A PROJECT REPORT

on

"Sign Language detection using Long Short-Term Memory Model"

Submitted to

KIIT Deemed to be University

In Partial Fulfillment of the Requirement for the Award of

BACHELOR'S DEGREE IN
COMPUTER SCIENCE
AND
SYSTEMS ENGINEERING

BY

DRISHTI

Roll No. : 1928228

Reg. No. : 19592272019

UNDER THE GUIDANCE OF

Prof. Pradeep Kandula

SCHOOL OF COMPUTER ENGINEERING

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

BHUBANESWAR, ODISHA -751024

May 2023

KIIT Deemed to be University

School of Computer Engineering

Bhubaneswar, ODISHA 751024

CERTIFICATE

This is certify that the project entitled

"Sign Language detection using Long Short-Term Memory Model"

submitted by

DRISHTI

Roll No. : 1928228

Reg. No. : 19592272019

is a record of bonafide work carried out by her, in the partial fulfillment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science and Systems Engineering) at KIIT deemed to be university, Bhubaneswar. This work is done during the year 2022-2023, under my guidance.

Date:28/04/2023

Prof. Pradeep Kandula

(Project Guide)

ACKNOWLEDGEMENT

On this auspicious occasion of the completion of my “Sign Language detection using Long Short-Term Memory model” project, I would like to convey my heartfelt appreciation to Prof. Pradeep Kandula for his excellent direction and constant support in ensuring that this project met its objectives since its commencement to completion.

I would also like to express my sincere gratitude to our Director General of School of Computer Engineering (KIIT-DU) Prof. Biswajeet Sahoo for giving me this opportunity and all of the necessary resources to accomplish my project.

I would like to convey my deep appreciation to Deans of the School of Computer Engineering (KIIT-DU), Prof. Bhabani Shankar Prasad Mishra, Dr. Amulya Ratna Swain, and Dr. Arup Abhinna Acharya, for their encouragement in completion of my project.

Finally, I would like to thank my parents and friends, for their constant support throughout the project’s timeline.

DRISHTI
Roll No.: 1928228
Reg. No.: 19592272019

ABSTRACT

The inability to talk is considered a real impairment. People with this handicap utilize a variety of communication techniques to interact with others. Sign language is one of the most popular forms of communication among this group. Creating sign language applications for the deaf might be crucial since it will enable them to easily interact with individuals who do not understand sign language. Our initiative intends to take the initial step toward adopting sign language to reduce the communication gap between hearing individuals, the deaf, and the dumb. Making a vision-based system to recognize sign language motions from video sequences is the major goal of this effort. A system based on vision is preferred because it offers a more straightforward and natural means of communication between a human and a machine. Several different gestures have been taken into consideration for this project as examples.

If we were to sum up how this project operates, we might say that it is only a back-end. Here, we access the camera on our system using the MediaPipe version of OpenCV, which then recognizes the points on our hands and faces and establishes connections. By using the same procedure, we may create our own data sets. For this project's example, we have done four actions in total, each of which has thirty sequences and thirty videos that were recorded within them as .npy file. When we perform a specific action, it attempts to compare it with the other files that we independently created, and if it purportedly finds a match, it indicates how likely this model believes it to be a specific word. Live real-time testing displays the likelihood and the words from the action.

Keywords: Deep Learning LSTM Dense TensorFlow Keras

Contents

List of Figures

List of Tables

Chapter 1

Introduction

What does sign language represent and why is it important? The community of the deaf and dumb uses gestures to communicate in sign language. This group relied on sign language for communication when it was hard to transmit audio or when typing and writing were challenging but there was still the potential of vision. At that time, communication between people was only possible through sign language. When someone doesn't want to talk, they often utilize sign language, however for the deaf and dumb people, this is their sole means of communication. The same meanings that spoken language conveys are likewise conveyed through sign language. The deaf and dumb community uses this around the world, but in localized forms like ISL and ASL.

Hand gestures can be used to communicate in sign language, either with one hand or two. It comes in two varieties: continuous sign language and isolated sign language. While continuous ISL, also known as continuous sign language, is a series of gestures that produce a meaningful sentence, isolated sign language consists of a single gesture with a single word. We used an independent ISL gesture recognition method in this work. Deaf individuals utilize sign language, a visual language that employs a system of manual, facial, and bodily motions as the method of communication, in place of spoken language in their daily lives. Similar to the numerous spoken languages spoken around the world, sign language is not a universal language. Numerous sign languages may exist in some nations, including India, the United States, the UK, and Belgium. There are hundreds of sign languages in use worldwide, including Turkish Sign Language, British Sign Language (BSL), Spanish Sign Language, and Japanese Sign Language.
Sign language has three components
a. Fingerspelling : used to spell words letter by letter
b. Word level sign vocabulary : used for majority of communication
c. Non-manual features : facial expression and tongue, mouth and body position
ASL, which first emerged in 1817 to educate the deaf, now has about 1000000 impaired users.

However, there are only about 250000 non-disabled users, which is a small number [1]. In order to encourage its use, we are looking at how deep learning and sequence processing techniques may be used to interpret ASL in this project. To put it more specifically, we try to determine which words are depicted using the ASL fingerspelling set given a live video feed (via the computer's camera). Each identified word must be instantly displayed onto a window to tell the user. The user will be able to interactively write words on the screen in this fashion.

What purpose does the sign language detection system serve? Any movement of a bodily part, such as the hand or face, is a type of gesture. Here, we are utilizing sequence processing and deep learning for gesture identification. Gesture recognition makes it possible for computers to comprehend human behavior and serves as a translator between computers and people. This could make it possible for people to communicate organically with computers without coming into contact with any mechanical parts physically.

How does it work? Conducted research to determine which deep learning algorithms could be most effective for sign language recognition prior to deployment. Determined that the following ones are likely to fulfill the objectives in light of our findings: Principal Component Analysis, Linear Discriminant Analysis, Convolutional Neural Networks, Support Vector Machine, Mel Frequency Cepstral Coefficients Analysis, Gaussian Mixture Modelling, Linear Predictive Coding, K Nearest Neighbors and many more. Similarly, to identify how we could represent images as feature vectors, we needed to identify some basic image processing techniques. According to research : Static Thresholding, Adaptive Thresholding, Color Thresholding, Gaussian Filters, Morphological Operations (Dilation and Erosion), Logical Operations (and, or, not, xor), Edge Detection, Histogram Equalization, and many more. We will go into the specifics of the project later in the report but in the layman's term how does it exactly work?

- To further simplify the model process, we accept the input in the form of a picture from the system camera. The image passes through numerous pre-processing processes where it is resized, normalized, and given a different color format.
- Using mediapipe holistic, a predefined function to generate holistic models and drawing tools, we then try to identify the essential spots in the image. We introduce a function called media pipe detection where if we do the color conversion from BGR to RGB as the openCV reads the image in BGR form but media pipe needs RGB form to work further, then after we process the image for the model we again change the image from RGB to BGR because that's what we see in the image form. If you don't convert it then it comes as a blue green kind of image if you don't convert it.

- Using mediapipe holistic, a predefined function to generate holistic models and drawing tools, we then try to identify the essential spots in the image. We introduce a function called media pipe detection where if we do the color conversion from BGR to RGB as the opencv reads the image in BGR form but media pipe needs RGB form to work further, then after we process the image for the model we again change the image from RGB to BGR because that's what we see in the image form. If you don't convert it then it comes as a blue green kind of image if you don't convert it.

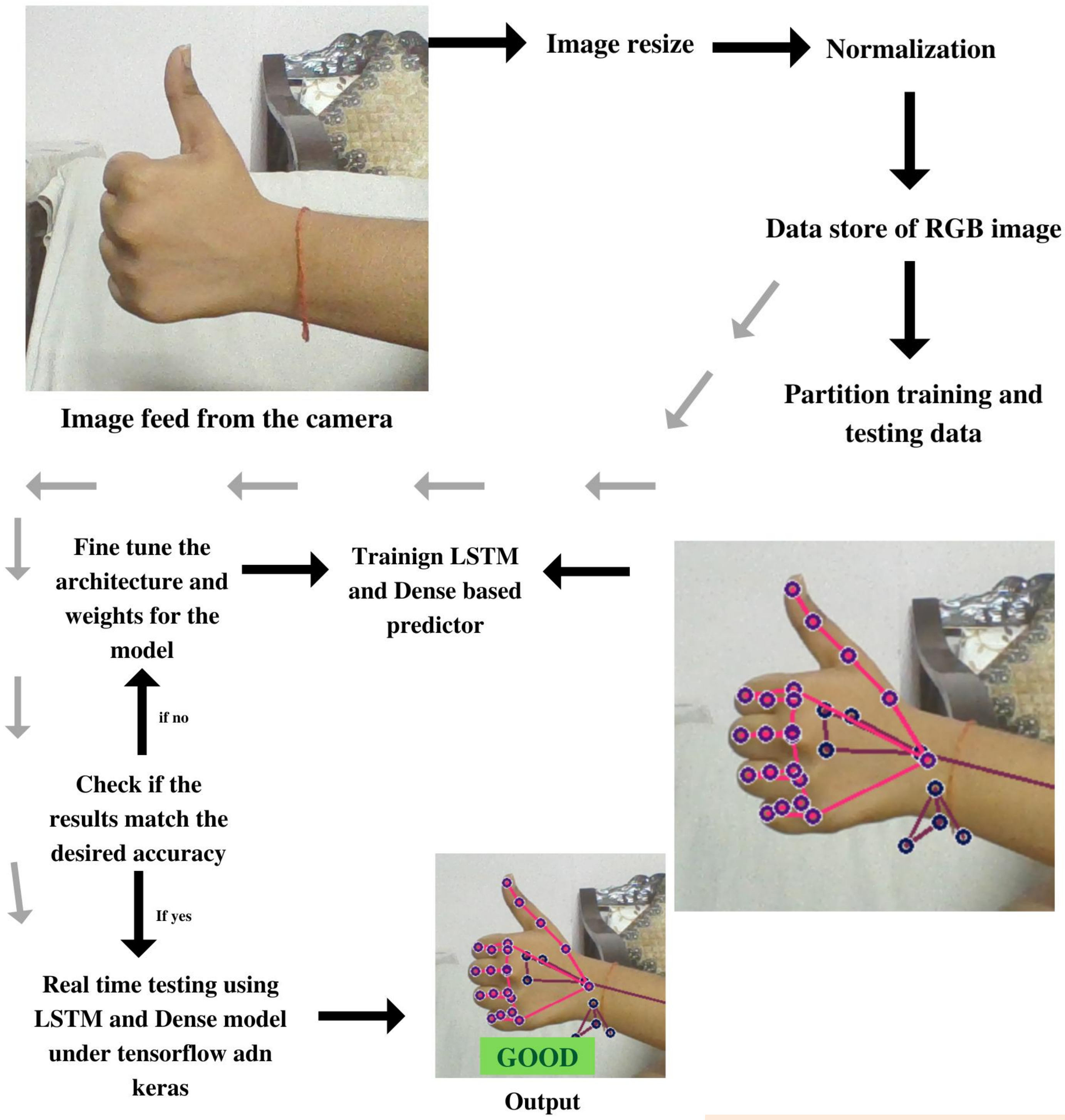


Image resize
Normalization
Data store of RGB image
Partition training and testing data
Image feed from the camera
Fine tune the architecture and weights for the model
Trainign LSTM and Dense based predictor
if no
Check if the results match the desired accuracy
If yes
Real time testing using LSTM and Dense model under tensorflow adn keras
GOOD
Output

Fig 1.1 : Layman working of the project

- The landmarks are then drawn in the face, position, right hand, and left hand of the picture using facemesh tesselation, position connection, and hand connection. Whether it be the sticks or the circle's thickness, we may design the landmarks and make them more distinctive before utilizing OpenCV's video capture function to take a picture. We take a picture using landmarks and MP holistic, then try to extract the essential data from it, like as where our head was held, how our left and right hands are positioned, and how our face is positioned, and we store it as a file.

- Then we try to save and capture 30 sequences in all the actions, and in those 30 sequences, the algorithm takes up 30 videos and captures 30.npy files, so if I have one action, it will contain 60.npy files. In my case, the folder is called mp_data or trial_data where we define the words that we want to use moving forward.

- Using scikit-learn train test split, tensorflow, and keras tools, we will attempt to analyse the data and construct labels and features after defining the sequence and labels.

- We train test split and build the LSTM neural network by introducing sequential LSTM, dense, and tensorboard to find the logs of the work we've been doing. b. We do the one hot encoder to further simplify the model process.

- The accuracy is determined when we construct our model, fit it, and get the summary. We next make a brief forecast and calculate the accuracy; if it is 1.0, the model is highly accurate and 100% correct.

- If the alternative values are larger than zero, we also discover the confusion matrix. The model will function more effectively the higher the value. Alternative values are the left upper and right lower values, which correspond to true negative and true positive, respectively.

- Then we test it in real time so we make sure we introduced a probability visualization in which whatever value we are doing, it is difficult to know whether our model is understanding what we are trying to do; therefore, to determine how the model is working, we write the probability visualization which enumerates more about the dynamic way of showing the output and then we tested in real time.

