Proposed Title: Detection Of Hand Signals for Traffic Control Using Deep Learning MoveNet Model

Development Model:

Different Traffic Gesture - Pose detection

Group Members

Reg.No	Name
20BCI0090	Vandit Gabani
20BCI0128	Aditi Nitin Tagalpallewar
20BCI0176	Yash Bobde
20BCI0271	Harsh Rajpal
20BCE2759	Payal Maheshwari
20BCI0138	Bagade Shaunak Rahul
20BCI0169	Konark Patel

Reg.No	Name
20BCI0159	Nikhil Harshwardhan

Roles - Responsibilities :

1.Collecting Data and Extracting Landmarks points and finding Angles for Different pose Using Mediapipe and Movenet Model -
Konark Patel(20BCI0169) , Nikhil(20BCI0159)

2.Build and train a pose classification model that takes landmark coordinates from a CSV file as input and outputs predicted labels-
Vandit Gabani(20BCI0090) ,Aditi(20BCI0128) , Shaunak(20BCI0138)

3.Convert the pose classification model to TFLite and Testing Using Movenet Model-
Harsh(20BCI0271) , Payal(20BCE2759) , Yash(20BCI0176)

Abstract

- Gesture recognition is one of the most difficult challenges in computer vision. While recognising traffic police hand signals, one must take into account the speed and dependability of the instructing signal. It is significantly easier to extract the three-dimensional coordinates of skeletons when depth information is given with the photos. Here, we present a method for detecting hand signals that does not rely on skeletons. Instead of skeletons, we employ basic object detectors that have been trained to respond to hand signals.
- Autonomous vehicles require traffic police gesture recognition. Current traffic police gesture identification systems frequently extract pixel-level characteristics from RGB photos, which are incoherent owing to the absence of gesture skeleton features and can result in erroneous results. Existing object detection algorithms enable the detection of automobiles, trees, people, bicycles, animals, and so forth (YOLO).
- In this project, we will employ the Convolutional Neural Network (CNN) approach (Deep Learning) to recognise traffic police hand signals. As there are no acceptable datasets available, we shall attempt to generate our own. Mediapipe will be utilised in its development.

Introduction

- Artificial intelligence has been implemented in various industries, particularly in computer vision, improving object detection and classification. Traffic signals play a significant role in traffic flow and safety, and AI can be used to enhance their effectiveness.
- Recognizing traffic police gestures is challenging due to the lack of interpretable features from RGB images and interference from complex environments. Gesture skeleton extractor (GSE) can be used to extract interpretable skeleton coordinate information, eliminating background interference. However, existing deep learning methods are not suitable for handling gesture skeleton features, requiring the development of new methods.
- Previous studies faced limitations due to the reliance on handcrafted components or pixel-level information, resulting in poor recognition performance and weak generalization capability. Autonomous vehicles must also recognize hand signals from traffic controllers in real-time, which can be challenging due to the high speed of the vehicles. Many skeleton-based methods require high computational load or expensive devices, making them unsuitable for real-time processing.
- Prior approaches used skeleton-based action recognition to identify hand signals in videos, requiring preprocessing and limiting generalizability to real-world problems. Accurate detection of hand signals requires distinguishing between intentional and non-intentional signals. The ability to differentiate situations affects the accuracy of detection.
- The paper proposes a new method to recognize hand signals from raw video streams without preprocessing, overcoming the challenges of previous methods that relied on skeleton-based action recognition. The proposed method utilizes an attention-based spatial-temporal graph convolutional network (ASTGCN) that achieved higher accuracy than previous methods and can distinguish between intentional and non-intentional signals. The potential applications of this method include traffic management, public safety, and military operations. The paper highlights the significance of deep learning in recognizing hand signals in real-world environments.

Division Of Work:

Our Project Will be Divided into 3 parts:

Part 1:
DATASET Collection and Preprocess the pose classification training data into a CSV file,specifying the landmarks (body key points) and ground truth pose labelsrecognized by the Mediapipe and MoveNet.
Konark Patel(20BCI0169) , Nikhil(20BCI0159)

- There are various processes involved in preparing pose classification training data into a CSV file, and they will be distributed among the team members.
- Together, we will check the annotated data for accuracy and consistency, as well as the main points that were extracted and the CSV file that was produced.

