding more interactions per module (or cell). LSTM is a type of RNN that can learn new things with long-term dependencies and can memorize information for extended periods. The LSTM model is organized in a chain structure. Nonetheless, the structure of a repeated module may be different for different applications. Rather than using a single neural network, a one-of-a-kind communication system with four interconnected layers is proposed by Le Xuan . The LSTM's structure is as follows: $h_{t-1}$ is the previous output block; $C_{t-1}$ is a memory block of the previous block; $x_t$ is an input vector; $c_t$ is a memory block from the current block; $h_t$ is the output block of the current block.

Similar to LSTM, GRUs are used for enhancement in the hidden cells. GRUs were developed to overcome the vanishing gradient problem by retaining prior memory to aid future decision-making. The GRU was created in response to a query (whether all of the components found in the LSTM are required for managing amnesia and the unit time scale) [21,22,23,24].

The GRU has gating units that modulate the flow of information inside the unit without having separate memory cells. Activation in GRU can be represented as hjt,

(2)

covering the detailed structure of LSTM along with other layers, such as dense layers and dropout layers. After extracting the requisite features, the values are passed to the input layer with a shape of (40, 2048). The next steps are followed



# 5 : SYSTEM REQUIREMENTS

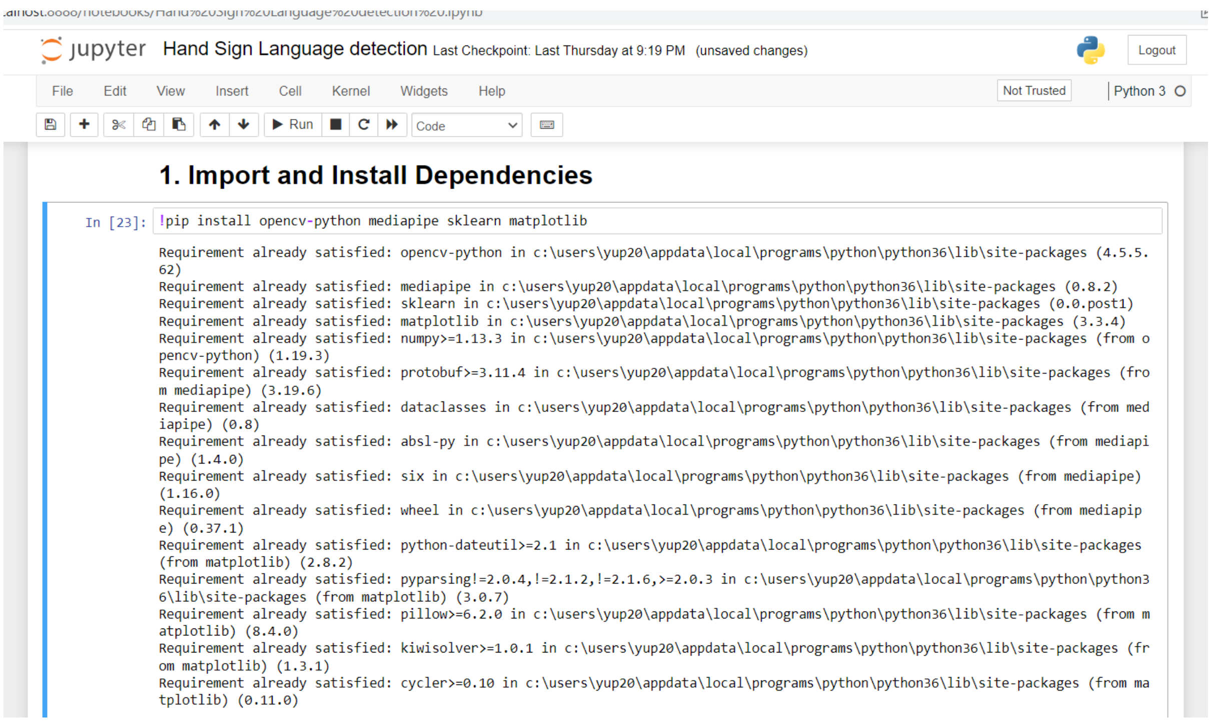
## 5.1 Hardware Requirements :

- The following are the minimum hardware requirements needed:
- Windows/Linux/Mac OS Machine
- GPU
- RAM 8GB or more.
- System type: 64-bit OS, x64-based processor

## 5.2 Software Requirements :

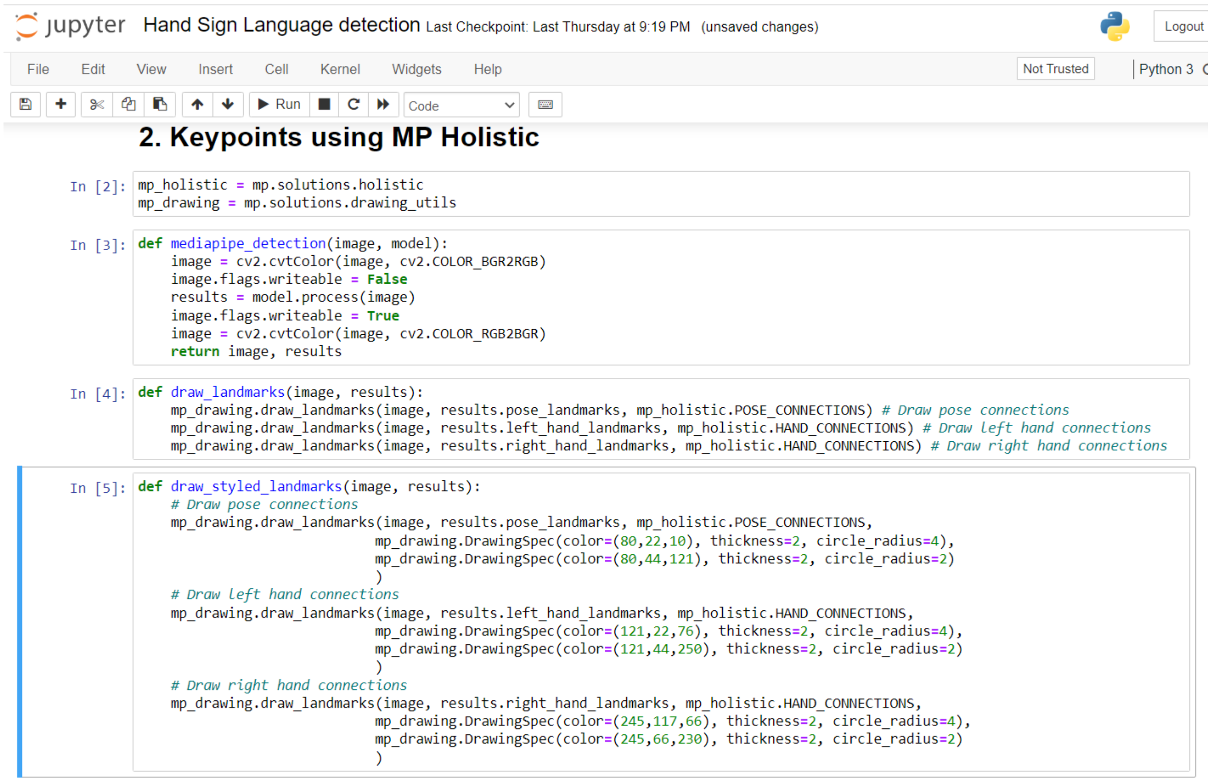
- The following are the minimum software requirements needed:
- Python 3
- TensorFlow and Keras library
- Packages numpy,pandas,matplotlib, sklearn
- Jupyter Notebook

# 6 : SCREEN SHOTS


alhost:8888/notebooks/Hand%20Sign%20Language%20detection%20.ipynb
jupyter
Hand Sign Language detection Last Checkpoint: Last Thursday at 9:19 PM (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Not Trusted
Python 3
Run
Code