Chapter 2

Basic Concepts

2.1 Tools employed

2.1.1 Jupyter Notebook (via Anaconda Navigator)

The Jupyter Notebook is a free and open source web tool that lets us create and share documents with live code, equations, visualisations, and text. I utilised Anaconda Navigator to use it in this project. It's a desktop graphical user interface (GUI) that lets us effortlessly handle conda packages, environments, and channels without having to use command-line instructions. As a result, I didn't have to import every time I used a new type of function in my notebook when coding for machine learning.

2.1.2 Anaconda prompt

The command line interface for Anaconda Distribution is called Anaconda Prompt. Linux and macOS both have a command line interface called Terminal. The answer the Anaconda team came up with to enable quick setup for conda on Windows computers is Anaconda Prompt. With the exception of running a batch script at startup to activate the default conda environment, it is essentially simply the usual CMD.

2.1.3 Command prompt

The Command Prompt is an application that uses the Windows Graphical User Interface (GUI) to simulate the input field in a text-based user interface screen in Windows operating systems. It may be used to carry out complex administrative tasks and put entered directives into action. In a text-based user interface screen for an operating system (OS) or software, it is the input field. The question is intended to compel a response. The user puts command prompt commands into the flashing cursor that follows a short text string known as the command prompt.

2.2 Imports/installs employed

2.2.1 TensorFlow gpu

Google's well-known open source machine learning framework is called TensorFlow. It may be used to perform mathematical operations on CPUs, GPUs, and Tensorflow Processing Units (TPUs) that are exclusive to Google. GPUs are frequently employed for deep learning model inference and training. We must have the necessary GPU device drivers and setup TensorFlow to utilize GPUs (which is slightly different for Windows and Linux PCs) in order for it to function with GPUs. Then, by default, TensorFlow performs operations on your GPUs. TensorFlow can be told to utilize a CPU even when a GPU is available; we can select which GPU it uses for a specific task.

2.2.2 TensorFlow

Google developed and released TensorFlow, a Python library for quick numerical computations. It is a foundation library that may be used to build Deep Learning models directly or indirectly using wrapper libraries created on top of TensorFlow to make the process easier. in simple terms An open source framework called TensorFlow was created by Google researchers to handle deep learning, machine learning, and other tasks including statistical and predictive analytics.TensorFlow is meant to operate with huge data sets composed of several unique individual properties. Any information that you want to use TensorFlow to process must be kept in the multi-dimensional array. Tensors are yet another name for these multidimensional arrays.

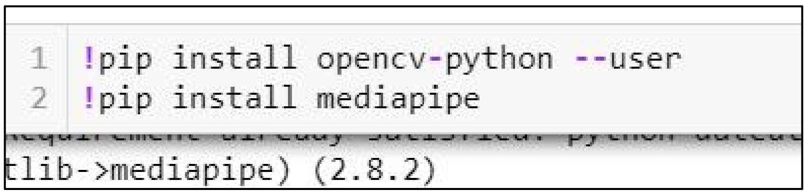
2.2.3 OpenCV python

A computer vision and machine learning software library called OpenCV is available for free use. A standard infrastructure for computer vision applications was created with OpenCV in order to speed up the incorporation of artificial intelligence into products. OpenCV was created to execute computing-intensive vision tasks as effectively and quickly as possible. As a result, it places a lot of emphasis on real-time AI vision applications. The program is multi-threaded and written in C that has been optimized for multicore CPUs.

2.2.4 MediaPipe

For developers wishing to incorporate computer vision and machine learning into their applications, MediaPipe Python is a potent tool. For the development of real-time ML solutions for mobile, edge, cloud, and online, it offers a high-level API. It is mostly used for quick prototyping of inferencing pipelines for perception using AI models and other reusable parts. Additionally, it makes it easier to integrate computer vision software into applications and demos running on various hardware platforms.

It differs from OpenCV in that the OpenCV offers real-time optimized hardware, techniques, and libraries for computer vision. For live and streaming video, MediaPipe provides cross-platform, adaptable machine learning solutions.



```
!pip install opencv-python --user
!pip install mediapipe
tlib->mediapipe) (2.8.2)
```

Fig 2.1 mediapipe installation

2.2.5 NumPy

NumPy may be used to conduct a wide range of array-based mathematical operations. It extends Python with sophisticated data structures that ensure fast computations with arrays and matrices, as well as a large library of high-level mathematical functions that work with these arrays and matrices.

2.2.6 Scikit-learrn

For the Python programming language, Scikit-learn is a machine learning library. Support-vector machines, random forests, gradient boosting, and other classification, regression, and clustering techniques are included.

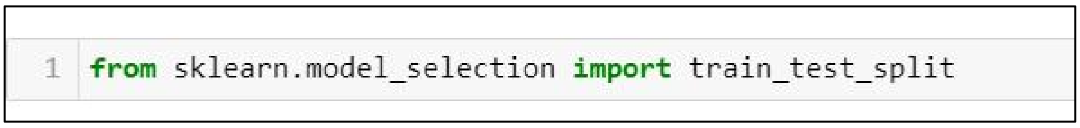
2.2.6.1 Train Test split

A model validation technique called train-test-split allows you to mimic how a model would perform on brand-new, untested data. When machine learning algorithms are used to generate predictions on data that was not used to train the model, it is used to estimate how well they perform. To evaluate prediction performance objectively, you must divide your dataset.

2.2.6.2 Multilabel confusion matrix

In multiclass tasks, labels are binarized in a one-vs-rest manner; nonetheless, confusion_matrix creates one confusion matrix for confusion between every two classes. The multilabel_confusion_matrix computes class-wise or sample-wise multilabel confusion matrices.

To determine the accuracy of a classification, compute a class- or sample-wise (samplewise=True) multilabel confusion matrix and output confusion matrices for each.

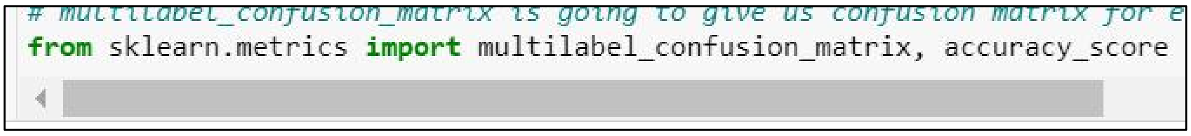


```
from sklearn.model_selection import train_test_split
```

Fig 2.2 Train test import

2.2.6.3 Accuracy score

By dividing the total number of forecasts by the number of right predictions, the accuracy score is determined. The terms True Positive, True Negative, False Positive, and False Negative from the Confusion Matrix are useful for describing Accuracy because they are used to measure the performance of the model by summing the True Positives and True Negatives from all the predictions.



```
# multilabel_confusion_matrix is going to give us confusion matrix for e
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score
```

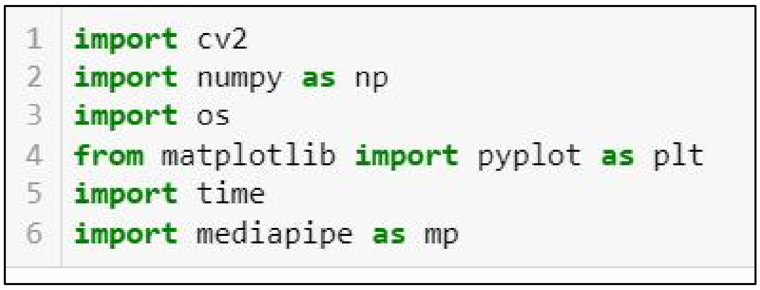
Fig 2.3 confusion matrix and accuracy score import

2.2.7 Matplotlib

Matplotlib is a charting library for the Python programming language and NumPy, the Python numerical mathematics extension. It aids in data visualisation and a better comprehension of the data.

2.2.8 OS

The Python OS module gives users the ability to create interactions with their operating systems. It provides a variety of practical OS features that may be utilized to carry out OS-based operations and obtain pertinent OS-related data. Python's basic utility modules cover the OS. It offers capabilities for adding and deleting directories (folders), retrieving their contents, changing the current directory, and more. Before we can communicate with the underlying operating system, we must import the os module.



```
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp
```

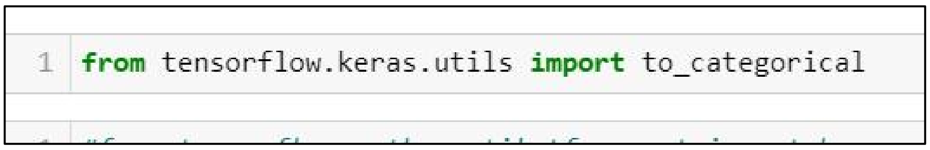
Fig 2.4 Multiple import of several modules

2.2.9 Keras

Google created the high-level Keras deep learning API to implement neural networks. It is used to make the implementation of neural networks simple and is developed in Python. Additionally, multiple backend neural network computation is supported.

2.2.9.1 to_categorical

It is used to transform the class vector into the binary class matrix. It will provide output in either a format of 1 or 0.

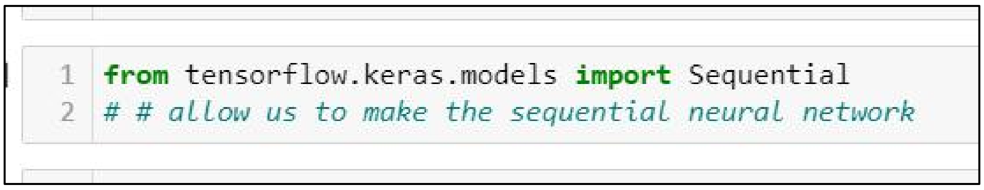


```
from tensorflow.keras.utils import to_categorical
```

Fig 2.5 to_categorical import

2.2.9.2 Sequential

The sequential approach enables us to define a neural network precisely sequentially, moving from input to output while traversing a sequence of sequential neural layers. For a simple stack of layers where each layer has precisely one input tensor and one output tensor, a sequential approach is sufficient. The add() technique allows us to build a Sequential model progressively as well.



```
from tensorflow.keras.models import Sequential
# # allow us to make the sequential neural network
```

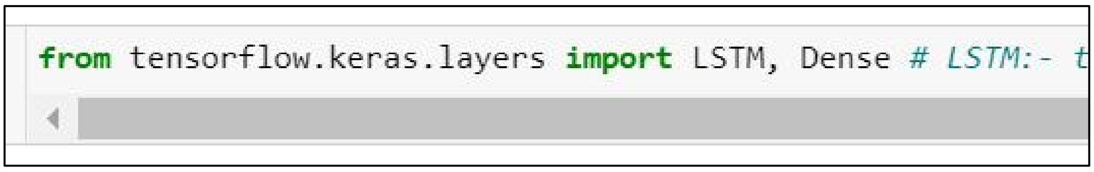
Fig 2.6 Sequential import

2.2.9.3 LSTM

Long short-term memory networks, or LSTMs, are employed in deep learning. Many recurrent neural networks (RNNs) are able to learn long-term dependencies, particularly in problems involving sequence prediction. Due to its ability to understand long-term connections between data time steps, LSTMs are frequently used to learn, analyze, and categorize sequential data. Based on prior, sequential data, it makes predictions about future values. This improves demand forecasters' accuracy, which helps the firm make better decisions.

2.2.9.4 Dense

One of the often utilized layers in the keras model or neural network, where all connections are established extremely deeply, is keras dense. In other words, all of the neurons in the network's preceding layer serve as the dense layer's source of input data. Each neuron in this basic layer, which is considered dense, gets information from every neuron in the layer below it. Images are categorized using the dense layer based on the results of the convolutional layers.

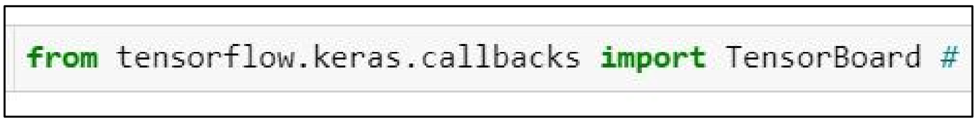


```
from tensorflow.keras.layers import LSTM, Dense # LSTM:- t
```

Fig 2.7 LSTM, Dense import

2.2.9.5 Tensorboard

With the help of the open source toolkit TensorBoard, we can analyze the status of the training process and enhance model performance by changing the hyperparameters. The dashboard for the TensorBoard toolkit shows many ways to view logs, including graphs, pictures, histograms, embeddings, text, and more.



```
from tensorflow.keras.callbacks import TensorBoard #
```

Fig 2.8 Tensorboard import

Chapter 3

Problem Statement

To create a sign language detection system using Keras' Long Short Term Memory model and Dense model. It's objective would be identifying words through actions in real time on any simplified system.

3.1 Project Planning

3.1.1 We start off by making TensorFlow GPU using command prompt named as gputest. gputest is an environment which we will create which will have its own TensorFlow, Keras and other modules present in it and will be situated in its own kernel, which too we will create separately

3.1.2 We install all the required modules Like OpenCV, MediaPipe, scikit-learn, matplotlib and import further modules from it. then we create a user defined function to find the key points using mpholistic under MediaPipe, then we draw landmarks to it, personalize it and capture the live video feed from our systems camera.

