Machine Learning Lab

T.Y. B.Tech
Computer Engg.
Sem VI

Group 18
**Project Report**

# Real Time Sign Language detection

## Overview

A practical implementation of sign language estimation using an LSTM NN built on TF Keras.

## Goals - Real time sign language detection using sequences.

1. Extract holistic keypoints.
2. Train a LSTM DL model.
3. Make real-time predictions using sequences.

## Introduction

Sign Language significantly facilitates communication in the deaf community. Sign language is a language in which communication is based on visual sign patterns to express one's feelings. There is a communication gap when a deaf community wants to express their views, thoughts of speech and hearing with normal people. Currently, two communities mostly rely on human-based translators which can be expensive and inconvenient. With the development in areas of deep learning and computer vision, researchers have developed various automatic sign language recognition methods that can interpret sign gestures in an understandable way. This narrow downs the communication gap between impaired and normal people. This also empowers deaf-mute people to stand with an equal opportunity and improve personal growth.

According to the report of the World Federation of the Deaf (WFD) over 5% of the world's population (≈ 360 million people) has hearing impairment including 328 million adults and 32 Million children. Approximately 300 sign languages are in use around the globe. Sign language recognition is a challenging task as sign language alphabets are different for different sign languages. For instance, American Sign Language (ASL) alphabets vary widely from Indian Sign Language or Italian Sign Language. Thus Sign language varies from region to region. Moreover, articulation of single as well as double hands is used to convey meaningful messages. Sign Language can be expressed by the compressed version, where a single gesture is sufficient to describe a word.

So , In our project we have trained our ML model with modified sign languages for certain words, which can eventually predict the words using OpenCV in Real Time.

## How does it work?

1. Collect key points from mediapipe holistic.

2. Train a deep neural network with LSTM layers for sequences.

3. Perform Real time sign language detection using OpenCV

# Libraries/Dependencies used :

1. Tensorflow

TensorFlow is an open-source library developed by Google primarily for deep learning applications. It also supports traditional machine learning. TensorFlow was originally developed for large numerical computations without keeping deep learning in mind. However, it proved to be very useful for deep learning development as well, and therefore Google open-sourced it.

TensorFlow accepts data in the form of multi-dimensional arrays of higher dimensions called tensors. Multi-dimensional arrays are very handy in handling large amounts of data.

TensorFlow works on the basis of data flow graphs that have nodes and edges. As the execution mechanism is in the form of graphs, it is much easier to execute TensorFlow code in a distributed manner across a cluster of computers while using GPUs.

2. OpenCV :

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

The first OpenCV version was 1.0. OpenCV is released under a BSD license and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. When OpenCV was

designed the main focus was real-time applications for computational efficiency. All things are written in optimized C/C++ to take advantage of multi-core processing.

### 3. Sklearn :

Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms. It's built upon some of the technology you might already be familiar with, like NumPy, pandas, and Matplotlib!

The functionality that scikit-learn provides include:

a. Regression, including Linear and Logistic Regression
b. Classification, including K-Nearest Neighbors
c. Clustering, including K-Means and K-Means++
d. Model selection
e. Preprocessing, including Min-Max Normalization

### 4. Matplotlib :

Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open source alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications.

A Python matplotlib script is structured so that a few lines of code are all that is required in most instances to generate a visual data plot. The matplotlib scripting layer overlays two APIs:

The pyplot API is a hierarchy of Python code objects topped by matplotlib.pyplot

An OO (Object-Oriented) API collection of objects that can be assembled with greater flexibility than pyplot. This API provides direct access to Matplotlib's backend layers.