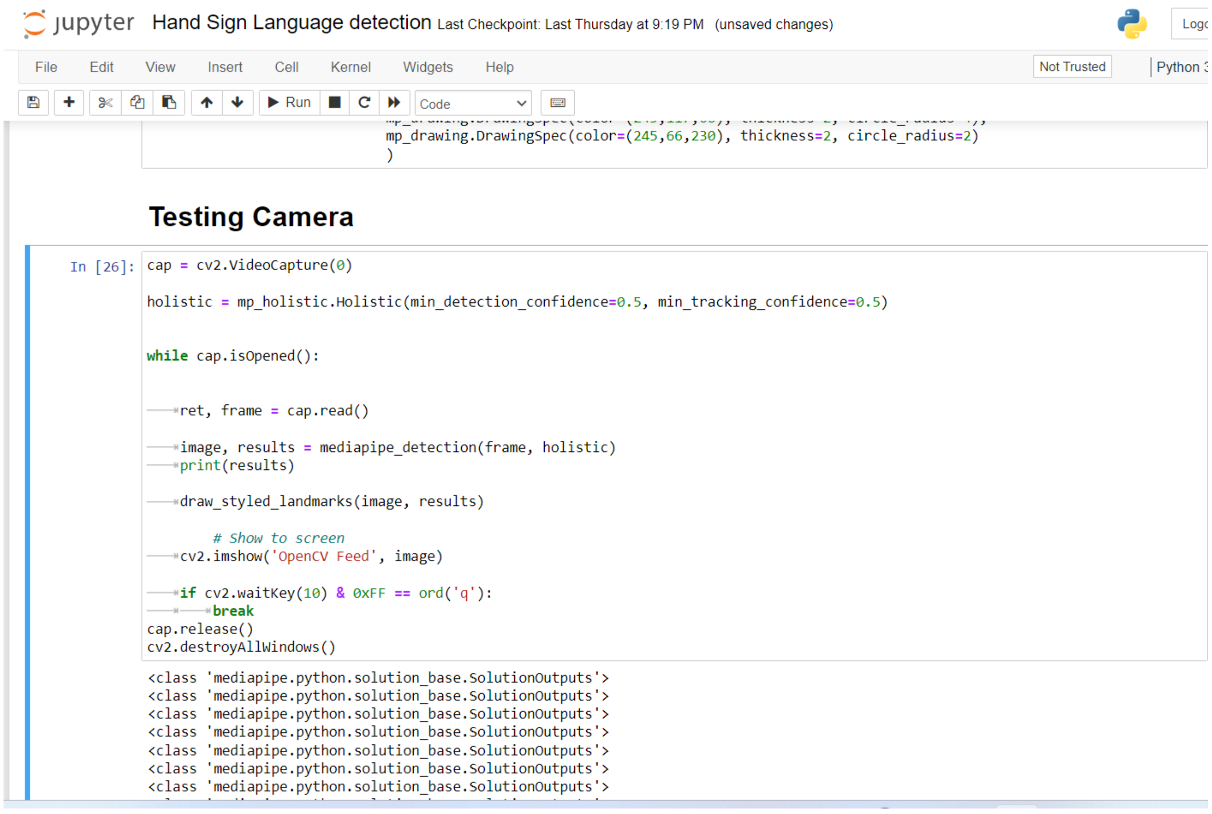
## 1. Import and Install Dependencies

```
In [23]: !pip install opencv-python mediapipe sklearn matplotlib

Requirement already satisfied: opencv-python in c:\users\yup20\appdata\local\programs\python\python36\lib\site-packages (4.5.5.
62)
Requirement already satisfied: mediapipe in c:\users\yup20\appdata\local\programs\python\python36\lib\site-packages (0.8.2)
Requirement already satisfied: sklearn in c:\users\yup20\appdata\local\programs\python\python36\lib\site-packages (0.0.post1)
Requirement already satisfied: matplotlib in c:\users\yup20\appdata\local\programs\python\python36\lib\site-packages (3.3.4)
Requirement already satisfied: numpy>=1.13.3 in c:\users\yup20\appdata\local\programs\python\python36\lib\site-packages (from o
pencv-python) (1.19.3)
Requirement already satisfied: protobuf>=3.11.4 in c:\users\yup20\appdata\local\programs\python\python36\lib\site-packages (fro
m mediapipe) (3.19.6)
Requirement already satisfied: dataclasses in c:\users\yup20\appdata\local\programs\python\python36\lib\site-packages (from med
iapipe) (0.8)
Requirement already satisfied: absl-py in c:\users\yup20\appdata\local\programs\python\python36\lib\site-packages (from mediapi
pe) (1.4.0)
Requirement already satisfied: six in c:\users\yup20\appdata\local\programs\python\python36\lib\site-packages (from mediapipe)
(1.16.0)
Requirement already satisfied: wheel in c:\users\yup20\appdata\local\programs\python\python36\lib\site-packages (from mediapip
e) (0.37.1)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\yup20\appdata\local\programs\python\python36\lib\site-packages
(from matplotlib) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users\yup20\appdata\local\programs\python\python3
6\lib\site-packages (from matplotlib) (3.0.7)
Requirement already satisfied: pillow>=6.2.0 in c:\users\yup20\appdata\local\programs\python\python36\lib\site-packages (from m
atplotlib) (8.4.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\yup20\appdata\local\programs\python\python36\lib\site-packages (fr
om matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\yup20\appdata\local\programs\python\python36\lib\site-packages (from ma
tplotlib) (0.11.0)
```


jupyter
Hand Sign Language detection Last Checkpoint: Last Thursday at 9:19 PM (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Not Trusted
Python 3
Run
Code

## 2. Keypoints using MP Holistic

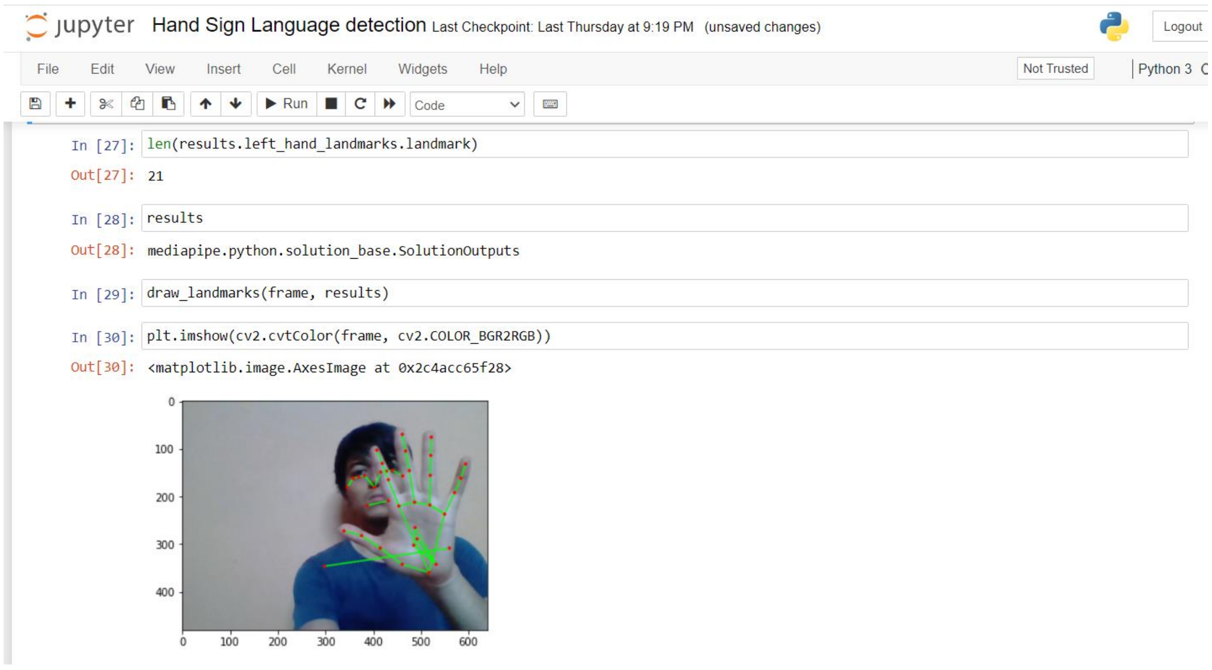
```
In [2]: mp_holistic = mp.solutions.holistic
        mp_drawing = mp.solutions.drawing_utils

In [3]: def mediapipe_detection(image, model):
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            image.flags.writeable = False
            results = model.process(image)
            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
            return image, results

In [4]: def draw_landmarks(image, results):
            mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS) # Draw pose connections
            mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS) # Draw left hand connections
            mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS) # Draw right hand connections

In [5]: def draw_styled_landmarks(image, results):
            # Draw pose connections
            mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                                     mp_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
                                     mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
                                     )
            # Draw left hand connections
            mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                     mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                                     mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                                     )
            # Draw right hand connections
            mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                     mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                                     mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                                     )
```


jupyter Hand Sign Language detection Last Checkpoint: Last Thursday at 9:19 PM (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Run
Code

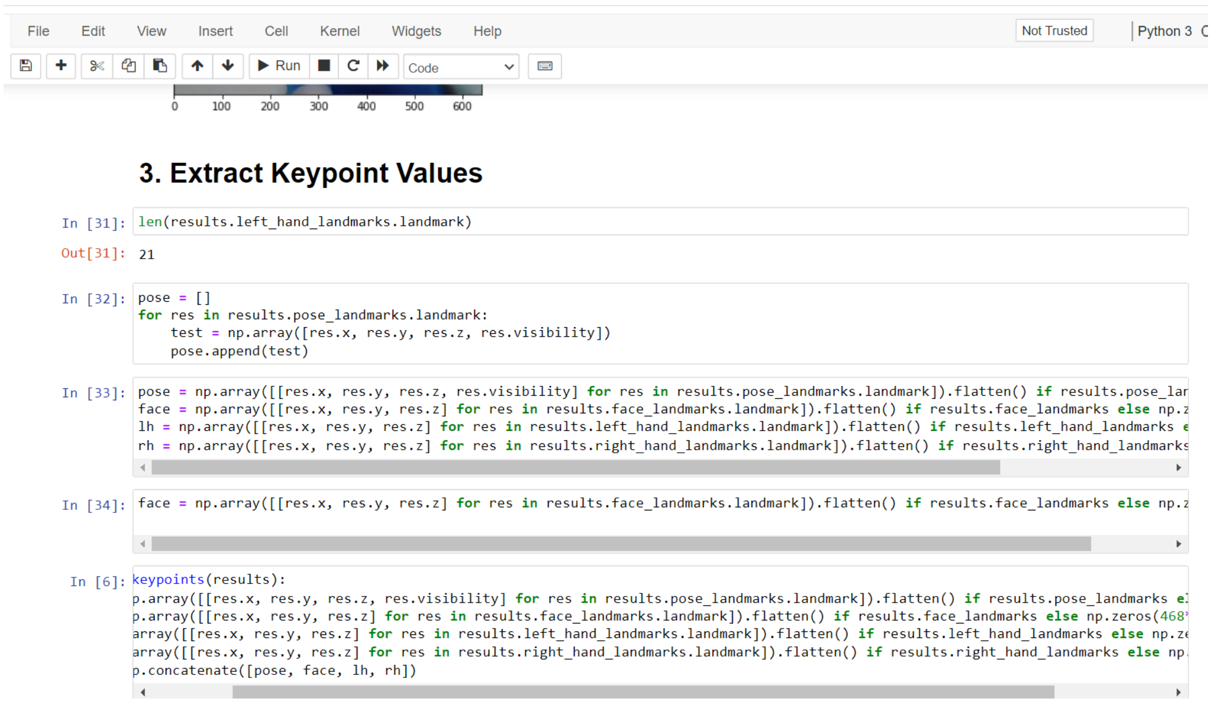
```
                         mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                         )
```