3.1.3 We extract the key point values whether its head position, body position, hands position and put it into a simplified numpy array using a user defined function known as extract key points and we save it into a separate .npy file

3.1.4 We setup folders for collecting the data set which we will be creating. we will be capturing the actions which will have 30 sequences and each sequence will have 30 npy files saved in it for the comparison in the later stages

3.1.5 We import train test split and to_categorical like modules from TensorFlow, keras we encode our data for further smooth processing and we build train and test LSTM neural network using sequential model and while doing it we can visualize the loss and the accuracy of each and every epoch through tensorboard as we will be fitting the model and we can make predictions and save weights in a separate file.

3.1.6 To ensure that our model is working perfectly we use the confusion matrix and accuracy score to judge our prediction.

3.1.7 We test it in real time including a new user defined function called as probability visualization where we can see how does the model think while visualizing the actions.

3.2 Project Analysis

Tensor flow and keras are fairly complex modules for Anaconda and Python versions, so there may be times when they cannot be installed directly in Jupiter using pip. In these cases, the user must use the command prompt to start the creation of a new kernel and environment that will set up the GPU to support the model's operation and install the tensor flow and keras modules in it so that when we run the Jupiter file. We go to the kernel and choose the gpu environment under Change Kernel in the jupyter notebook. In the gpu context, this would enable us to deal with tensorflow and keras. Then, we install all of the project's necessary modules in the same environment and import all of the submodules.

The image and the model we will be building for the prediction are input during the second process, when we try to extract the key points using mp_holistic using user defined MediaPipe_detection function. We will change the color of the image from BGR to RGB because openCV reads the image in BGR form but media pipe needs RGB form to continue working. The pictures are then marked as temporarily unwritable while predictions are created to prevent input from changing, after which the model is processed, the forecast is made, and the image is made writable once more. For us to see it and for mediapipe to function correctly, the color is changed from RGB to BGR, and then the image and results are provided. Using facemesh tesselation, pose connections, and hand connections, we can create landmarks on our face, pose, left hand, and right hand We can also customize the appearance of our landmarks by using mp_drawing, and we can use these user-defined functions when we run openCV. The subsequent section will address additional phases in the implementation..

The project can be said to be divided into four parts primarily. That being the first one, importing and installing all the required modules. Second, finding all the key points, collecting all the data in folders user defined in the program. Third, building a suitable model to predict accurately and run it for enough epochs and fourth, testing it live in real time using system's camera.

3.3 System Design

3.3.1 Design Constraints

- Software used
  - Python : Python 3.10
  - Libraries : TensorFlow, Keras, Matplotlib, Scikit-learn etc
  - Operating system : Windows 11
  - IDE : Jupyter notebook, command prompt, anaconda prompt
- Hardware used : HP laptop with basic hardware

3.3.2 Block Diagram

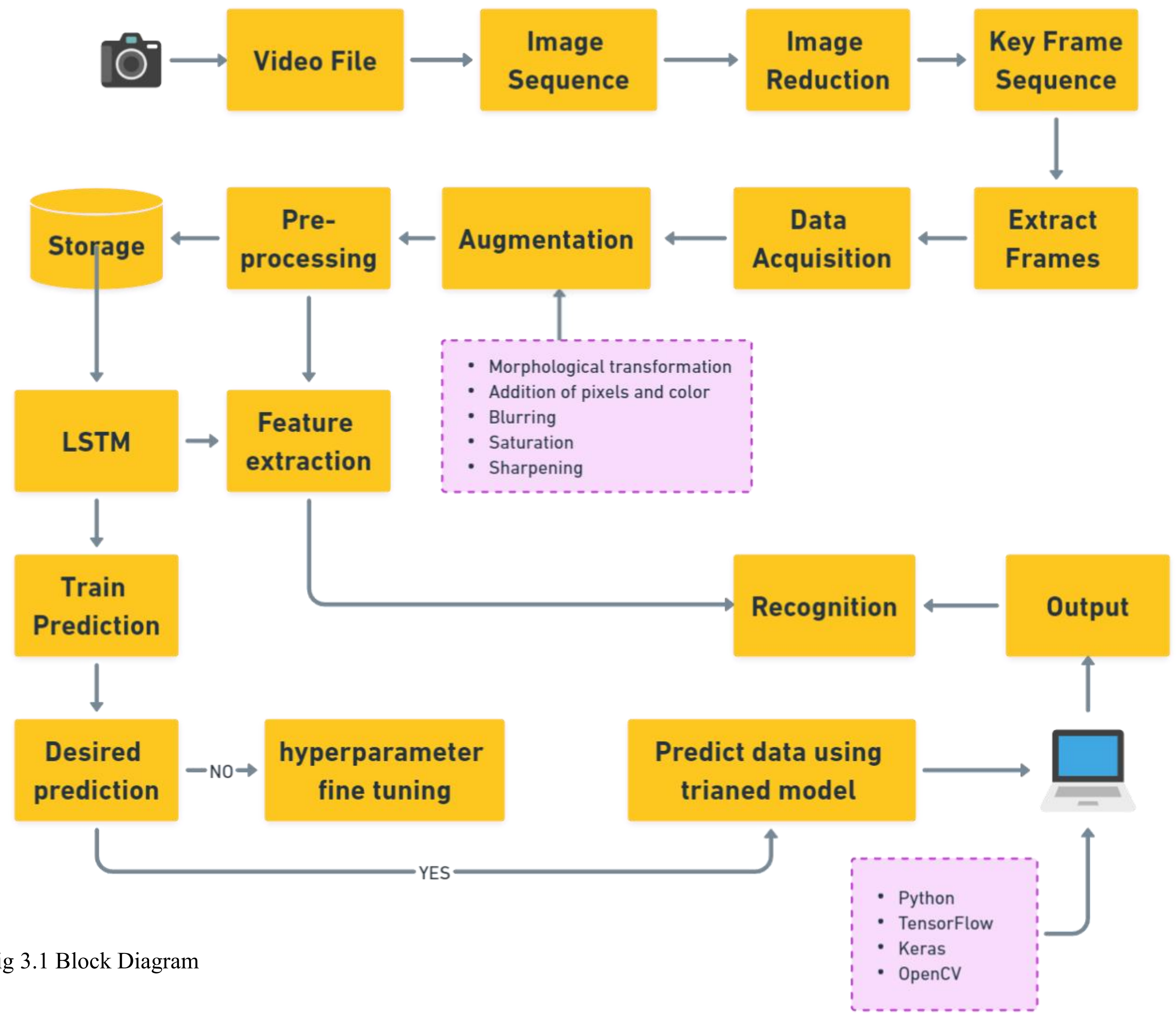


Fig 3.1 Block Diagram

Chapter 4

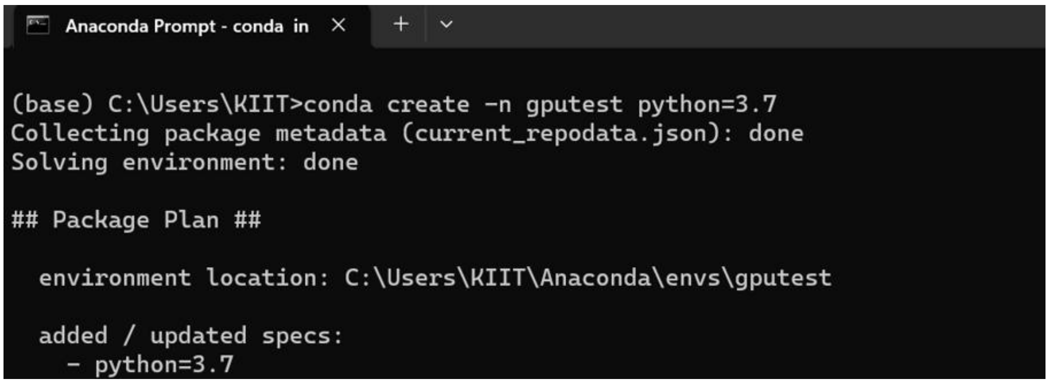
Implementation

4.1 Methodology

4.1.1 Setting up the kernel and gpu

4.1.1.1 Open the command prompt or anaconda prompt

4.1.1.2 Create a new environment in the prompt with the name we want our gpu to have with the python version. After we will press enter it will start collecting data.



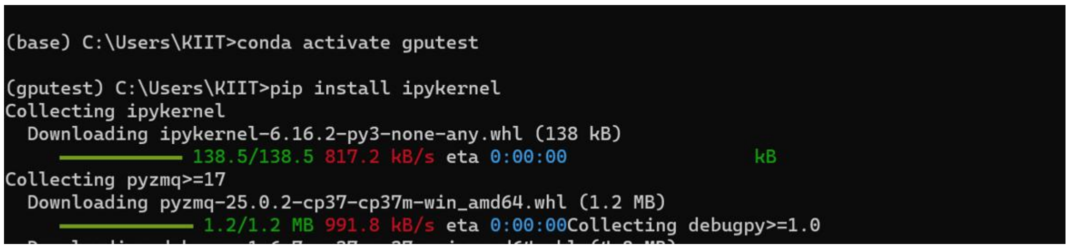
Anaconda Prompt - conda in

```
(base) C:\Users\KIIT>conda create -n gputest python=3.7
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\KIIT\Anaconda\envs\gputest

  added / updated specs:
    - python=3.7
```

Fig 4.1 creating environment

4.1.1.3 Now we will activate the environment which we made, so anything in the near future we want to do can be done in that environment.

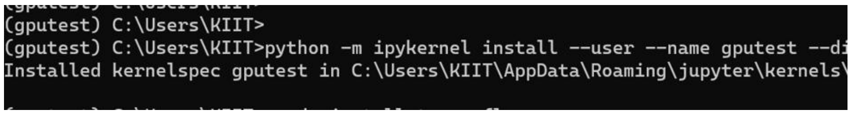
4.1.1.4 As we have created an environment, we have to install a kernel to the environment. Why? as it knows which hardware resources are available and which processes need them. It then allocates time for each process to use those resources.



```
(base) C:\Users\KIIT>conda activate gputest

(gputest) C:\Users\KIIT>pip install ipykernel
Collecting ipykernel
  Downloading ipykernel-6.16.2-py3-none-any.whl (138 kB)
     ━━━━━━━━━━━━━━━━ 138.5/138.5 817.2 kB/s eta 0:00:00                kB
Collecting pyzmq>=17
  Downloading pyzmq-25.0.2-cp37-cp37m-win_amd64.whl (1.2 MB)
     ━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 991.8 kB/s eta 0:00:00Collecting debugpy>=1.0
```

Fig 4.2 activating environment

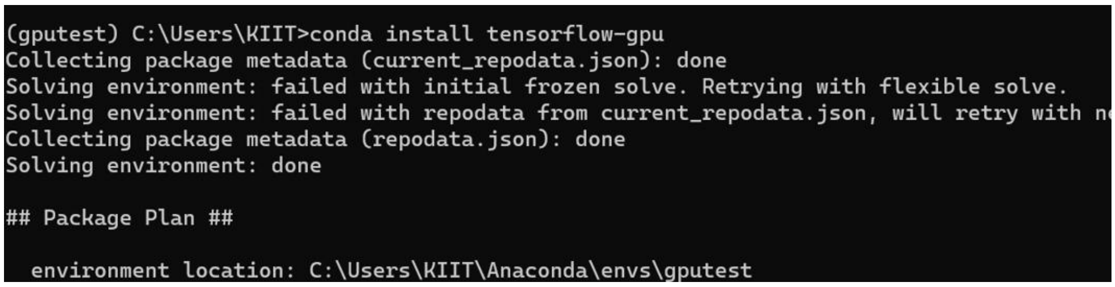
4.1.1.5 We install the kernel in the gpu as we already installed it on the system



```
(gputest) C:\Users\KIIT>
(gputest) C:\Users\KIIT>python -m ipykernel install --user --name gputest --di
Installed kernelspec gputest in C:\Users\KIIT\AppData\Roaming\jupyter\kernels\
```

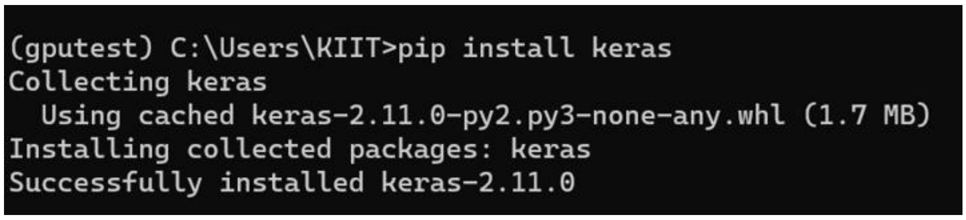
Fig 4.3 installing kernel

4.1.1.6 Now as we have already installed the gpu and kernel, we can expect TensorFlow and keras to run smoothly on the system as well. Installing TensorFlow gpu and keras and specific jupyter notebbok for the gputest