## 5. Mediapipe :

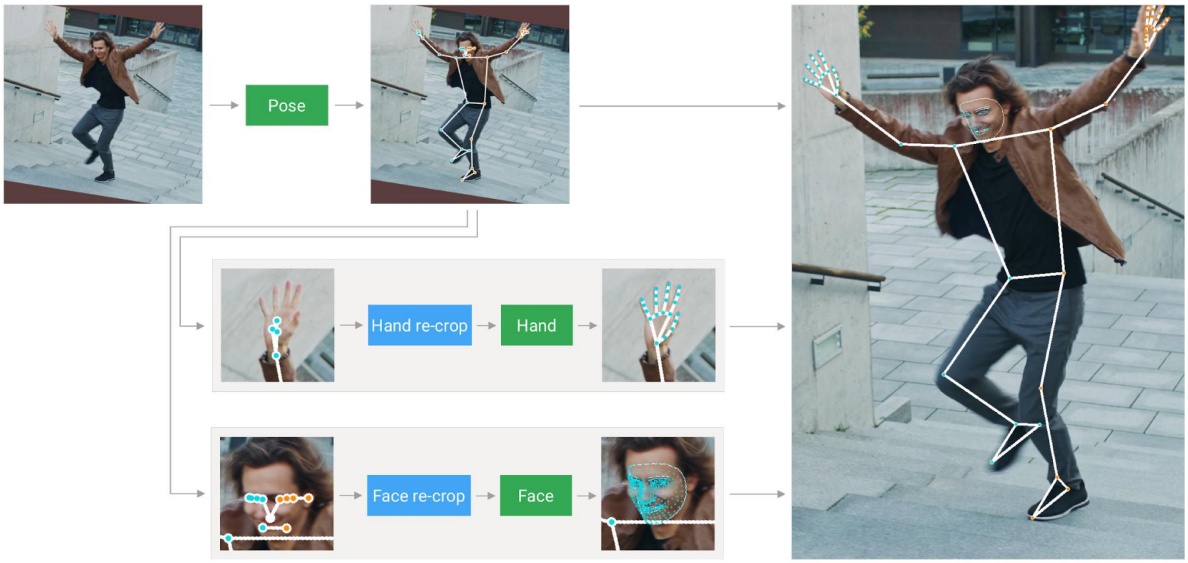
ML Pipeline

The MediaPipe Holistic pipeline integrates separate models for pose, face and hand components, each of which are optimized for their particular domain. However, because of their different specializations, the input to one component is not well-suited for the others. The pose estimation model, for example, takes a lower, fixed resolution video frame (256x256) as input. But if one were to crop the hand and face regions from that image to pass to their respective models, the image resolution would be too low for accurate articulation. Therefore, we designed MediaPipe Holistic as a multi-stage pipeline, which treats the different regions using a region appropriate image resolution.

First, we estimate the human pose (top of Fig 2) with BlazePose's pose detector and subsequent landmark model. Then, using the inferred pose landmarks we derive three regions of interest (ROI) crops for each hand (2x) and the face, and employ a re-crop model to improve the ROI. We then crop the full-resolution input frame to these ROIs and apply task-specific face and hand models to estimate their corresponding landmarks. Finally, we merge all landmarks with those of the pose model to yield the full 540+ landmarks.

To streamline the identification of ROIs for face and hands, we utilize a tracking approach similar to the one we use for standalone face and hand pipelines. It assumes that the object doesn't move significantly between frames and uses estimation from the previous frame as a guide to the object region on the current one. However, during fast movements, the tracker can lose the target, which requires the detector to re-localize it in the image. MediaPipe Holistic uses pose prediction (on every frame) as an additional ROI prior to reduce the response time of the pipeline when reacting to