## Testing Camera

```
In [26]: cap = cv2.VideoCapture(0)

         holistic = mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5)


         while cap.isOpened():


             ret, frame = cap.read()

             image, results = mediapipe_detection(frame, holistic)
             print(results)

             draw_styled_landmarks(image, results)

                 # Show to screen
             cv2.imshow('OpenCV Feed', image)

             if cv2.waitKey(10) & 0xFF == ord('q'):
                 break
         cap.release()
         cv2.destroyAllWindows()

         <class 'mediapipe.python.solution_base.SolutionOutputs'>
         <class 'mediapipe.python.solution_base.SolutionOutputs'>
         <class 'mediapipe.python.solution_base.SolutionOutputs'>
         <class 'mediapipe.python.solution_base.SolutionOutputs'>
         <class 'mediapipe.python.solution_base.SolutionOutputs'>
         <class 'mediapipe.python.solution_base.SolutionOutputs'>
         <class 'mediapipe.python.solution_base.SolutionOutputs'>
```

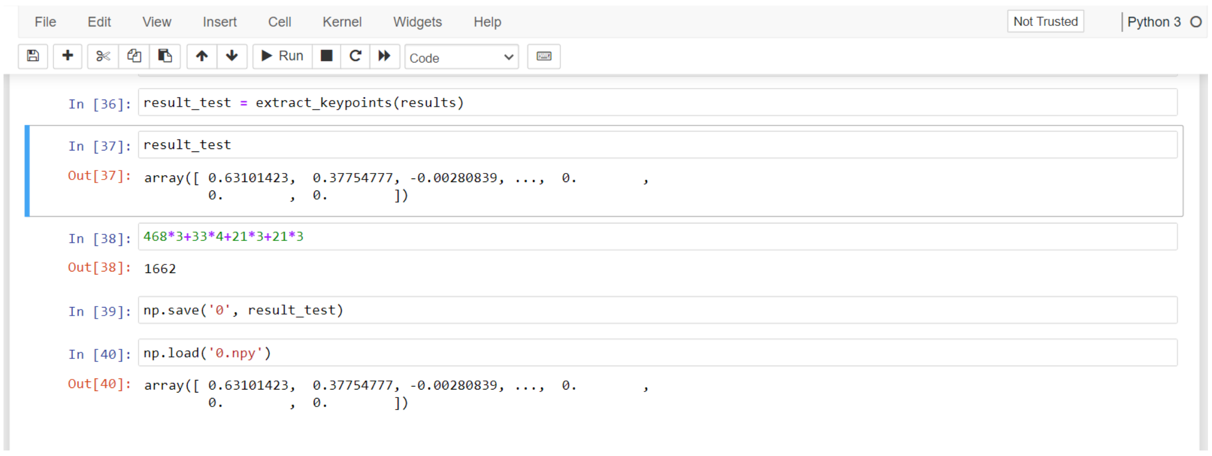

jupyter Hand Sign Language detection Last Checkpoint: Last Thursday at 9:19 PM (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Code
0
100
200
300
400
0
100
200
300
400
500
600

```
In [27]: len(results.left_hand_landmarks.landmark)

Out[27]: 21

In [28]: results

Out[28]: mediapipe.python.solution_base.SolutionOutputs

In [29]: draw_landmarks(frame, results)

In [30]: plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))

Out[30]: <matplotlib.image.AxesImage at 0x2c4acc65f28>
```

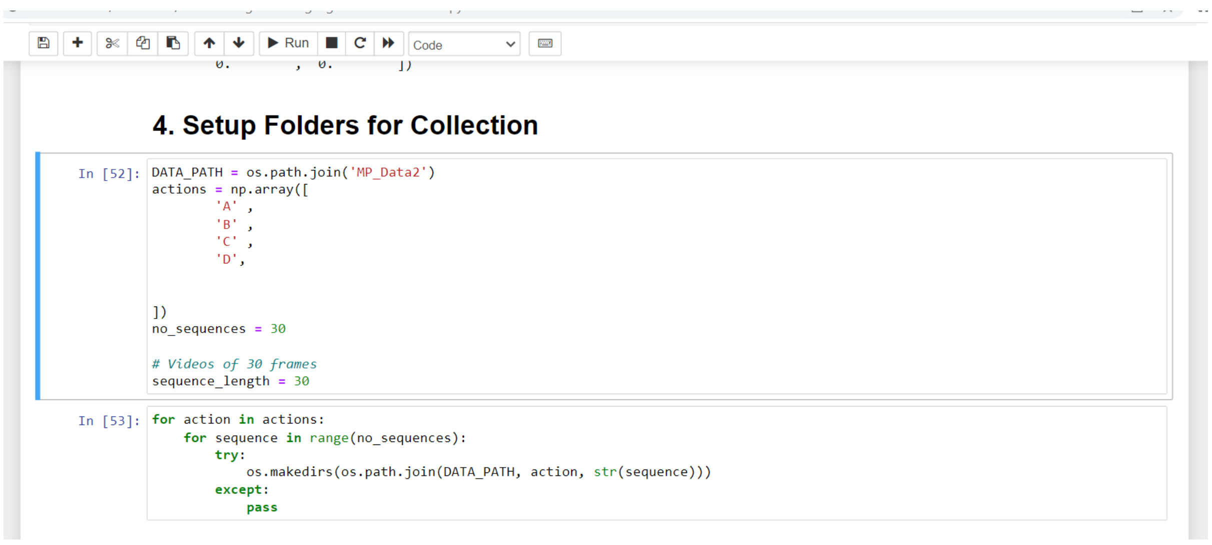


## 3. Extract Keypoint Values

```
In [31]: len(results.left_hand_landmarks.landmark)
Out[31]: 21

In [32]: pose = []
         for res in results.pose_landmarks.landmark:
             test = np.array([res.x, res.y, res.z, res.visibility])
             pose.append(test)

In [33]: pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).flatten() if results.pose_lar
         face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks else np.z
         lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landmarks e
         rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if results.right_hand_landmarks

In [34]: face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks else np.z

 In [6]: keypoints(results):
        p.array([[res.x, res.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).flatten() if results.pose_landmarks el
        p.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks else np.zeros(468*
        array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landmarks else np.ze
        array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if results.right_hand_landmarks else np.
        p.concatenate([pose, face, lh, rh])
```



```
In [36]: result_test = extract_keypoints(results)

In [37]: result_test
Out[37]: array([ 0.63101423,  0.37754777, -0.00280839, ...,  0.        ,
                 0.        ,  0.        ])

In [38]: 468*3+33*4+21*3+21*3
Out[38]: 1662

In [39]: np.save('0', result_test)

In [40]: np.load('0.npy')
Out[40]: array([ 0.63101423,  0.37754777, -0.00280839, ...,  0.        ,
                 0.        ,  0.        ])
```

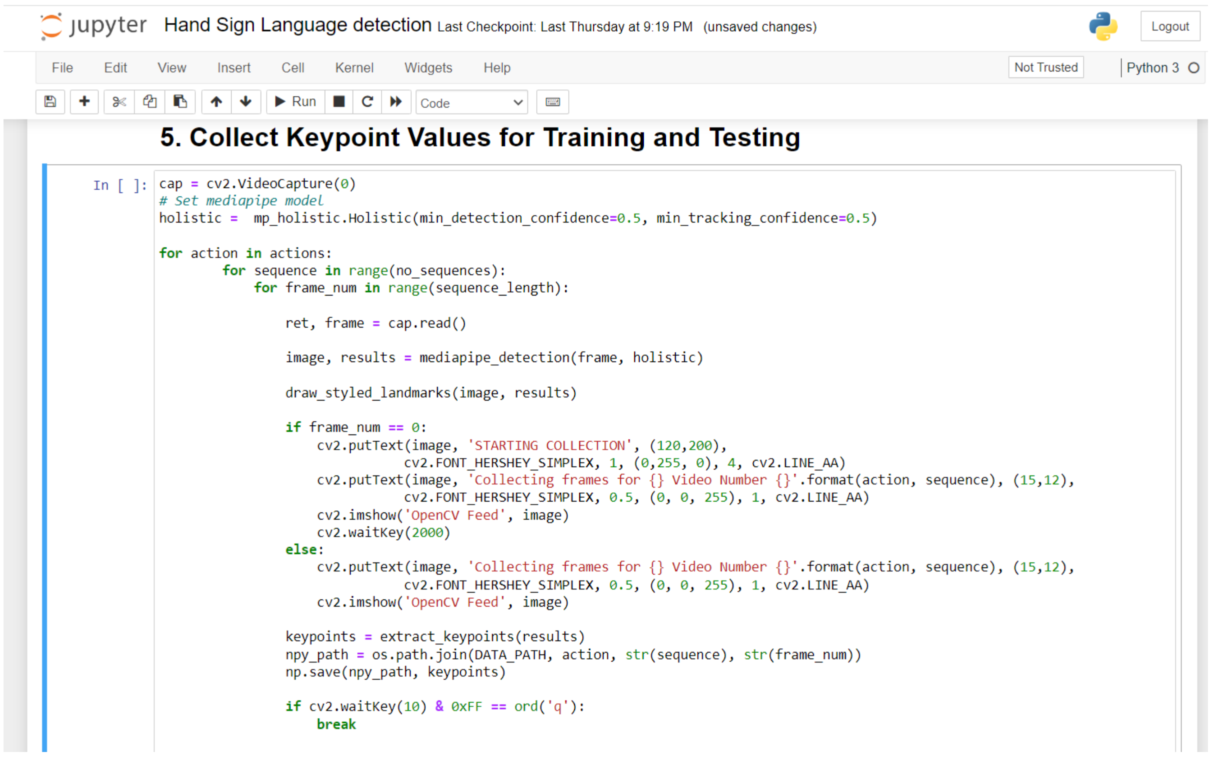


```
0.        , 0.        ])
```

## 4. Setup Folders for Collection

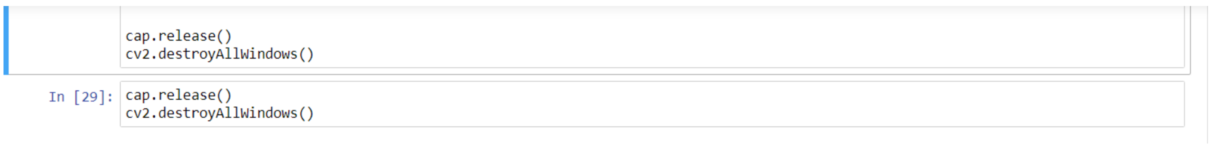
```
In [52]: DATA_PATH = os.path.join('MP_Data2')
         actions = np.array([
                 'A' ,
                 'B' ,
                 'C' ,
                 'D',


         ])
         no_sequences = 30

         # Videos of 30 frames
         sequence_length = 30
```

```
In [53]: for action in actions:
             for sequence in range(no_sequences):
                 try:
                     os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))
                 except:
                     pass
```


jupyter
Hand Sign Language detection Last Checkpoint: Last Thursday at 9:19 PM (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Not Trusted
Python 3
Run
Code