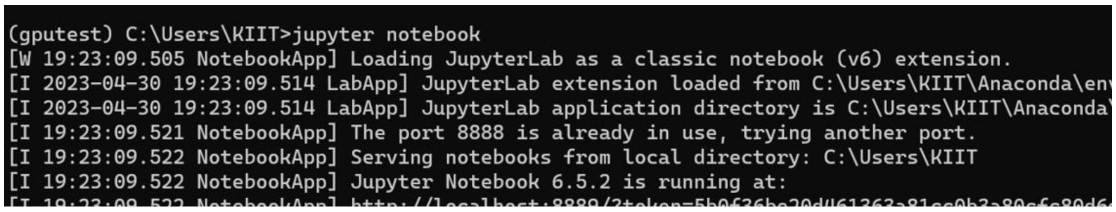
```
(gputest) C:\Users\KIIT>conda install tensorflow-gpu
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: failed with repodata from current_repodata.json, will retry with ne
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\KIIT\Anaconda\envs\gputest
```

Fig 4.4 Installing tensorflow gpu



```
(gputest) C:\Users\KIIT>pip install keras
Collecting keras
  Using cached keras-2.11.0-py2.py3-none-any.whl (1.7 MB)
Installing collected packages: keras
Successfully installed keras-2.11.0
```

Fig 4.5 installing keras

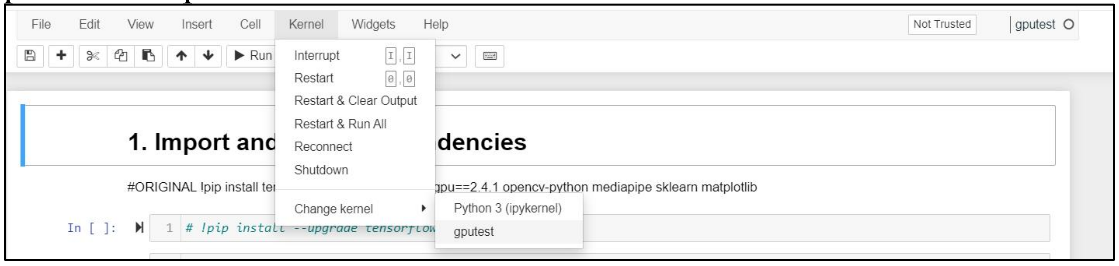


```
(gputest) C:\Users\KIIT>jupyter notebook
[W 19:23:09.505 NotebookApp] Loading JupyterLab as a classic notebook (v6) extension.
[I 2023-04-30 19:23:09.514 LabApp] JupyterLab extension loaded from C:\Users\KIIT\Anaconda\env
[I 2023-04-30 19:23:09.514 LabApp] JupyterLab application directory is C:\Users\KIIT\Anaconda\
[I 19:23:09.521 NotebookApp] The port 8888 is already in use, trying another port.
[I 19:23:09.522 NotebookApp] Serving notebooks from local directory: C:\Users\KIIT
[I 19:23:09.522 NotebookApp] Jupyter Notebook 6.5.2 is running at:
```

Fig 4.6 opening jupyter notebook in the

4.1.2 Jupyter notebook

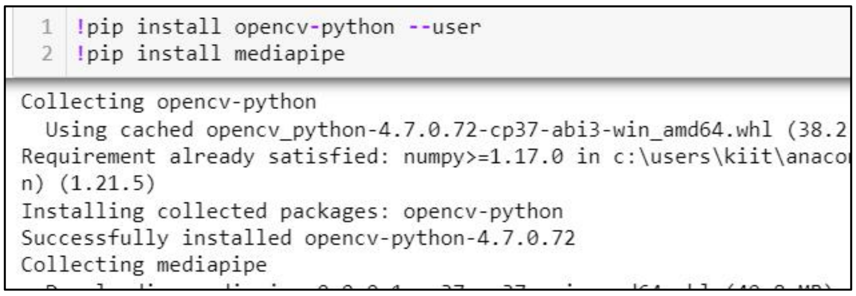
4.1.2.1 Before starting with any installation and imports we need to check that our system is working on correct kernel which in our case is gputest. Just in case we want to be sure, click on kernel and then change kernel and click the environemnt which we use to install the TensorFlow gpu and keras in the previous step.



1. Import and dencies

```
#ORIGINAL !pip install te pu==2.4.1 opencv-python mediapipe sklearn matplotlib
In [ ]:  # !pip install --upgrade tensorflow
```

4.1.2.2 Install and import all the dependencies like opencv, mediapipe

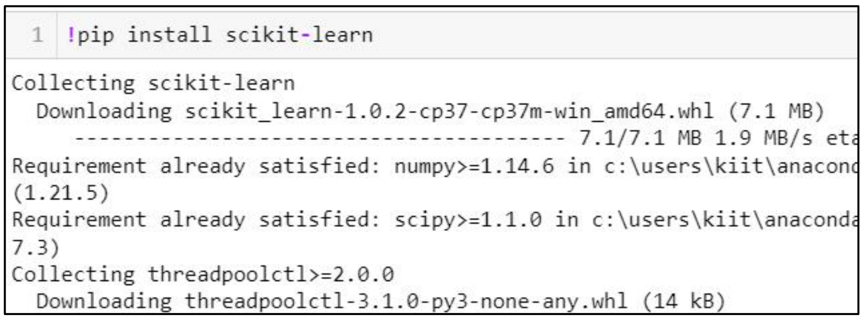


```
!pip install opencv-python --user
!pip install mediapipe
```

```
Collecting opencv-python
  Using cached opencv_python-4.7.0.72-cp37-abi3-win_amd64.whl (38.2
Requirement already satisfied: numpy>=1.17.0 in c:\users\kiit\anaco
n) (1.21.5)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.7.0.72
Collecting mediapipe
```

Fig 4.7 installing opencv and mediapipe

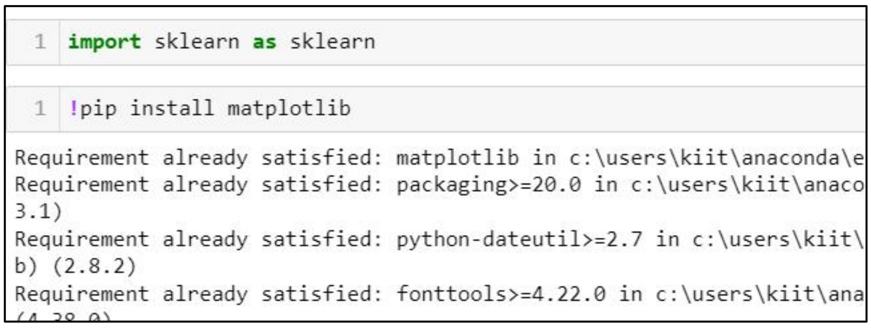
,



```
!pip install scikit-learn
```

```
Collecting scikit-learn
  Downloading scikit_learn-1.0.2-cp37-cp37m-win_amd64.whl (7.1 MB)
     ---------------------------------------- 7.1/7.1 MB 1.9 MB/s eta
Requirement already satisfied: numpy>=1.14.6 in c:\users\kiit\anacond
(1.21.5)
Requirement already satisfied: scipy>=1.1.0 in c:\users\kiit\anaconda
7.3)
Collecting threadpoolctl>=2.0.0
  Downloading threadpoolctl-3.1.0-py3-none-any.whl (14 kB)
```

Fig 4.8 install scikit-learn

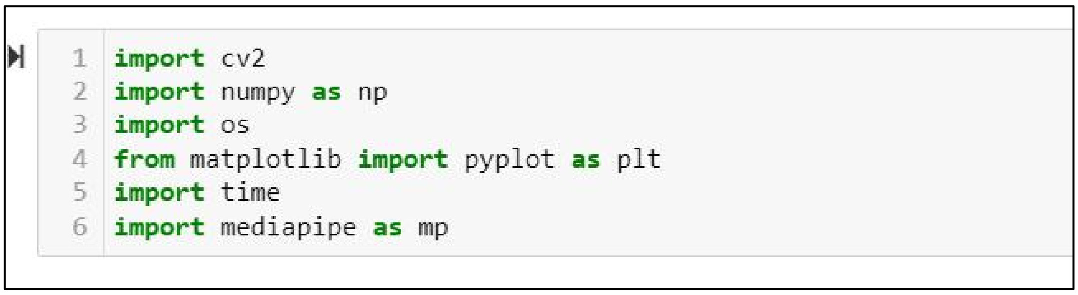


```
import sklearn as sklearn
```

```
!pip install matplotlib
```

```
Requirement already satisfied: matplotlib in c:\users\kiit\anaconda\e
Requirement already satisfied: packaging>=20.0 in c:\users\kiit\anaco
3.1)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\kiit\
b) (2.8.2)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\kiit\ana
```

Fig 4.9 install matplotlib

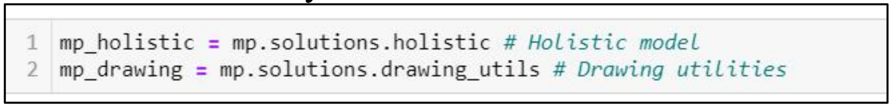


```python
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp
```

Fig 4.10 importing multiple submodules

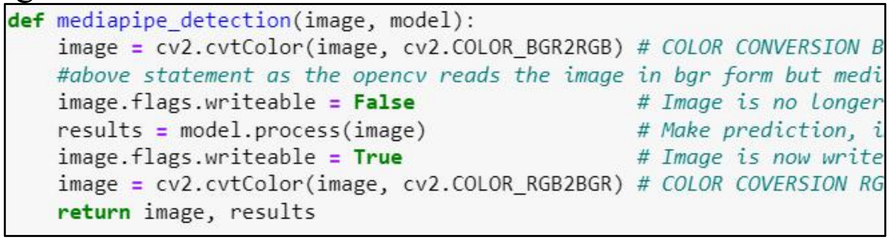
4.1.2.3 Finding the key points in the image in the real time using mp_holistic which is submodule of mediapipe which we installed in the previous step.

4.1.2.3.1 Putting the modules in separate variables so we do have to call the whole modules everytime we use them.



```python
mp_holistic = mp.solutions.holistic # Holistic model
mp_drawing = mp.solutions.drawing_utils # Drawing utilities
```

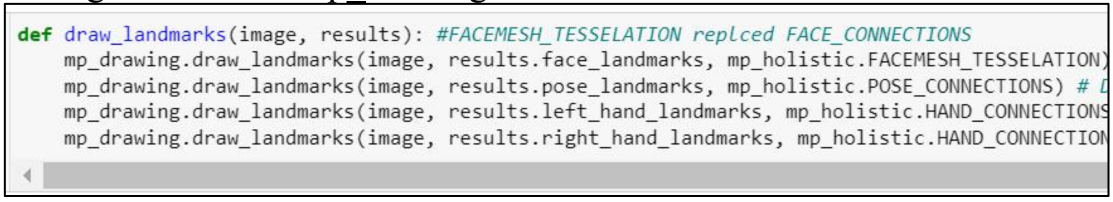
4.1.2.3.2 Making a user defined function to make the image comfortable for the medipipe module when it be used by changing the color of the image from bgr to rgb and vice versa.



```python
def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # COLOR CONVERSION B
    #above statement as the opencv reads the image in bgr form but medi
    image.flags.writeable = False                  # Image is no longer
    results = model.process(image)                 # Make prediction, i
    image.flags.writeable = True                   # Image is now write
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) # COLOR COVERSION RG
    return image, results
```

Fig 4.11 mediapipe_detection

4.1.2.3.3 Making a user defined function detect landmarks on the face, pose and hands. Taking the input of image and the processed image called result by calling the variable mp_drawing.



```python
def draw_landmarks(image, results): #FACEMESH_TESSELATION replced FACE_CONNECTIONS
    mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic.FACEMESH_TESSELATION)
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS) # D
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTION
```