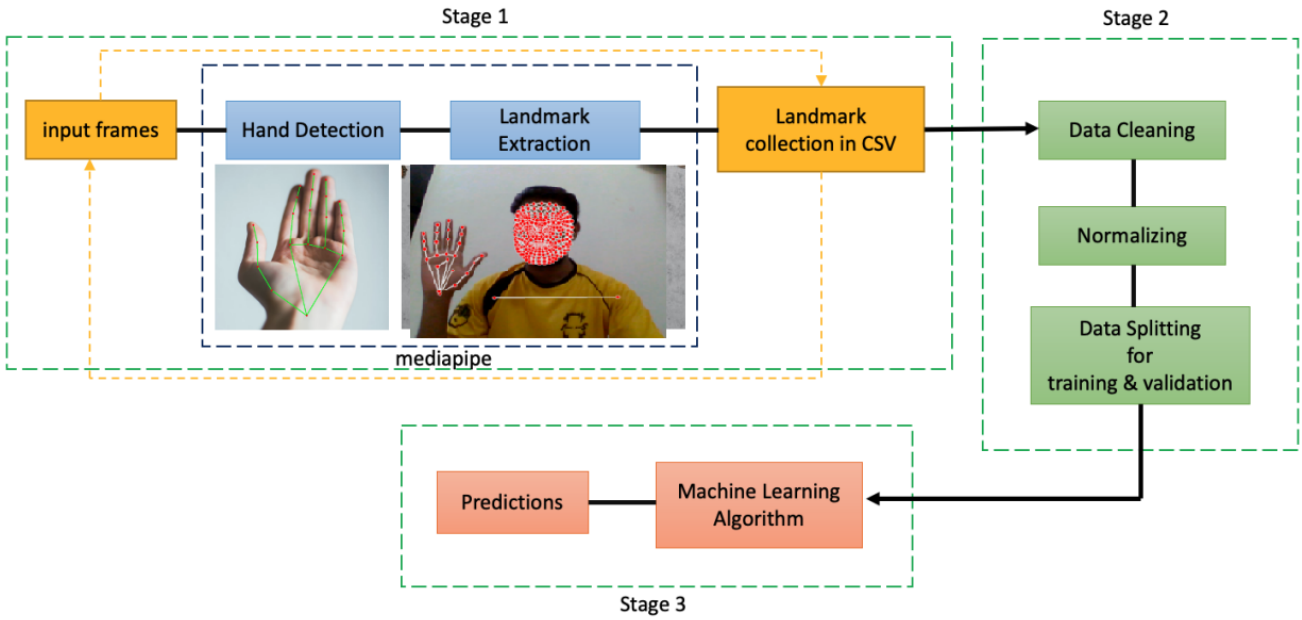
fast movements. This also enables the model to retain semantic consistency across the body and its parts by preventing a mixup between left and right hands or body parts of one person in the frame with another.




In addition, the resolution of the input frame to the pose model is low enough that the resulting ROIs for face and hands are still too inaccurate to guide the re-cropping of those regions, which require a precise input crop to remain lightweight. To close this accuracy gap we use lightweight face and hand re-crop models that play the role of spatial transformers and cost only ~10% of corresponding model's inference time.

The pipeline is implemented as a MediaPipe graph that uses a holistic landmark subgraph from the holistic landmark module and renders using a dedicated holistic renderer subgraph. The holistic landmark subgraph internally uses a pose landmark module , hand landmark module and face landmark module.

# Architecture :



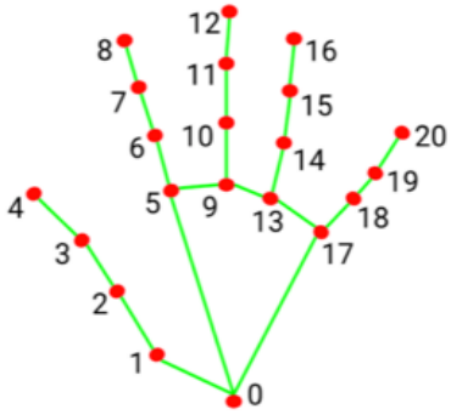

1.1 Stage 1: Pre-Processing of Images to get Landmarks using MediaPipe

MediaPipe is a framework that enables developers for building multi-modal(video, audio, any times series data) cross-platform applied ML pipelines. MediaPipe has a large collection of human body detection and tracking models which are trained on a massive and most diverse dataset of Google. As the skeleton of nodes and edges or landmarks, they track key points on different parts of the body. All coordinate points are three-dimension normalized. Models built by Google developers using Tensorflow lite facilitates the flow of information easily adaptable and modifiable via graphs. MediaPipe

pipelines are composed of nodes on a graph which are generally specified in a pbtxt file. These nodes are connected to C++ files. Expansion upon these files is the base calculator class in Mediapipe. Just like a video stream this class gets contracts of media

streams from other nodes in the graph and ensures that it is connected. Once, rest of the pipeline nodes are connected, the class generates its own output processed data. Packet objects encapsulating many different types of information are used to send each stream of information to each calculator. Into a graph, side packets can also be imposed, where a calculator node can be introduced with auxiliary data like constants or static properties. This simplified structure in the pipeline of dataflow enables additions or modifications with ease and the flow of data becomes more precisely controllable.