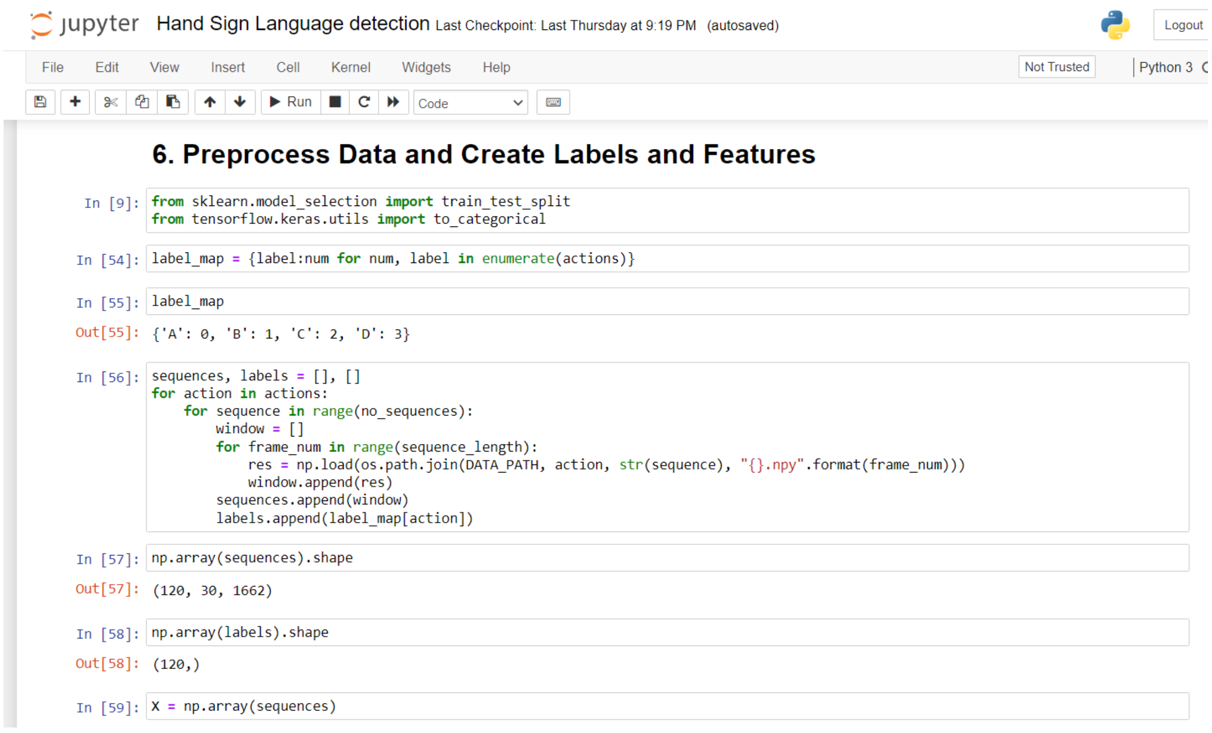
## 5. Collect Keypoint Values for Training and Testing

```
In [ ]: cap = cv2.VideoCapture(0)
        # Set mediapipe model
        holistic =  mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5)

        for action in actions:
                for sequence in range(no_sequences):
                    for frame_num in range(sequence_length):

                        ret, frame = cap.read()

                        image, results = mediapipe_detection(frame, holistic)

                        draw_styled_landmarks(image, results)

                        if frame_num == 0:
                            cv2.putText(image, 'STARTING COLLECTION', (120,200),
                                       cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE_AA)
                            cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),
                                       cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                            cv2.imshow('OpenCV Feed', image)
                            cv2.waitKey(2000)
                        else:
                            cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),
                                       cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                            cv2.imshow('OpenCV Feed', image)

                        keypoints = extract_keypoints(results)
                        npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
                        np.save(npy_path, keypoints)

                        if cv2.waitKey(10) & 0xFF == ord('q'):
                            break
```



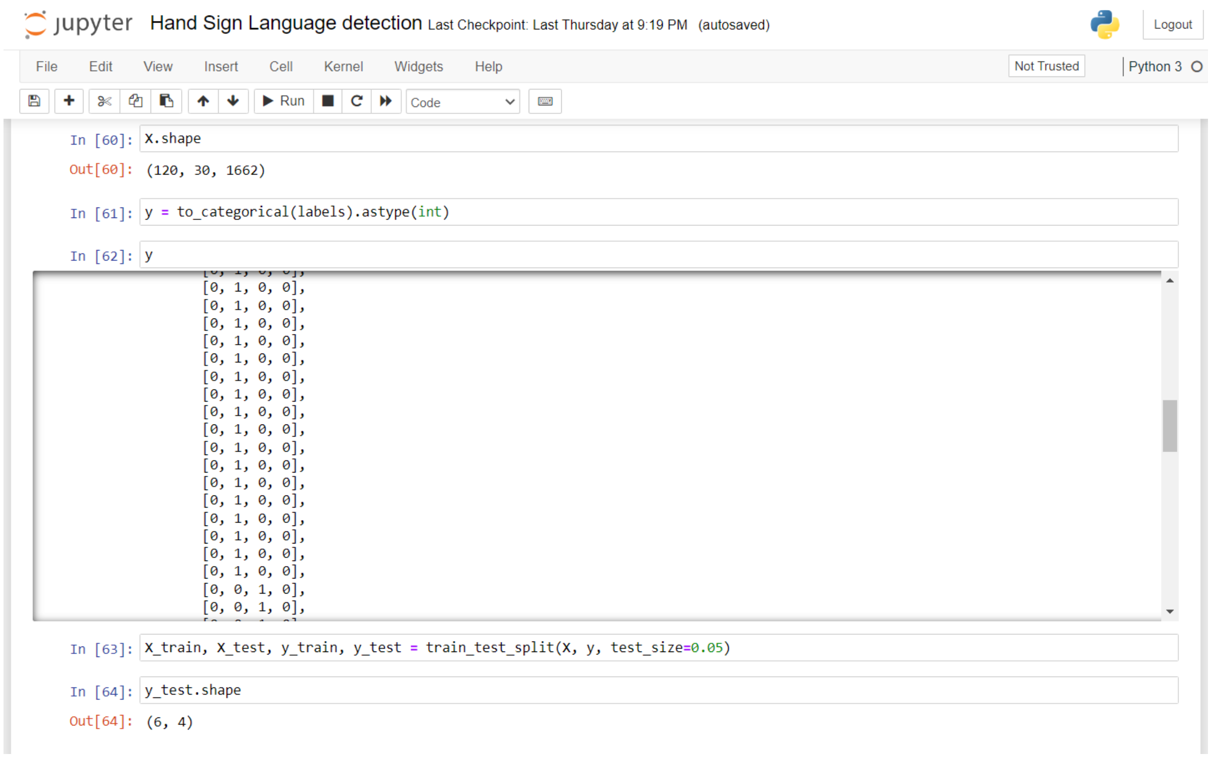
```
         cap.release()
         cv2.destroyAllWindows()
```

```
In [29]: cap.release()
         cv2.destroyAllWindows()
```

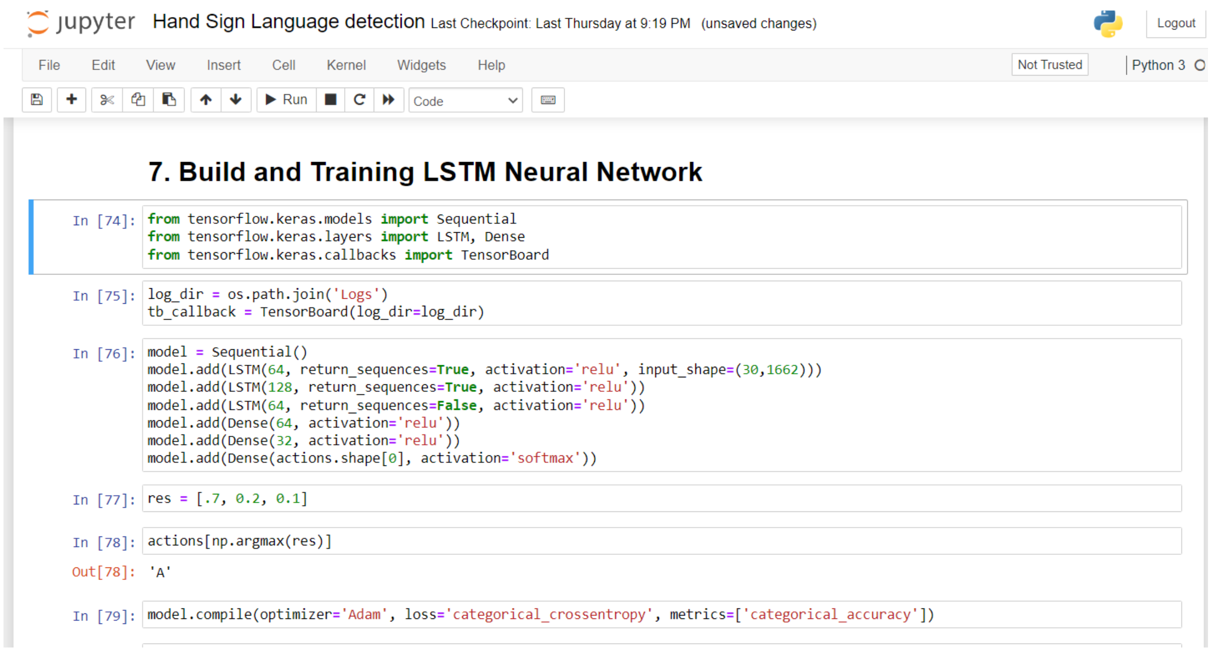

jupyter
Hand Sign Language detection Last Checkpoint: Last Thursday at 9:19 PM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Code