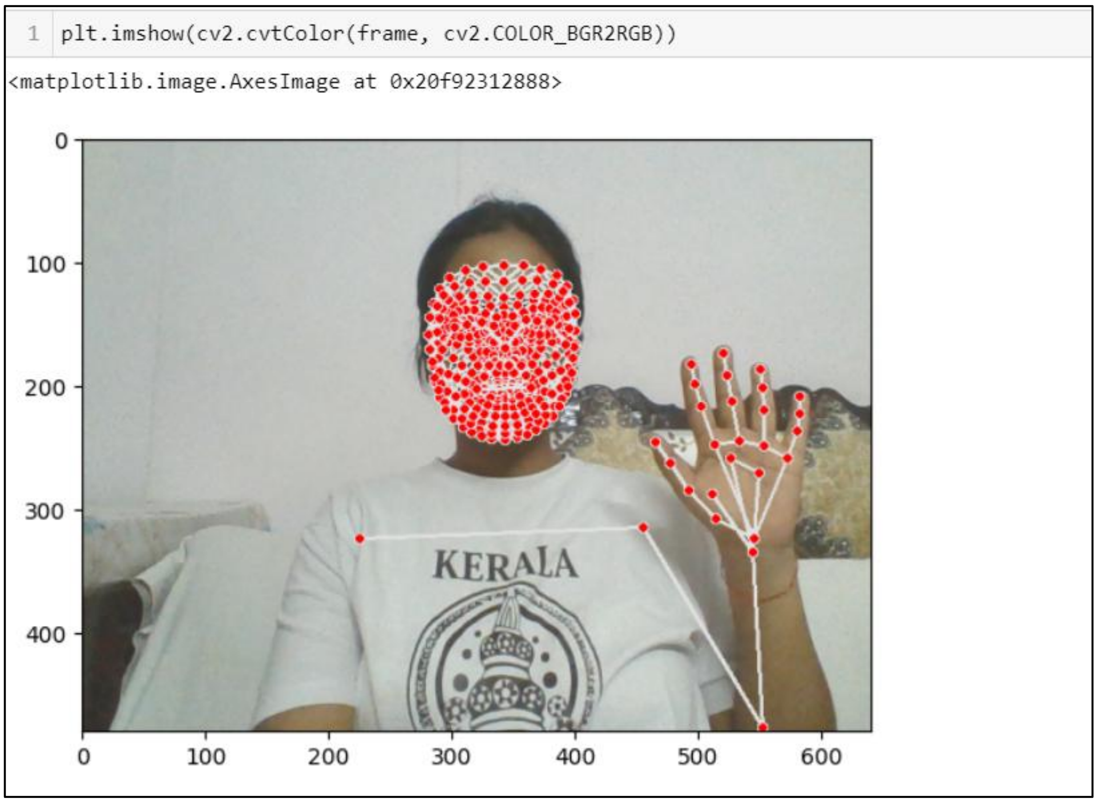
Fig 4.12 drawing landmarks

4.1.2.3.4 We can personalize how the landmark on our face and hands look.

4.1.2.3.5 We read the image through system's camera, a basic code is below

```
cap = cv2.VideoCapture(0) #opening the webcam
while cap.isOpened():#double checking the webcam is opened
    # Read feed
    ret, frame = cap.read()
    # Show to screen
    cv2.imshow('OpenCV Feed', frame)
    # Break gracefully
    if cv2.waitKey(10) & 0xFF == ord('q'): #if we press q then its going to break the loop and stop the video
        break
cap.release()
cv2.destroyAllWindows()
```

4.1.2.3.6 We record the image before the rgb and bgr conversion and after it

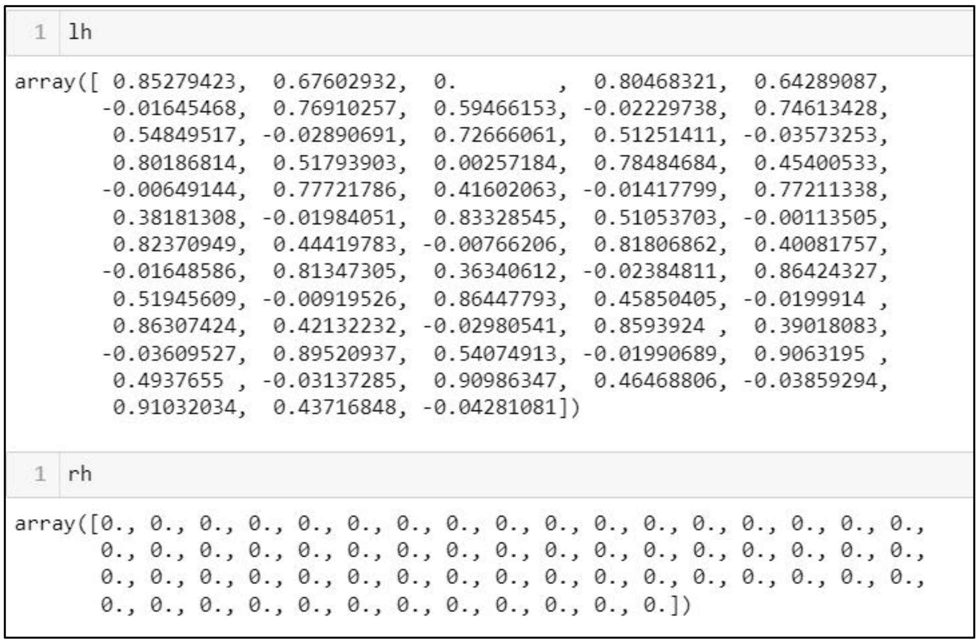


```
plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x20f92312888>
```

Fig 4.13 figure by opencv

4.1.2.4 Extract the keypoints which we detected in the previous step. And find their coordinates, and olace in in array format.



```
lh
```

```
array([ 0.85279423,  0.67602932,  0.        ,  0.80468321,  0.64289087,
       -0.01645468,  0.76910257,  0.59466153, -0.02229738,  0.74613428,
        0.54849517, -0.02890691,  0.72666061,  0.51251411, -0.03573253,
        0.80186814,  0.51793903,  0.00257184,  0.78484684,  0.45400533,
       -0.00649144,  0.77721786,  0.41602063, -0.01417799,  0.77211338,
        0.38181308, -0.01984051,  0.83328545,  0.51053703, -0.00113505,
        0.82370949,  0.44419783, -0.00766206,  0.81806862,  0.40081757,
       -0.01648586,  0.81347305,  0.36340612, -0.02384811,  0.86424327,
        0.51945609, -0.00919526,  0.86447793,  0.45850405, -0.0199914 ,
        0.86307424,  0.42132232, -0.02980541,  0.8593924 ,  0.39018083,
       -0.03609527,  0.89520937,  0.54074913, -0.01990689,  0.9063195 ,
        0.4937655 , -0.03137285,  0.90986347,  0.46468806, -0.03859294,
        0.91032034,  0.43716848, -0.04281081])
```

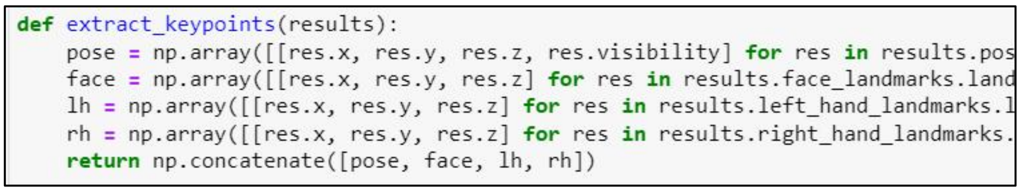
```
rh
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Fig 4.14 extraction of keypoints

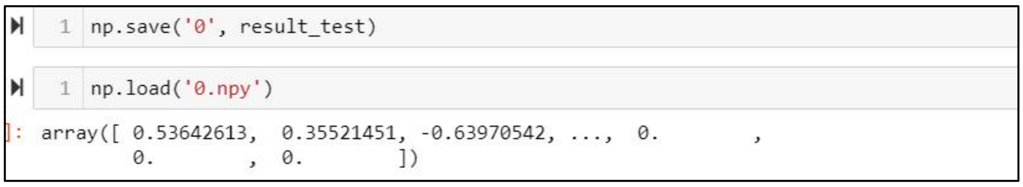
In the above screenshot we can see that in the lh which for left hand we have values but that for rh which is for right hand, all the values are zero as in the above screenshot for the point ‘4.1.2.3.6’ only the left hand is visible.

4.1.2.5 We extract all the keypoints in same array and save it in .npy file



```
def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pos
    face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.land
    lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.l
    rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.
    return np.concatenate([pose, face, lh, rh])
```

Fig 4.15 extract_keypoints

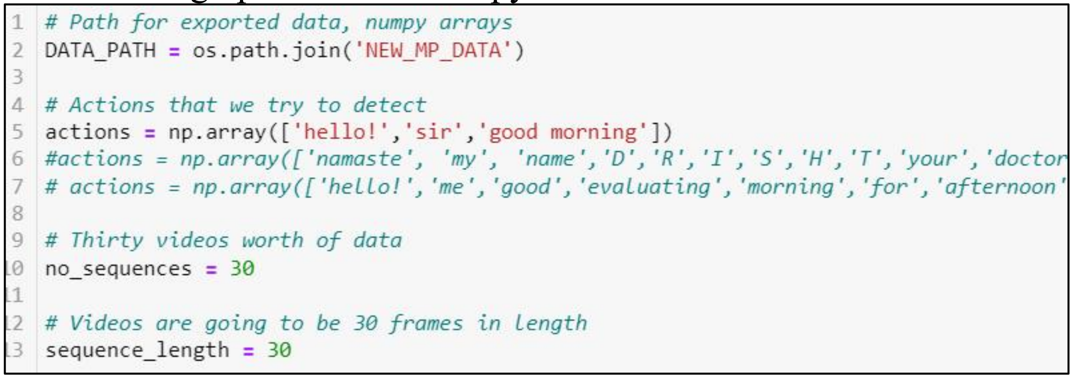


```
np.save('0', result_test)
```

```
np.load('0.npy')
```

```
]: array([ 0.53642613,  0.35521451, -0.63970542, ...,  0.        ,
         0.        ,  0.        ])
```

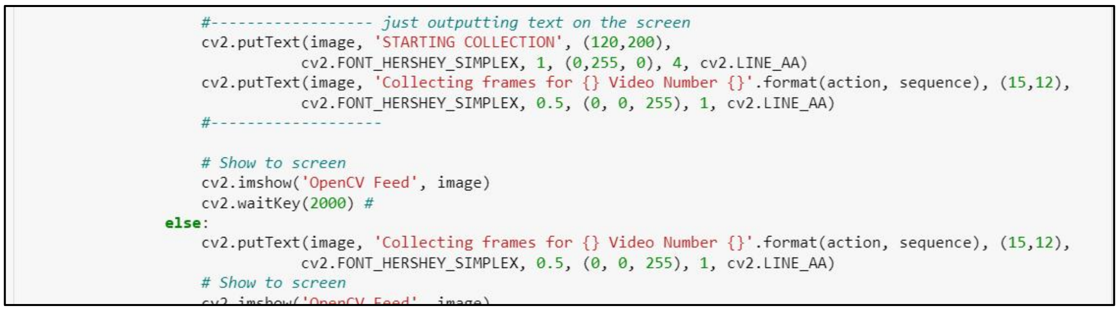
4.1.2.6 Setup the folders for collection of dataset.
As we will be making our own dataset, we will need to make up a folder in the same path where our project is situated and then write code to open the camera and record the actions, where it will be recording 30 actions and in each action it will be taking up 30 frames as .npy file.



```
# Path for exported data, numpy arrays
DATA_PATH = os.path.join('NEW_MP_DATA')

# Actions that we try to detect
actions = np.array(['hello!','sir','good morning'])
#actions = np.array(['namaste', 'my', 'name','D','R','I','S','H','T','your','doctor
# actions = np.array(['hello!','me','good','evaluating','morning','for','afternoon'

# Thirty videos worth of data
no_sequences = 30

# Videos are going to be 30 frames in length
sequence_length = 30
```

Fig 4.16 setup folders for data collection

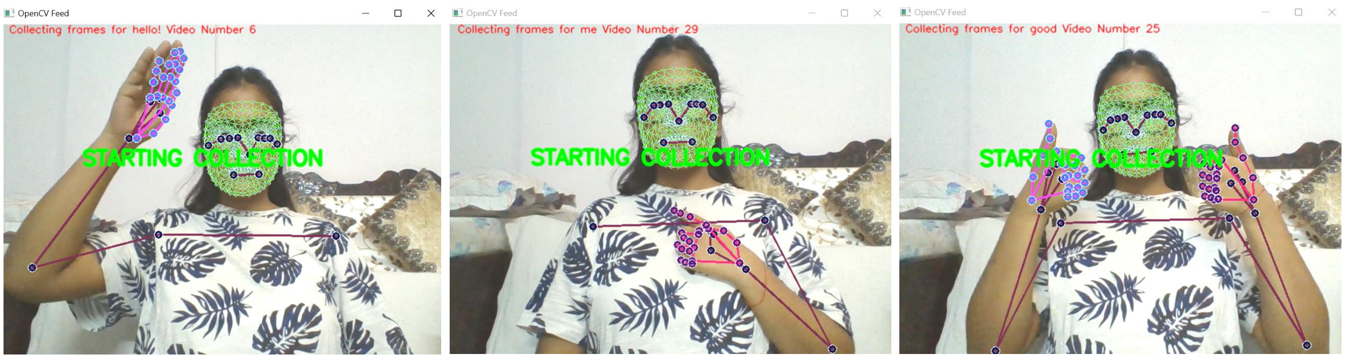
4.1.2.7 Collect keypoint values for training and testing.