The Hand tracking solution [13] has an ML pipeline at its backend consisting of two models working dependently with each other: a) Palm Detection Model b) Land Landmark Model. The Palm Detection Model provides an accurately cropped palm image and further is passed on to the landmark model. This process diminishes the use of data augmentation (i.e. Rotations, Flipping, Scaling) that is done in Deep Learning models and dedicates most of its power for landmark localization. The traditional way is to detect the hand from the frame and then do landmark localization over the current frame. But in this Palm Detector using ML pipeline challenges with a different strategy. Detecting hands is a complex procedure as you have to perform image processing and thresholding and work with a variety of hand sizes which leads to consumption of time. Instead of directly detecting the hand from the current frame, first, the Palm detector is trained which estimates bounding boxes around the rigid objects like palm and fists which is simpler than detecting hands with coupled fingers. Secondly, an encoder-decoder is used as an extractor for bigger scene context.




0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

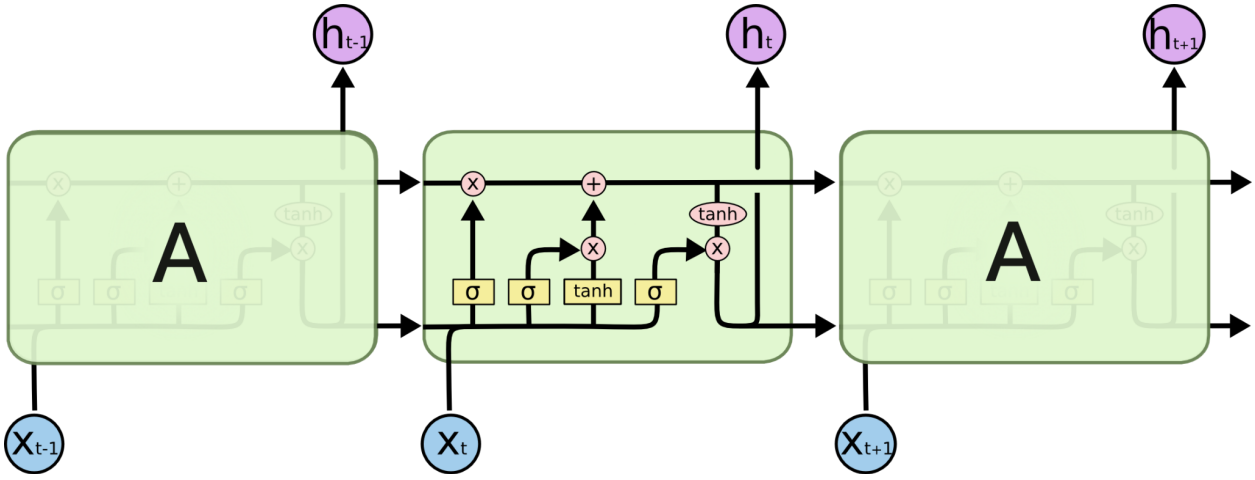
After the palm detection is skimmed over the whole image frame, subsequent Hand Landmark models come into the picture. This model precisely localizes 21 3D hand-knuckle coordinates (i.e., x, y, z-axis) inside the detected hand regions. The model is so well trained and robust in hand detection that it even maps coordinates to partially visible hands. Figure 2 shows the 21 landmark points detected by the Hand Landmark model. Now that we have a functional Palm and Hand detection model running, this model is passed over our dataset of various languages.

## Prediction using Machine Learning Algorithm :

### Long-Short-Term memory (LSTM)

Convolutional neural networks are great for a 1 to 1 relation; given an image of a sign, it generates fixed-size labels, like the class of the sign in the image. However, What CNNs cannot do is accept a sequence of vectors. That's where Recurrent Neural Networks (RNNs) are used. RNNs allow us to understand the context of a video frame, relative to the frames that came before it. They do this by passing the output of one training step to the input of the next training step, along with the new frame. We're using a special type of RNN here, called an LSTM, that allows our network to learn long term dependencies.

Figure below shows a visualization for an LSTM cell.




continuous word recognition with Mediapipe - LSTM architecture.