## 6. Preprocess Data and Create Labels and Features

```
In [9]: from sklearn.model_selection import train_test_split
        from tensorflow.keras.utils import to_categorical

In [54]: label_map = {label:num for num, label in enumerate(actions)}

In [55]: label_map

Out[55]: {'A': 0, 'B': 1, 'C': 2, 'D': 3}

In [56]: sequences, labels = [], []
         for action in actions:
             for sequence in range(no_sequences):
                 window = []
                 for frame_num in range(sequence_length):
                     res = np.load(os.path.join(DATA_PATH, action, str(sequence), "{}.npy".format(frame_num)))
                     window.append(res)
                 sequences.append(window)
                 labels.append(label_map[action])

In [57]: np.array(sequences).shape

Out[57]: (120, 30, 1662)

In [58]: np.array(labels).shape

Out[58]: (120,)

In [59]: X = np.array(sequences)
```


jupyter
Hand Sign Language detection Last Checkpoint: Last Thursday at 9:19 PM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Code

```
In [60]: X.shape

Out[60]: (120, 30, 1662)

In [61]: y = to_categorical(labels).astype(int)

In [62]: y

                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 1, 0, 0],
                [0, 0, 1, 0],
                [0, 0, 1, 0],

In [63]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)

In [64]: y_test.shape

Out[64]: (6, 4)
```


jupyter Hand Sign Language detection Last Checkpoint: Last Thursday at 9:19 PM (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Code

## 7. Build and Training LSTM Neural Network

```
In [74]: from tensorflow.keras.models import Sequential
         from tensorflow.keras.layers import LSTM, Dense
         from tensorflow.keras.callbacks import TensorBoard
```

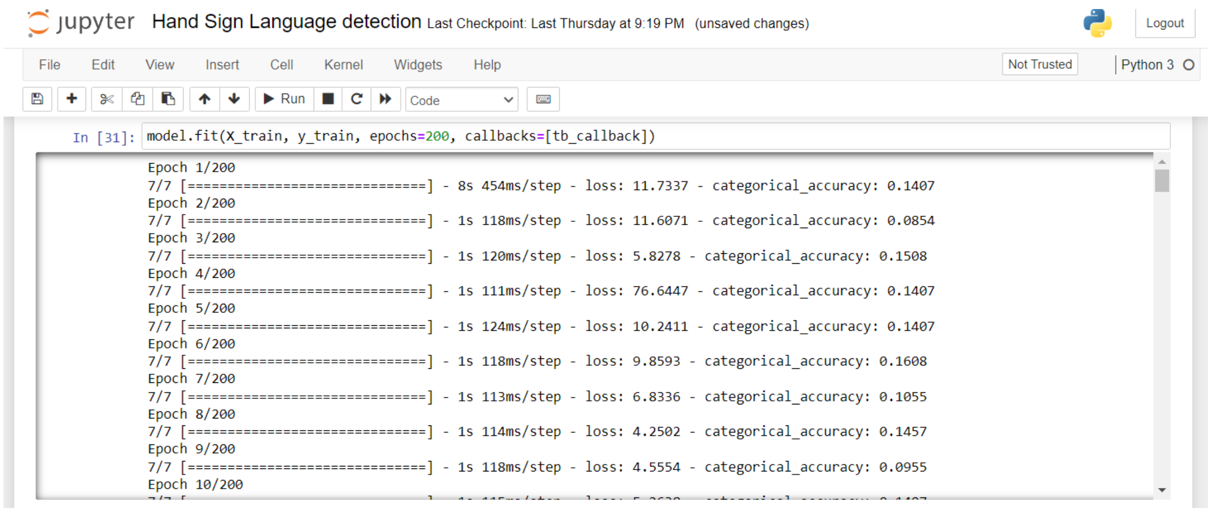
```
In [75]: log_dir = os.path.join('Logs')
         tb_callback = TensorBoard(log_dir=log_dir)
```

```
In [76]: model = Sequential()
         model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(30,1662)))
         model.add(LSTM(128, return_sequences=True, activation='relu'))
         model.add(LSTM(64, return_sequences=False, activation='relu'))
         model.add(Dense(64, activation='relu'))
         model.add(Dense(32, activation='relu'))
         model.add(Dense(actions.shape[0], activation='softmax'))
```

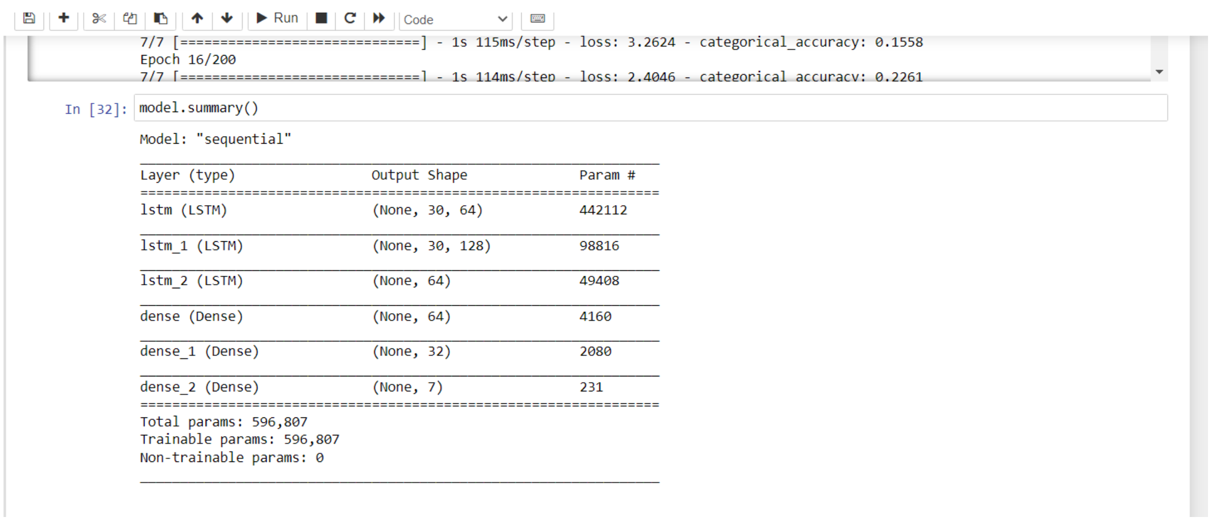
```
In [77]: res = [.7, 0.2, 0.1]
```

```
In [78]: actions[np.argmax(res)]
Out[78]: 'A'
```

```
In [79]: model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
```


jupyter Hand Sign Language detection Last Checkpoint: Last Thursday at 9:19 PM (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Code

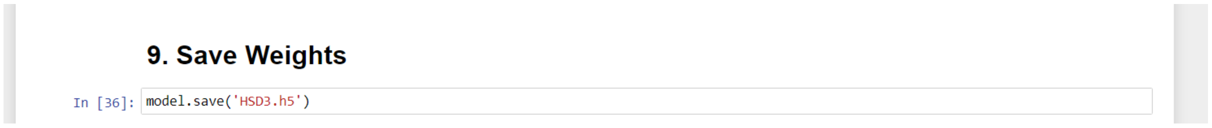
```
In [31]: model.fit(X_train, y_train, epochs=200, callbacks=[tb_callback])

         Epoch 1/200
         7/7 [==============================] - 8s 454ms/step - loss: 11.7337 - categorical_accuracy: 0.1407
         Epoch 2/200
         7/7 [==============================] - 1s 118ms/step - loss: 11.6071 - categorical_accuracy: 0.0854
         Epoch 3/200
         7/7 [==============================] - 1s 120ms/step - loss: 5.8278 - categorical_accuracy: 0.1508
         Epoch 4/200
         7/7 [==============================] - 1s 111ms/step - loss: 76.6447 - categorical_accuracy: 0.1407
         Epoch 5/200
         7/7 [==============================] - 1s 124ms/step - loss: 10.2411 - categorical_accuracy: 0.1407
         Epoch 6/200
         7/7 [==============================] - 1s 118ms/step - loss: 9.8593 - categorical_accuracy: 0.1608
         Epoch 7/200
         7/7 [==============================] - 1s 113ms/step - loss: 6.8336 - categorical_accuracy: 0.1055
         Epoch 8/200
         7/7 [==============================] - 1s 114ms/step - loss: 4.2502 - categorical_accuracy: 0.1457
         Epoch 9/200
         7/7 [==============================] - 1s 118ms/step - loss: 4.5554 - categorical_accuracy: 0.0955
         Epoch 10/200
```



```
7/7 [==============================] - 1s 115ms/step - loss: 3.2624 - categorical_accuracy: 0.1558
Epoch 16/200
7/7 [==============================] - 1s 114ms/step - loss: 2.4046 - categorical_accuracy: 0.2261
```