```
            #----------------- just outputting text on the screen
            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)
            #------------------

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

Fig 4.17 collect keypoint values in real time

Fig 4.18 real time collection

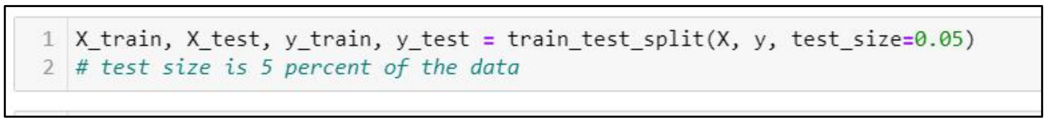

OpenCV Feed
Collecting frames for hello! Video Number 6
STARTING COLLECTION
OpenCV Feed
Collecting frames for me Video Number 29
STARTING COLLECTION
OpenCV Feed
Collecting frames for good Video Number 25
STARTING COLLECTION

4.1.2.8 Pre-process data and create labels and features

We will be importing two submodules

1. from sklearn.model_selection import train_test_split
2. from tensorflow.keras.utils import to_categorical

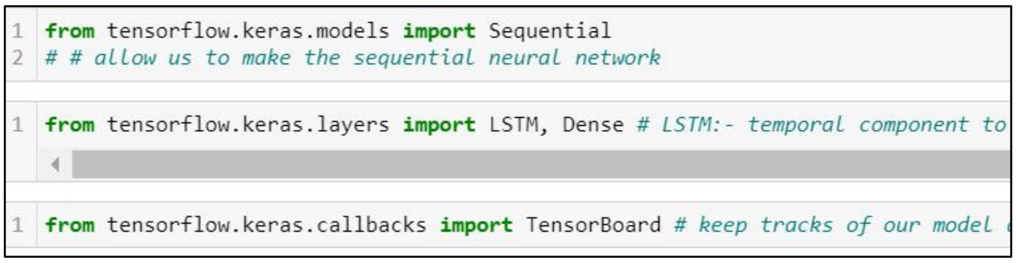
We label our actions and use one hot encoder to encode all the actions for easier manipulation and prediction in the later stages. And even do the train test split.



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
# test size is 5 percent of the data
```

4.1.2.9 Build and train the model

4.1.2.9.1 Import all the dependencies

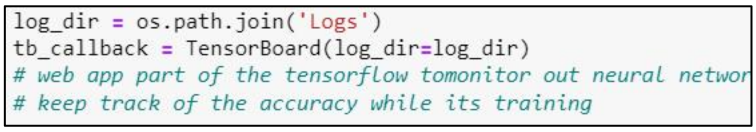


```
from tensorflow.keras.models import Sequential
# # allow us to make the sequential neural network
```

```
from tensorflow.keras.layers import LSTM, Dense # LSTM:- temporal component to
```

```
from tensorflow.keras.callbacks import TensorBoard # keep tracks of our model
```

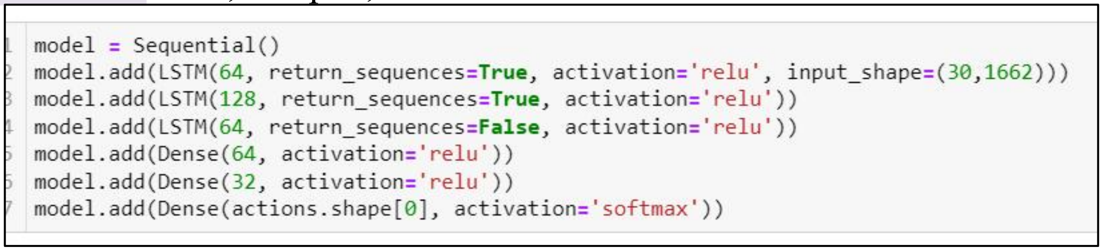
4.1.2.9.2 Make path for keeping all the logs. Logs being the info about all the epochs’ loss and accuracy and keeping a track of it in the local system.



```
log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)
# web app part of the tensorflow tomonitor out neural networ
# keep track of the accuracy while its training
```

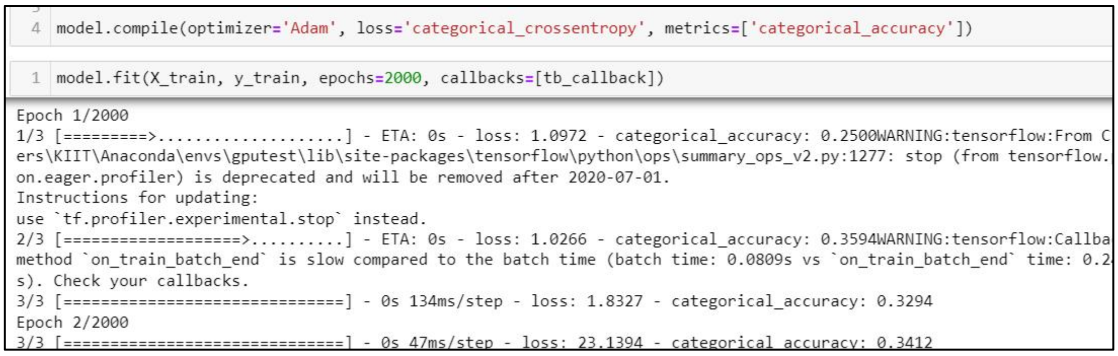
Fig 4.19 setting up log files

4.1.2.9.3 Build, compile, and fit the model



```python
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'))
```

Fig 4.20 Building the model



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

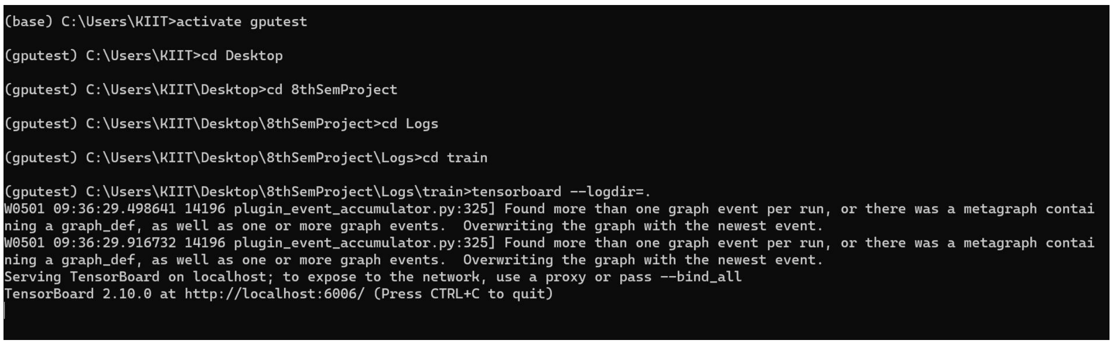
```python
model.fit(X_train, y_train, epochs=2000, callbacks=[tb_callback])
```

```
Epoch 1/2000
1/3 [=========>....................] - ETA: 0s - loss: 1.0972 - categorical_accuracy: 0.2500WARNING:tensorflow:From C
ers\KIIT\Anaconda\envs\gputest\lib\site-packages\tensorflow\python\ops\summary_ops_v2.py:1277: stop (from tensorflow.
on.eager.profiler) is deprecated and will be removed after 2020-07-01.
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
2/3 [===================>..........] - ETA: 0s - loss: 1.0266 - categorical_accuracy: 0.3594WARNING:tensorflow:Callba
method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0809s vs `on_train_batch_end` time: 0.2
s). Check your callbacks.
3/3 [==============================] - 0s 134ms/step - loss: 1.8327 - categorical_accuracy: 0.3294
Epoch 2/2000
3/3 [==============================] - 0s 47ms/step - loss: 23.1394 - categorical_accuracy: 0.3412
```

Fig 4.21 Compiling and fitting the model

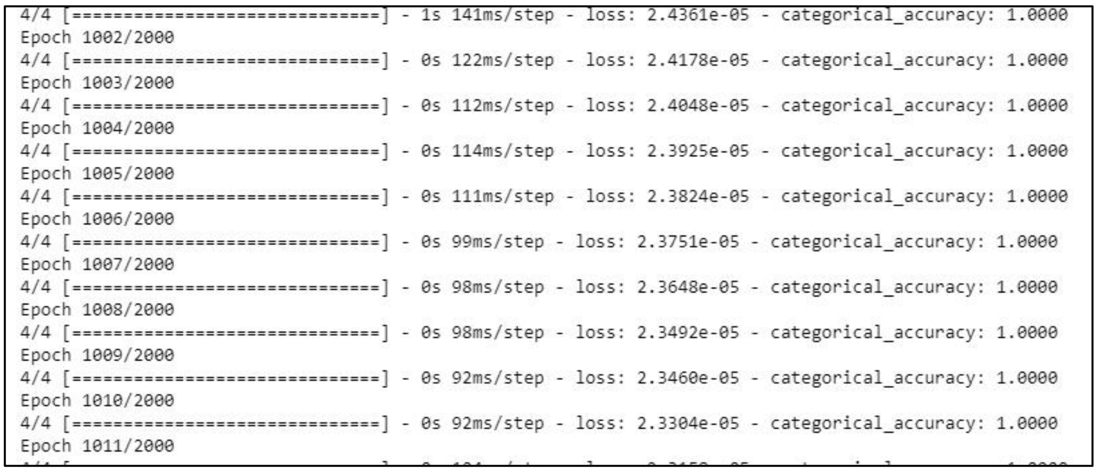
4.1.2.10 Open tensorboard and find in real time how our model is performing.

First we need to open anaconda prompt or any command prompt for that matter and activate our gpu, go into the path where our log is stored and write (tensorboard --logdir=.) which will show a link in the output, if we will paste that link in the chrome, we will be able to see the real time actio of the epoch as loss and accuracy.



```
(base) C:\Users\KIIT>activate gputest

(gputest) C:\Users\KIIT>cd Desktop

(gputest) C:\Users\KIIT\Desktop>cd 8thSemProject

(gputest) C:\Users\KIIT\Desktop\8thSemProject>cd Logs

(gputest) C:\Users\KIIT\Desktop\8thSemProject\Logs>cd train

(gputest) C:\Users\KIIT\Desktop\8thSemProject\Logs\train>tensorboard --logdir=.
W0501 09:36:29.498641 14196 plugin_event_accumulator.py:325] Found more than one graph event per run, or there was a metagraph contai
ning a graph_def, as well as one or more graph events.  Overwriting the graph with the newest event.
W0501 09:36:29.916732 14196 plugin_event_accumulator.py:325] Found more than one graph event per run, or there was a metagraph contai
ning a graph_def, as well as one or more graph events.  Overwriting the graph with the newest event.
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.10.0 at http://localhost:6006/ (Press CTRL+C to quit)
```

Fig 4.22 Opening the log folder in anaconda prompt