## Evaluation and Accuracy :

## Evaluation using Confusion Matrix and Accuracy

```
1 from sklearn.metrics import multilabel_confusion_matrix, accuracy_score
```

```
1 yhat = model.predict(X_test)
```

```
1 ytrue = np.argmax(y_test, axis=1).tolist()
2 yhat = np.argmax(yhat, axis=1).tolist()
```

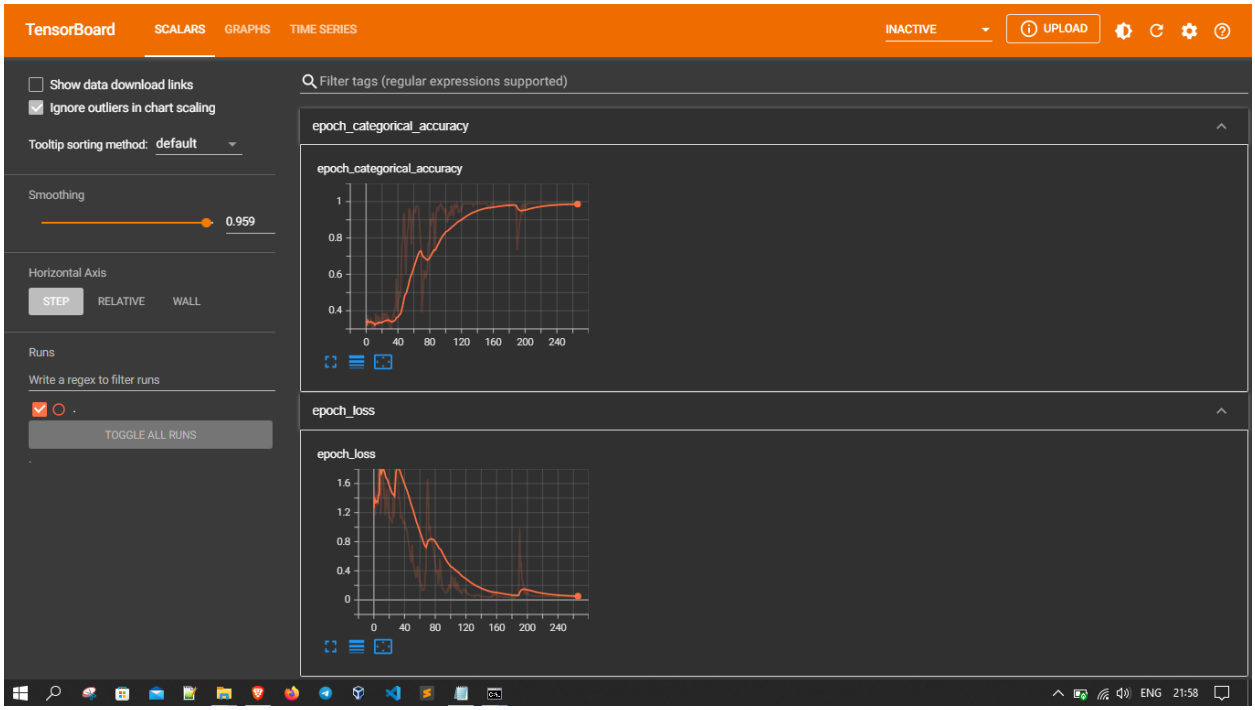
```
1 multilabel_confusion_matrix(ytrue, yhat)
```

```
array([[[3, 0],
        [0, 2]],

       [[3, 0],
        [0, 2]],

       [[4, 0],
        [0, 1]]], dtype=int64)
```

```
1 accuracy_score(ytrue, yhat)
```

```
1.0
```

Tensorboard :


TensorBoard
SCALARS
GRAPHS
TIME SERIES
INACTIVE
UPLOAD
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.959
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
Filter tags (regular expressions supported)
epoch_categorical_accuracy
epoch_categorical_accuracy
1
0.8
0.6
0.4
0 40 80 120 160 200 240
epoch_loss
epoch_loss
1.6
1.2
0.8
0.4
0
0 40 80 120 160 200 240
ENG 21:58

Output :

Real-time detection :