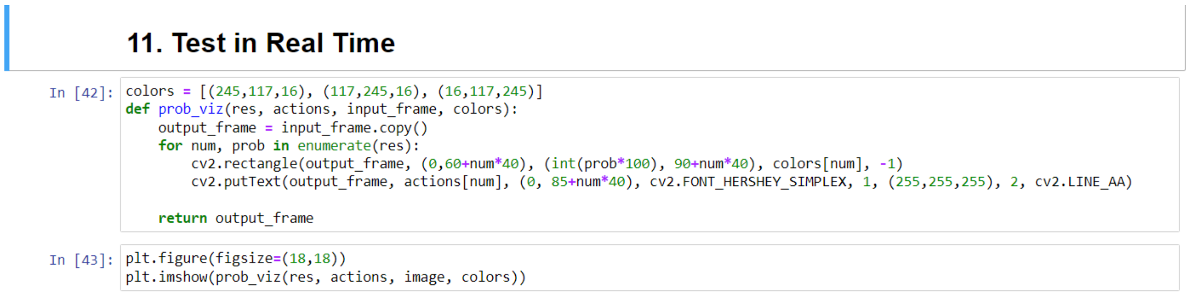
In [32]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 30, 64)            442112
_________________________________________________________________
lstm_1 (LSTM)                (None, 30, 128)           98816
_________________________________________________________________
lstm_2 (LSTM)                (None, 64)                49408
_________________________________________________________________
dense (Dense)                (None, 64)                4160
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 231
=================================================================
Total params: 596,807
Trainable params: 596,807
Non-trainable params: 0
_________________________________________________________________
```



## 9. Save Weights

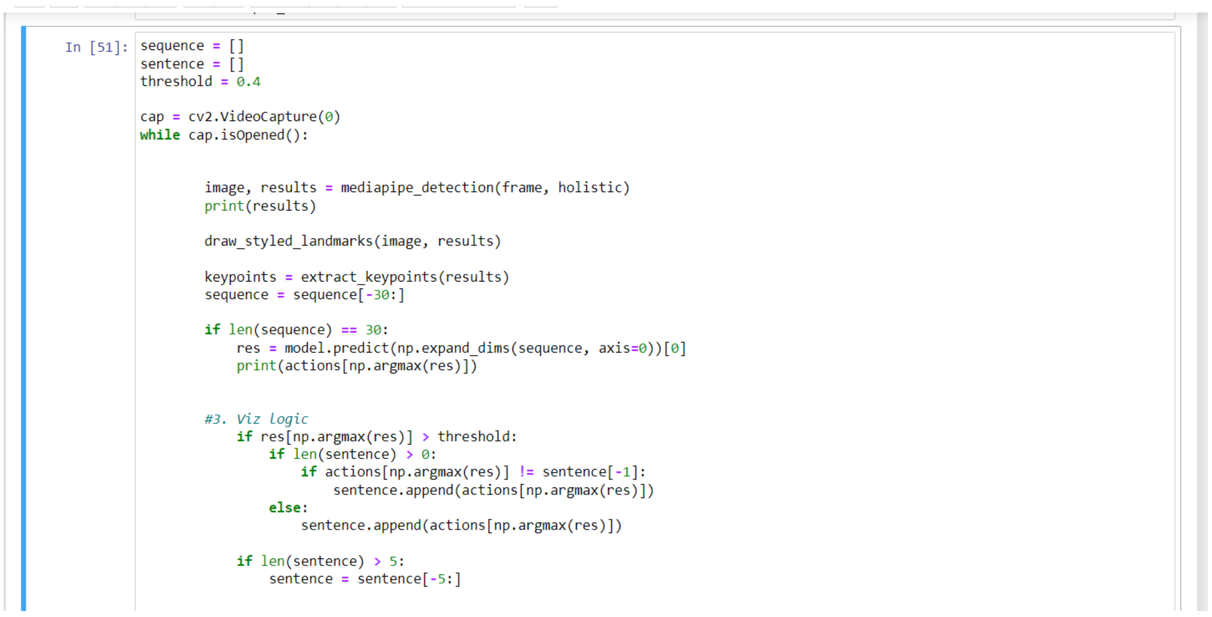
In [36]:
```
model.save('HSD3.h5')
```



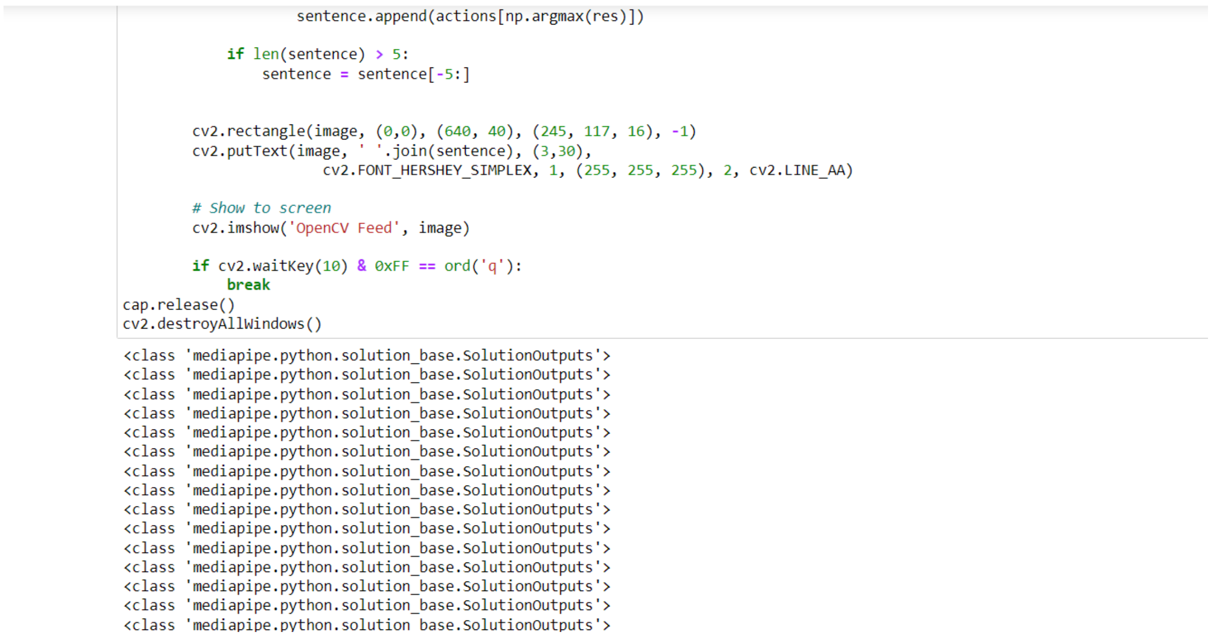
## 11. Test in Real Time

In [42]:
```
colors = [(245,117,16), (117,245,16), (16,117,245)]
def prob_viz(res, actions, input_frame, colors):
    output_frame = input_frame.copy()
    for num, prob in enumerate(res):
        cv2.rectangle(output_frame, (0,60+num*40), (int(prob*100), 90+num*40), colors[num], -1)
        cv2.putText(output_frame, actions[num], (0, 85+num*40), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2, cv2.LINE_AA)

    return output_frame
```

In [43]:
```
plt.figure(figsize=(18,18))
plt.imshow(prob_viz(res, actions, image, colors))
```

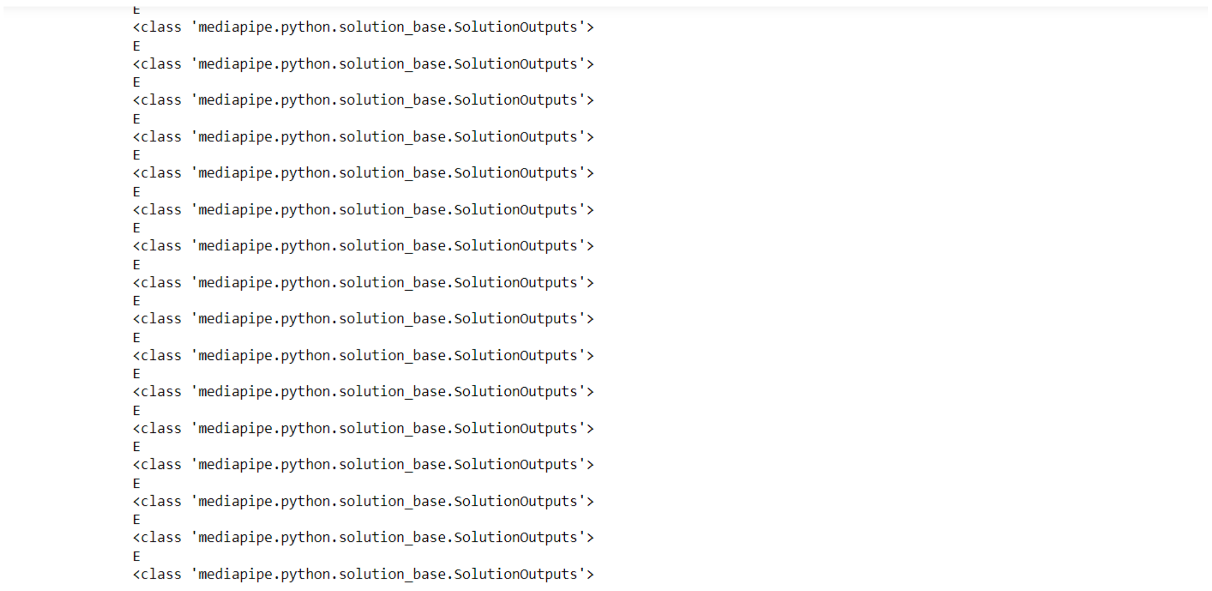


In [51]:
```
sequence = []
sentence = []
threshold = 0.4

cap = cv2.VideoCapture(0)
while cap.isOpened():


        image, results = mediapipe_detection(frame, holistic)
        print(results)

        draw_styled_landmarks(image, results)

        keypoints = extract_keypoints(results)
        sequence = sequence[-30:]

        if len(sequence) == 30:
            res = model.predict(np.expand_dims(sequence, axis=0))[0]
            print(actions[np.argmax(res)])