```
4/4 [==============================] - 1s 141ms/step - loss: 2.4361e-05 - categorical_accuracy: 1.0000
Epoch 1002/2000
4/4 [==============================] - 0s 122ms/step - loss: 2.4178e-05 - categorical_accuracy: 1.0000
Epoch 1003/2000
4/4 [==============================] - 0s 112ms/step - loss: 2.4048e-05 - categorical_accuracy: 1.0000
Epoch 1004/2000
4/4 [==============================] - 0s 114ms/step - loss: 2.3925e-05 - categorical_accuracy: 1.0000
Epoch 1005/2000
4/4 [==============================] - 0s 111ms/step - loss: 2.3824e-05 - categorical_accuracy: 1.0000
Epoch 1006/2000
4/4 [==============================] - 0s 99ms/step - loss: 2.3751e-05 - categorical_accuracy: 1.0000
Epoch 1007/2000
4/4 [==============================] - 0s 98ms/step - loss: 2.3648e-05 - categorical_accuracy: 1.0000
Epoch 1008/2000
4/4 [==============================] - 0s 98ms/step - loss: 2.3492e-05 - categorical_accuracy: 1.0000
Epoch 1009/2000
4/4 [==============================] - 0s 92ms/step - loss: 2.3460e-05 - categorical_accuracy: 1.0000
Epoch 1010/2000
4/4 [==============================] - 0s 92ms/step - loss: 2.3304e-05 - categorical_accuracy: 1.0000
Epoch 1011/2000
```

Fig 4.23 Epoch running while model fitting

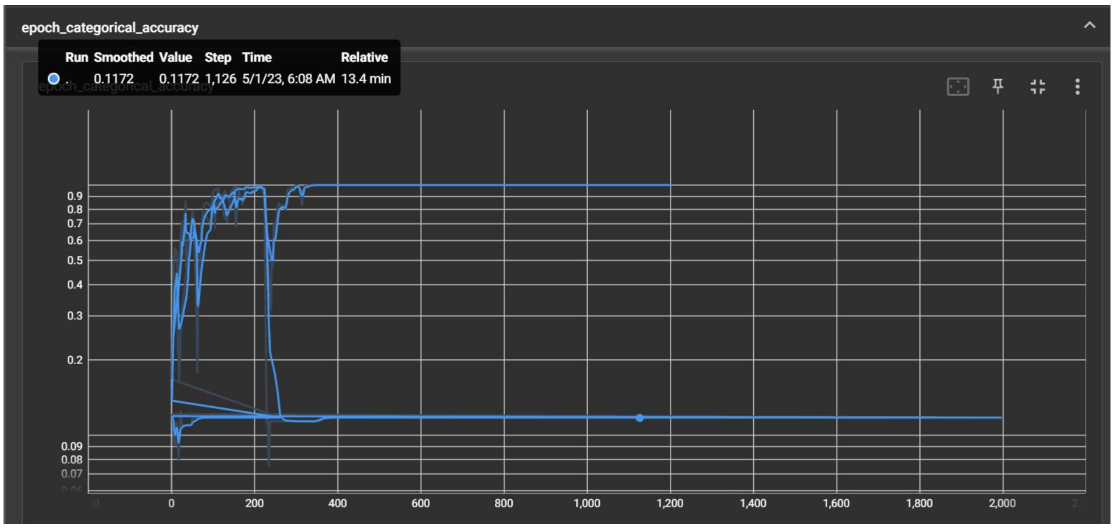


Fig 4.24
Epoch categorical accuracy

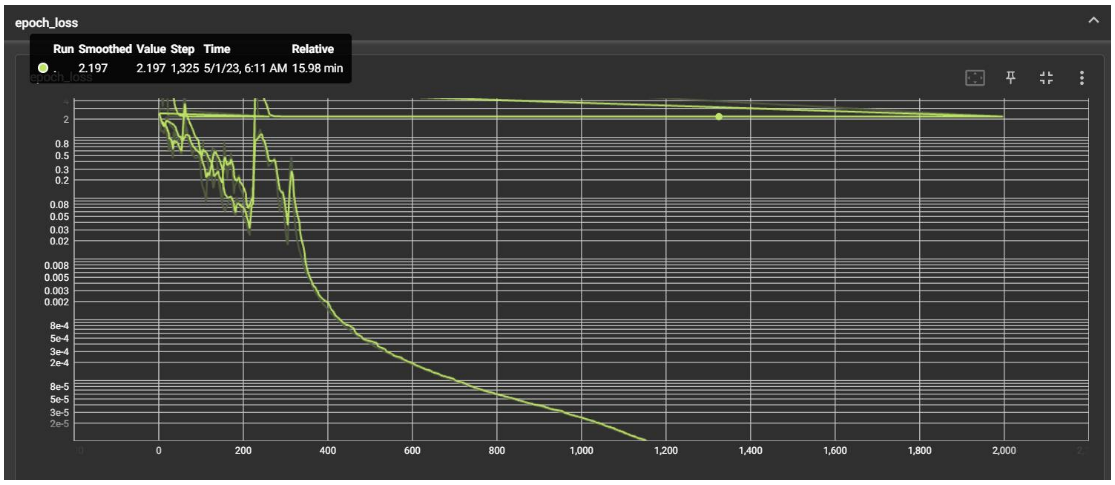
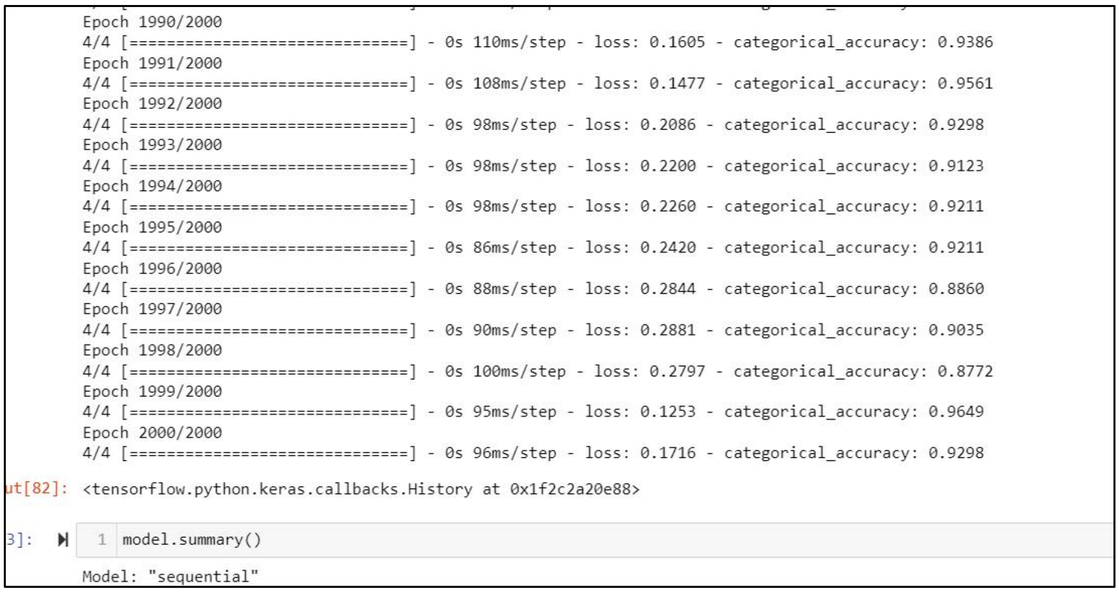


Fig 4.25
Epoch loss



```
       Epoch 1990/2000
       4/4 [==============================] - 0s 110ms/step - loss: 0.1605 - categorical_accuracy: 0.9386
       Epoch 1991/2000
       4/4 [==============================] - 0s 108ms/step - loss: 0.1477 - categorical_accuracy: 0.9561
       Epoch 1992/2000
       4/4 [==============================] - 0s 98ms/step - loss: 0.2086 - categorical_accuracy: 0.9298
       Epoch 1993/2000
       4/4 [==============================] - 0s 98ms/step - loss: 0.2200 - categorical_accuracy: 0.9123
       Epoch 1994/2000
       4/4 [==============================] - 0s 98ms/step - loss: 0.2260 - categorical_accuracy: 0.9211
       Epoch 1995/2000
       4/4 [==============================] - 0s 86ms/step - loss: 0.2420 - categorical_accuracy: 0.9211
       Epoch 1996/2000
       4/4 [==============================] - 0s 88ms/step - loss: 0.2844 - categorical_accuracy: 0.8860
       Epoch 1997/2000
       4/4 [==============================] - 0s 90ms/step - loss: 0.2881 - categorical_accuracy: 0.9035
       Epoch 1998/2000
       4/4 [==============================] - 0s 100ms/step - loss: 0.2797 - categorical_accuracy: 0.8772
       Epoch 1999/2000
       4/4 [==============================] - 0s 95ms/step - loss: 0.1253 - categorical_accuracy: 0.9649
       Epoch 2000/2000
       4/4 [==============================] - 0s 96ms/step - loss: 0.1716 - categorical_accuracy: 0.9298

ut[82]: <tensorflow.python.keras.callbacks.History at 0x1f2c2a20e88>

3]:        model.summary()

       Model: "sequential"
```

Fig 4.26
Epoch ending ending the graphs

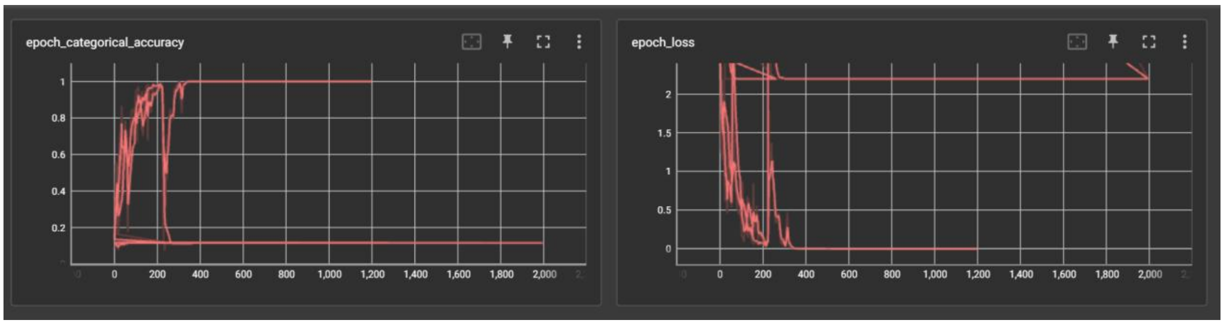


Fig 4.27 comparison graphs of accuracy and loss in epochs

4.1.2.11 After the model is trained it is time to check that where it is performing well or not, and save the weights of it. We even evaluate it using confusion matrix, find its accuracy score which in my case came 100 percent and even use model.predict.

4.1.2.12 Testing it in real time
We tweak up the previous code to record the dataset and concatenate the model into it to predict ir in real time and even show the probability with which the model is thinking. We can visualize the model working in real time using user defined function.

4.2 Testing OR Verification Plan

Test ID	Test Case Title	Test Condition	System Behavior	Expected Result
T01	Installing Keras	!pip install --upgrade --forced-reinstall keras==2.12	Error: pip's dependency resolver does not currently take into account all the source dependency conflicts	Successfully installed keras 2.12.0
T02	Import sequential	From tensorflow.keras .models import Sequentials	Pylint: disable=unused-import TypeError	Successfully imported sequential
T03	Install mediapipe	!pip install mediapipe	Python setup egg_info did not run successfully	Successfully installed mediapipe

T04	Image showing	Plt.imshow(prob_viz(res,actions,image,colors))	Prob_viz() missing one required positional argument	18x18 RGB image

4.3 Result Analysis

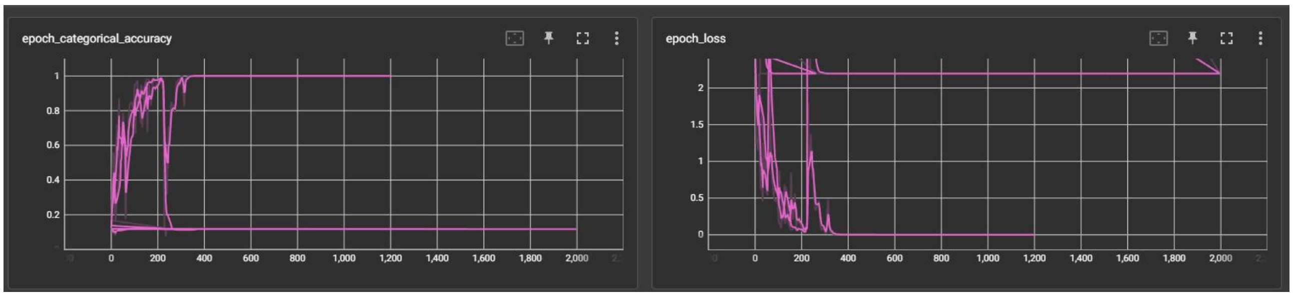
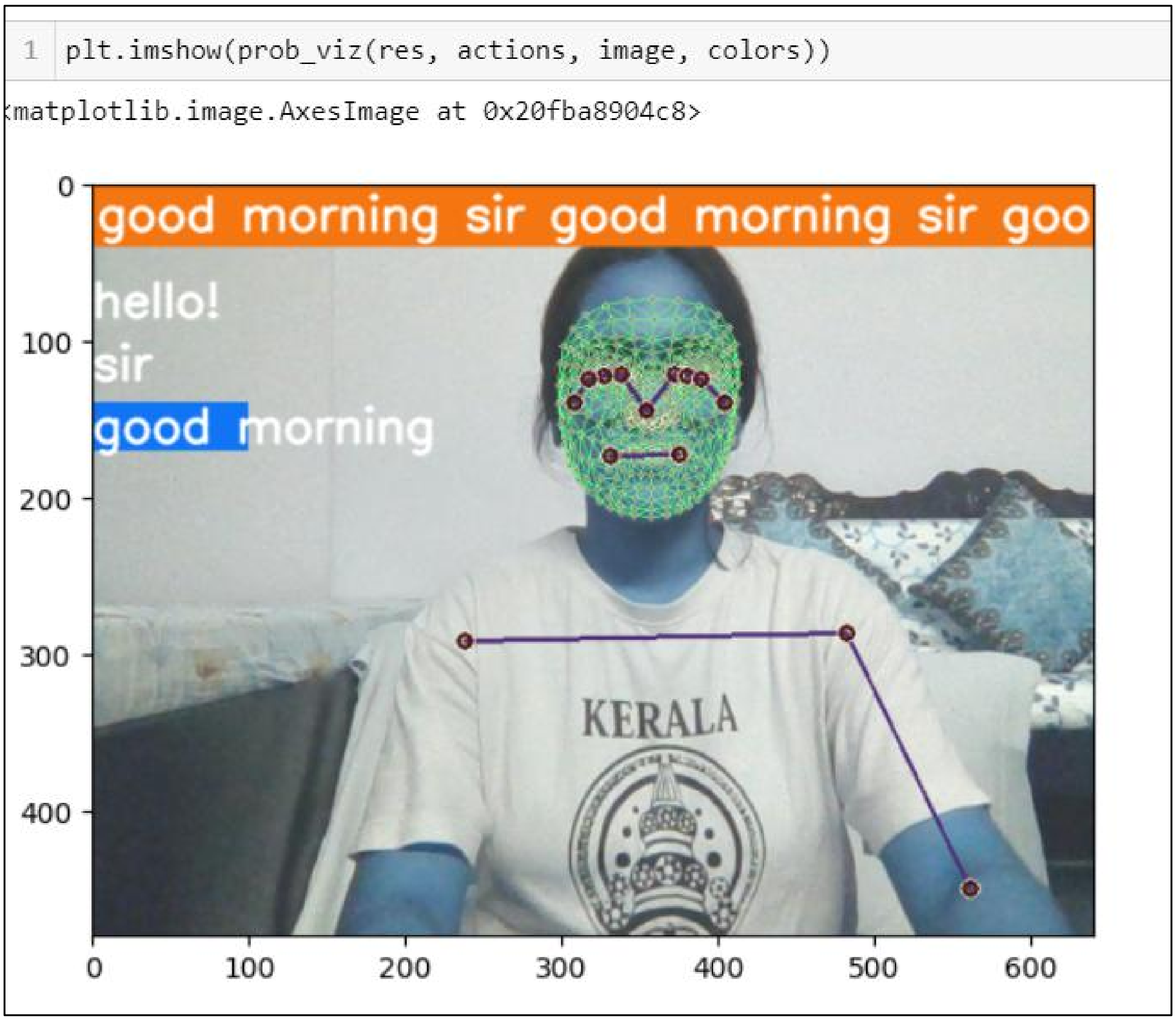


Fig 4.28 Final graphs of accuracy and loss



```
plt.imshow(prob_viz(res, actions, image, colors))
```

```
<matplotlib.image.AxesImage at 0x20fba8904c8>
```

Fig 4.29 Prediction show without rgb conversion

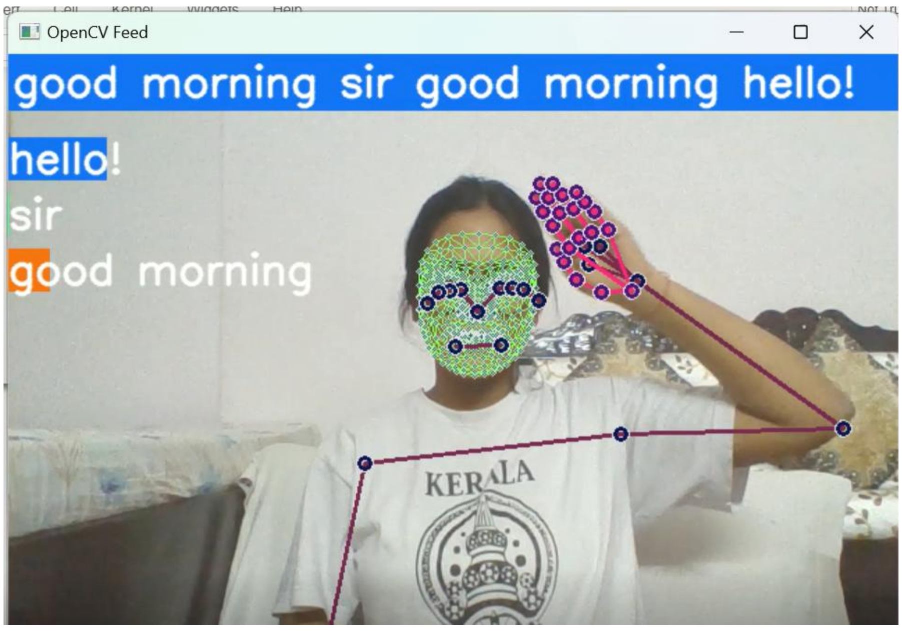


Fig 4.30
Real time prediction of action hello

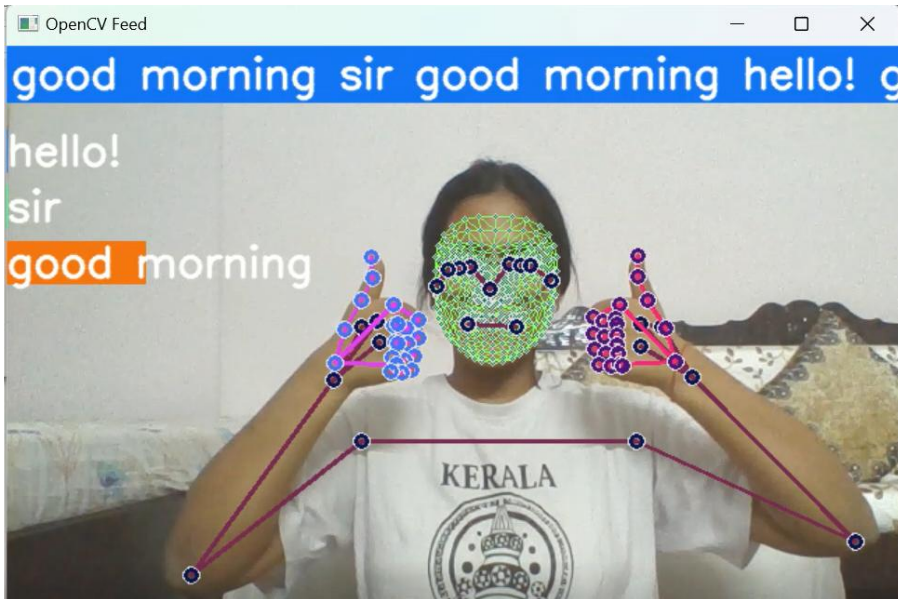


Fig 4.31
Real time prediction of action good morning

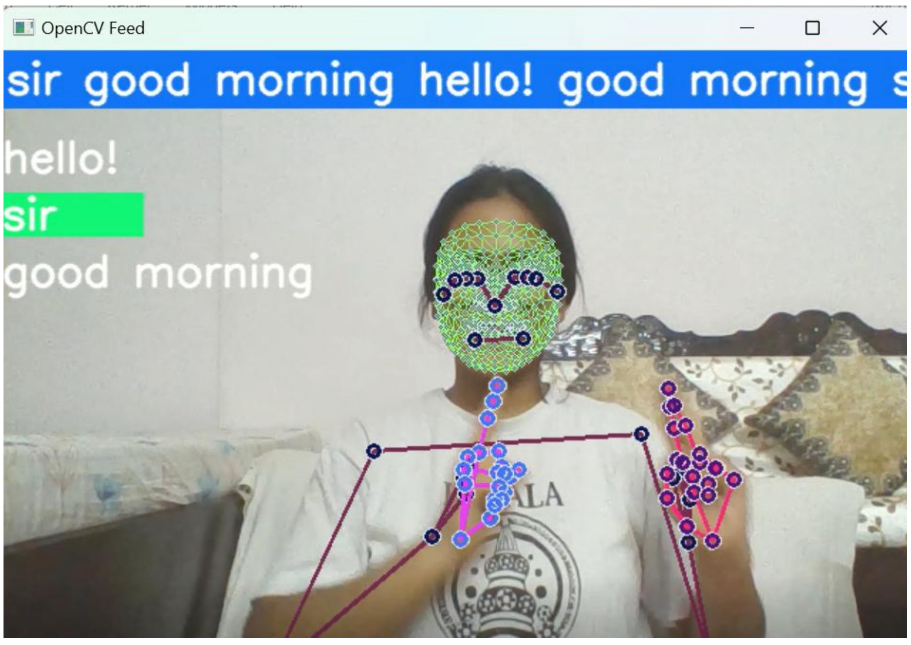


Fig 4.32
Real time prediction of action sir

Chapter 5

Conclusion and Future Scope

5.1 Conclusion

Human computer interaction has several potential uses for sign language and hand gestures, which are effective forms of human communication. When compared to conventional devices, vision-based hand gesture recognition techniques have many demonstrated advantages. Nevertheless, sign language gesture recognition is a challenging problem, and the current work only makes a small contribution to getting the outcomes required. This pproject introduced a vision-based system capable of deciphering discrete hand gestures and assisting with action-based communication. In order to forecast the activity and even indicate the likelihood, this study relied on LSTM and Dense models.

5.2 Future Scope

Lets talk about gaps and scope! some inconsistencies in the model used in earlier research by other authors were discovered during the research process. I tried to create a sequential model because we can add as many layers as possible later on without interfering with the model's functionality. This is because most models created in previous studies were done using CNN or RNN, leaving a significant gap in the field of action detection or sign language detection.
If we talk about more scope to explore in this project :

1. We can add different sign languages.
2. We can implement two way secure channel communication.
3. We can add more vocabulary.
4. We can try to deploy API using cloud to make it feasible to use.

References

[1] Tripathi, Kumud, and Neha Baranwal GC Nandi. "Continuous Indian Sign Language Gesture Recognition and Sentence Formation." Procedia Computer Science 54 (2015): 523531.

[2] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long shortterm memory." Neural computation 9, no. 8 (1997): 17351780.

[3] https://www.quora.com/WhatisanintuitiveexplanationofConvolutionalNeuralNetworks

[4]TensorFlow GPU by thehardwareguy

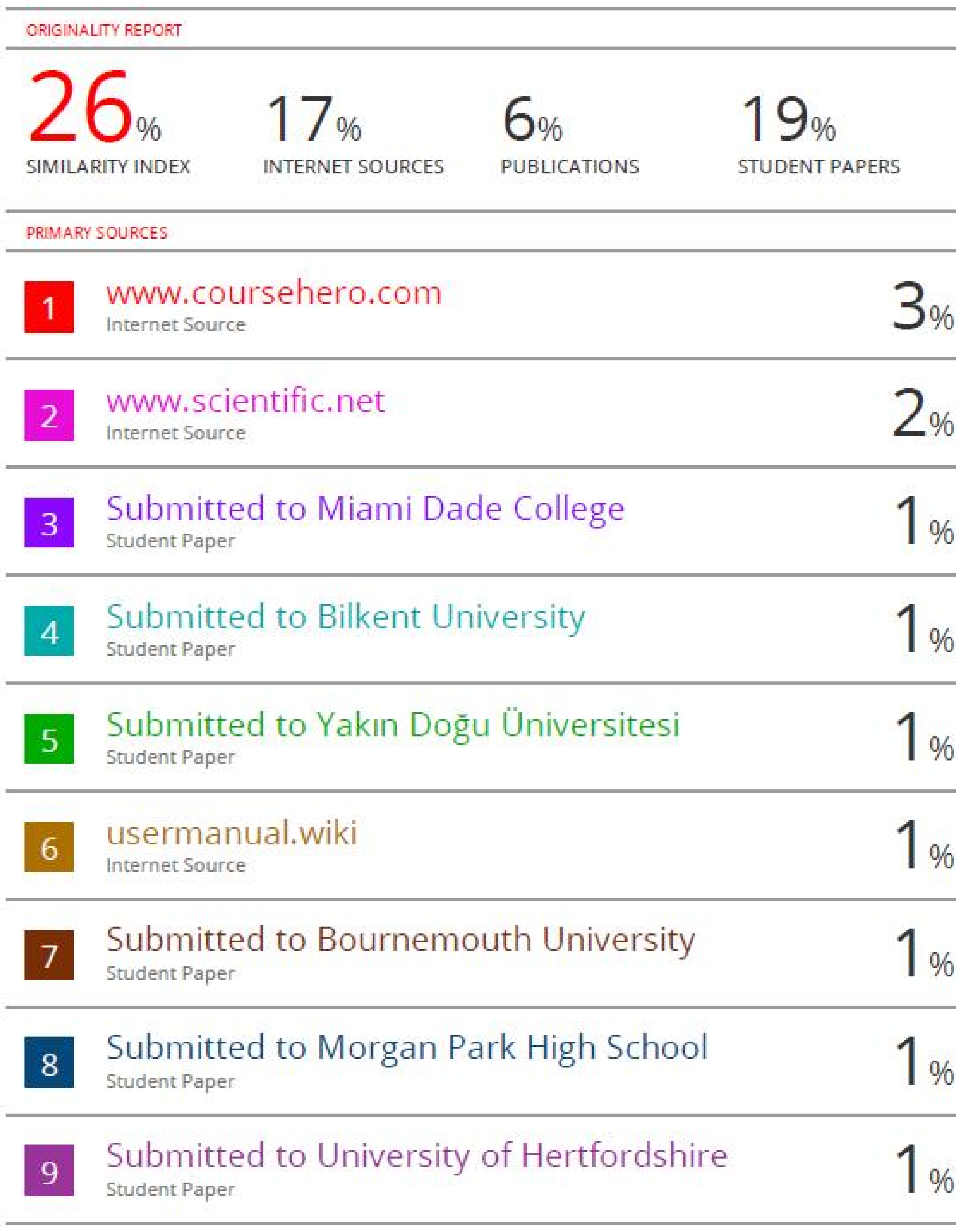
[5] Getting started in deep learning by AndrewNG

[6] Working with LSTM models by freecodecamp

[7] Liu, T., Zhou, W., & Li, H. (2016, September). Sign language recognition with long short-term memory. In 2016 IEEE international conference on image processing (ICIP) (pp. 2871-2875). IEEE.

[8] Abraham, E., Nayak, A., & Iqbal, A. (2019, October). Real-time translation of Indian sign language using LSTM. In 2019 global conference for advancement in technology (GCAT) (pp. 1-5). IEEE.

Plagiarism check



ORIGINALITY REPORT

26%	17%	6%	19%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.coursehero.com Internet Source	3%
2	www.scientific.net Internet Source	2%
3	Submitted to Miami Dade College Student Paper	1%
4	Submitted to Bilkent University Student Paper	1%
5	Submitted to Yakın Doğu Üniversitesi Student Paper	1%
6	usermanual.wiki Internet Source	1%
7	Submitted to Bournemouth University Student Paper	1%
8	Submitted to Morgan Park High School Student Paper	1%
9	Submitted to University of Hertfordshire Student Paper	1%