        #3. Viz logic
            if res[np.argmax(res)] > threshold:
                if len(sentence) > 0:
                    if actions[np.argmax(res)] != sentence[-1]:
                        sentence.append(actions[np.argmax(res)])
                else:
                    sentence.append(actions[np.argmax(res)])

            if len(sentence) > 5:
                sentence = sentence[-5:]
```



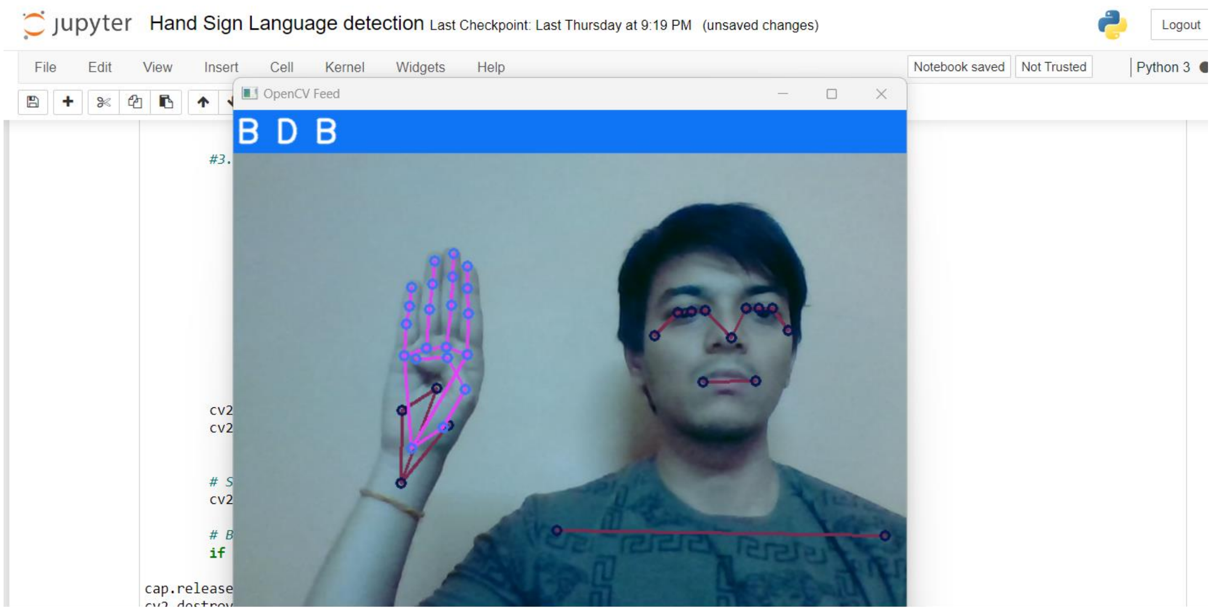
```
                    sentence.append(actions[np.argmax(res)])

            if len(sentence) > 5:
                sentence = sentence[-5:]


        cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1)
        cv2.putText(image, ' '.join(sentence), (3,30),
                       cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

        # Show to screen
        cv2.imshow('OpenCV Feed', image)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
cap.release()
cv2.destroyAllWindows()
```

```
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
```

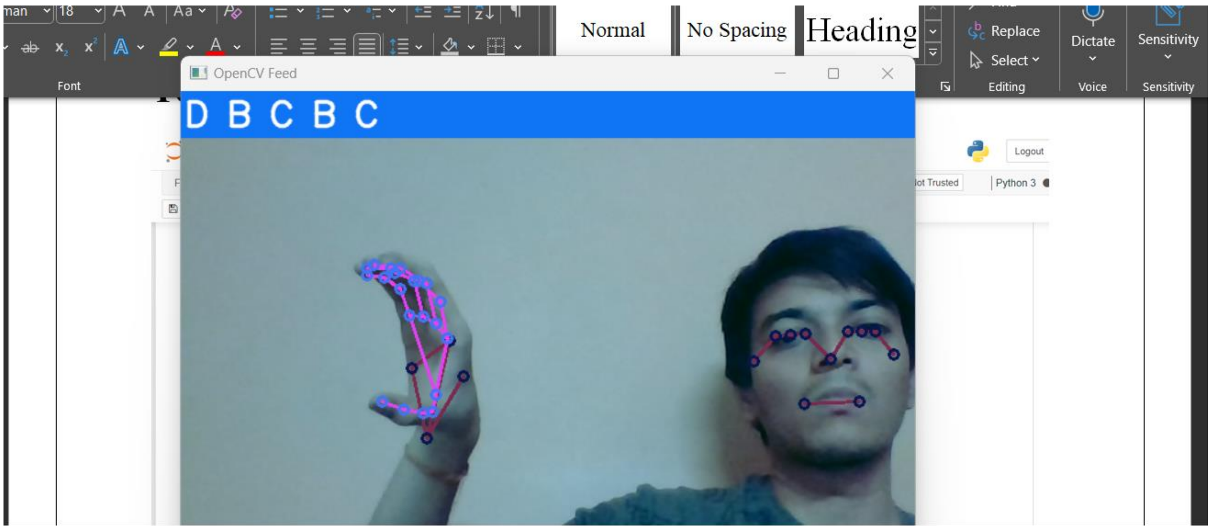


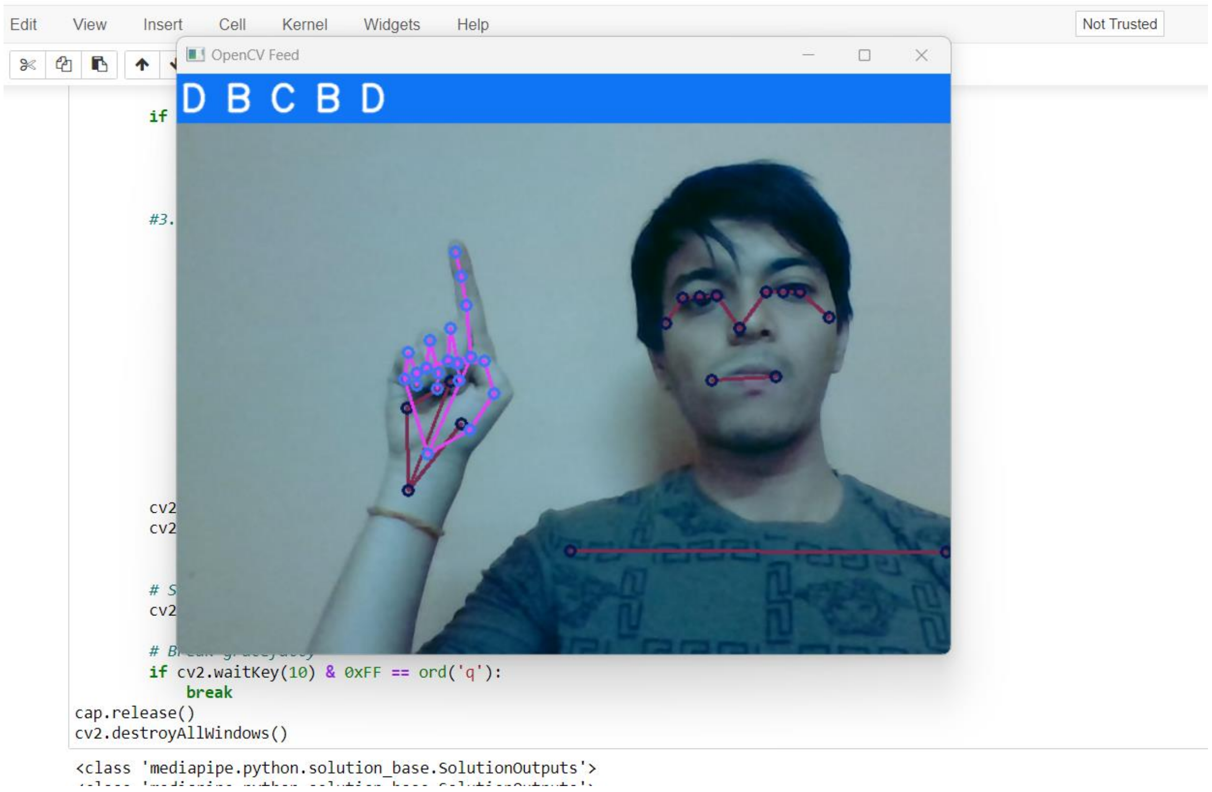
```
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
E
<class 'mediapipe.python.solution_base.SolutionOutputs'>
```

## Testing with real time data


Jupyter Hand Sign Language detection Last Checkpoint: Last Thursday at 9:19 PM (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Notebook saved
Not Trusted
Python 3
OpenCV Feed
B D B

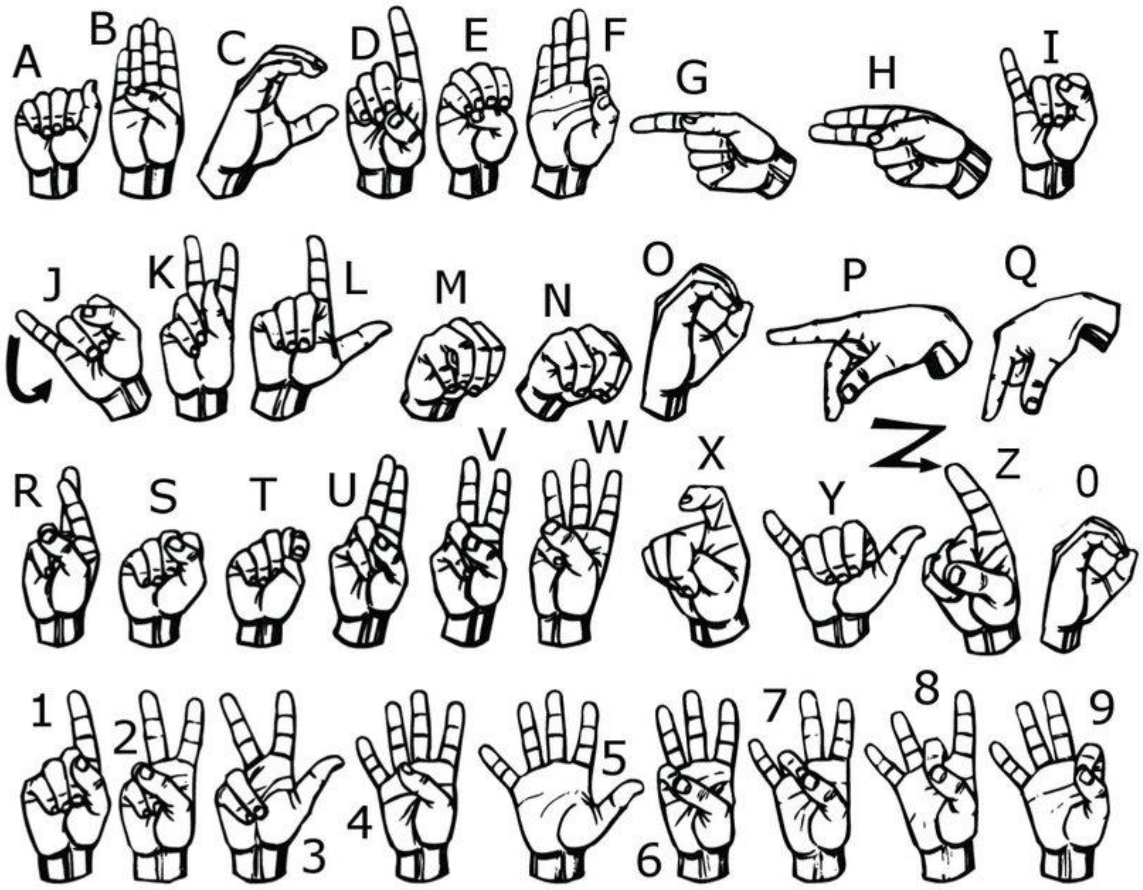
```
#3.
cv2
cv2
# S
cv2
# B
if
cap.release
```





```
        if

        #3.

        cv2
        cv2

        # S
        cv2

        # Br
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
cap.release()
cv2.destroyAllWindows()
```

```
<class 'mediapipe.python.solution_base.SolutionOutputs'>
```



# 7 : CODE DEVELOPMENT

1. Import and Install Dependencies

```
!pip install opencv-python mediapipe sklearn matplotlib
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp
```

## Keypoints using MP Holistic

```
mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils
def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image.flags.writeable = False
    results = model.process(image)
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    return image, results

def draw_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS) # Draw pose connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS) # Draw left hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS) # Draw right hand connections



def draw_styled_landmarks(image, results):

    # Draw pose connections

    mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS,

                             mp_drawing.DrawingSpec(color=(80,22,10), thickness=2,
circle_radius=4),

                             mp_drawing.DrawingSpec(color=(80,44,121), thickness=2,
circle_radius=2)

                             )

    # Draw left hand connections

    mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,

                             mp_drawing.DrawingSpec(color=(121,22,76), thickness=2,
circle_radius=4),

                             mp_drawing.DrawingSpec(color=(121,44,250), thickness=2,
circle_radius=2)

                             )

    # Draw right hand connections

    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,

                             mp_drawing.DrawingSpec(color=(245,117,66), thickness=2,
circle_radius=4),

                             mp_drawing.DrawingSpec(color=(245,66,230), thickness=2,
circle_radius=2)

                             )
```

## Extract Keypoint Values¶

```
len(results.left_hand_landmarks.landmark)

def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).flatten() if results.pose_landmarks else np.zeros(33*4)
    face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks else np.zeros(468*3)
    lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([pose, face, lh, rh])
```

## Setup Folders for Collection¶

```
DATA_PATH = os.path.join('MP_Data2')

actions = np.array([
        'A' ,
        'B' ,
        'C' ,
        'D',



])

no_sequences = 30
```

```
# Videos of 30 frames
sequence_length = 30


for action in actions:
    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))
        except:
            pass
cap = cv2.VideoCapture(0)
# Set mediapipe model
holistic = mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5)


for action in actions:
        for sequence in range(no_sequences):
            for frame_num in range(sequence_length):

                ret, frame = cap.read()

                image, results = mediapipe_detection(frame, holistic)

                draw_styled_landmarks(image, results)

                if frame_num == 0:
                    cv2.putText(image, 'STARTING COLLECTION', (120,200),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE_AA)
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action,
sequence), (15,12),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                    cv2.imshow('OpenCV Feed', image)
                    cv2.waitKey(2000)
                else:
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action,
sequence), (15,12),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                    cv2.imshow('OpenCV Feed', image)


                keypoints = extract_keypoints(results)
                npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
                np.save(npy_path, keypoints)


                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break


cap.release()
cv2.destroyAllWindows()


from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
label_map = {label:num for num, label in enumerate(actions)}
sequences, labels = [], []
for action in actions:
    for sequence in range(no_sequences):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(DATA_PATH, action, str(sequence), "{}.npy".format(frame_num)))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])
X = np.array(sequences)
y = to_categorical(labels).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
```

## Build and Training LSTM Neural Network¶

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard
log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)
model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(30,1662)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))
model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])

model.fit(X_train, y_train, epochs=200, callbacks=[tb_callback])

model.summary()
```

## Test in Real Time

```
sequence = []

sentence = []

threshold = 0.4


cap = cv2.VideoCapture(0)

# Set mediapipe model

holistic = mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5)


while cap.isOpened():


    # Read feed

    ret, frame = cap.read()


    # Make detections

    image, results = mediapipe_detection(frame, holistic)

    print(results)


    # Draw landmarks

    draw_styled_landmarks(image, results)
```

```
        # 2. Prediction logic
        keypoints = extract_keypoints(results)
#         sequence.insert(0,keypoints)
#         sequence = sequence[:30]
        sequence.append(keypoints)
        sequence = sequence[-30:]

        if len(sequence) == 30:
            res = model.predict(np.expand_dims(sequence, axis=0))[0]
            print(actions[np.argmax(res)])



        #3. Viz logic
            if res[np.argmax(res)] > threshold:
                if len(sentence) > 0:
                    if actions[np.argmax(res)] != sentence[-1]:
                        sentence.append(actions[np.argmax(res)])
                else:
                    sentence.append(actions[np.argmax(res)])

            if len(sentence) > 5:
                sentence = sentence[-5:]

            # Viz probabilities
            #image = prob_viz(res, actions, image, colors)
        cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1)
        cv2.putText(image, ' '.join(sentence), (3,30),
                       cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

        # Show to screen
        cv2.imshow('OpenCV Feed', image)



        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

# 8 : PROBLEM FACED

## PROBLEM FACED

- While building this model the problem faced was that the Deep Learning Algorithms LSTM requires a lot of precise training data.
- When we receive the video frames in the continuous format we need to separate the video frames.
- The other problem is that since the features are extracted it takes too much time to extract the features because the model is learning form it and also requires memory for calculating the features.
- The requirement of powerful GPU can be a problem to process the dataset as it required an efficient GPU for system training.

# 9 : SCOPE AND LIMITATIONS

## 9.1 Scope

The scope of hand sign language detection can be quite broad, depending on the specific context and application. Here are some possible examples:

- Accessibility: Hand sign language detection can be used to provide a more accessible experience for deaf or hard-of-hearing individuals, allowing them to communicate with others who do not understand traditional sign language.
- Human-Computer Interaction: Hand sign language detection can be used as a natural and intuitive way for people to interact with computers or other electronic devices, such as using gestures to control a video game or to navigate a virtual reality environment.
- Surveillance: Hand sign language detection can be used in security and surveillance contexts to monitor the movements and actions of individuals who may be using sign language to communicate secretly.
- Education: Hand sign language detection can be used in educational settings to help teach sign language to people who are learning it as a second language or who have no prior exposure to sign language.
- Medical Applications: Hand sign language detection can be used in medical contexts, for example, to help people with physical disabilities or conditions that affect their ability to communicate.

## 9.2 Limitations

- Hand sign language detection, like any technology, has its limitations. Here are some of the main ones:
- Accuracy: The accuracy of hand sign language detection can be limited, especially if the lighting conditions are poor, the camera angle is not optimal, or the user's hand movements are too fast or too subtle.
- Vocabulary: Hand sign language detection can only recognize a limited vocabulary of signs, which may not be sufficient for all users or situations.
- Variability: Sign languages can vary greatly between different countries and regions, making it challenging to develop a universal hand sign language detection system that can accurately recognize all signs.
- User Adaptability: Some users may find it difficult to adapt to using hand sign language detection, especially if they are used to traditional sign language or other forms of communication.
- Cost: The cost of implementing hand sign language detection technology can be high, especially if it requires specialized hardware or software development.
- Privacy: Hand sign language detection can raise concerns around privacy, as it may involve capturing and analyzing personal data and movements.

# 10 : FUTURE ENHANCEMENT

**Further Enhancement :**

- Wearable Technology: Wearable technology such as gloves or sensors could be used to capture and transmit hand movements, which could improve the accuracy of hand sign language detection and make it more practical for real-world use.
- E-learning Technology: An E-learning platform can build using this model
- Multi-Modal Input: Combining hand sign language detection with other input modalities such as speech recognition or facial expressions could enable more natural and flexible communication for users.
- Cross-Linguistic Sign Detection: Developing hand sign language detection systems that can recognize multiple sign languages could improve accessibility for users who communicate in different sign languages.