- To make the training data larger, we will also take into account data augmentation methods including flipping, rotating, and scaling.
- Ultimately, the entire team will finish the preparation of posture categorization training data into a CSV file, which is an essential step in training a pose estimate model.

Part 2:
Build and train a pose classification model(MoveNet Model) that takes landmark coordinates from a CSV file as input and outputs predicted labels.
Vandit Gabani(20BCI0090) ,Aditi(20BCI0128) , Shaunak(20BCI0138)

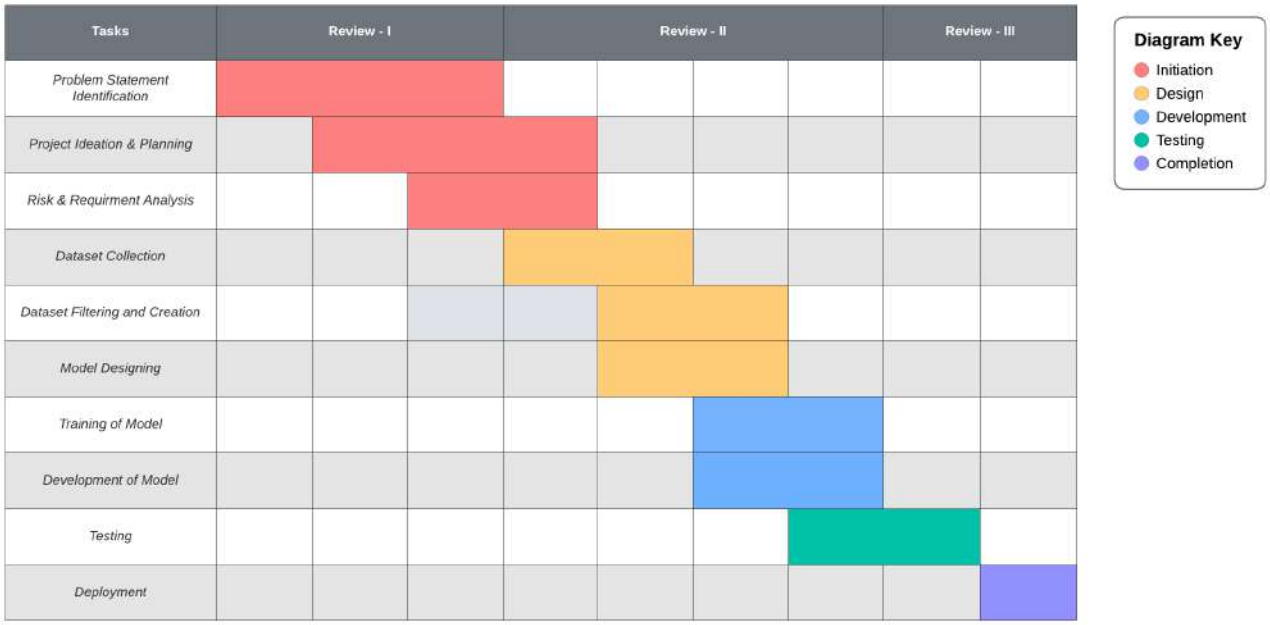
- There are various processes involved in creating and refining a pose categorization model, and they will be distributed among the team members.
- To ensure that the model architecture and training parameters are properly stated, everyone will work together to construct and train the model.
- They will also cooperate to assess the model's performance on the validation set and make any required modifications to increase its accuracy.
- In general, developing and training a pose classification model necessitates a solid grasp of computer vision and deep learning, as well as a methodical approach to model choice, implementation, and training.

Part 3:
Convert the pose classification model to TFLite and Test and Deployment of Model.
Harsh(20BCI0271) , Payal(20BCE2759) , Yash(20BCI0176)

- There are various processes involved in converting a pose classification model to TFLite, which may be broken down into jobs for each team member.
- To guarantee that the conversion procedure goes without a hitch and that the TFLite model is appropriately optimised and validated, everyone will cooperate.
- Overview: We will utilise our own dataset, which consists of 7 distinct traffic police/pose photographs, as there isn't a dataset readily available online.
- Our dataset consists of almost 5000–6000 unique photos of the crucial 7 traffic postures that were narrowed down.

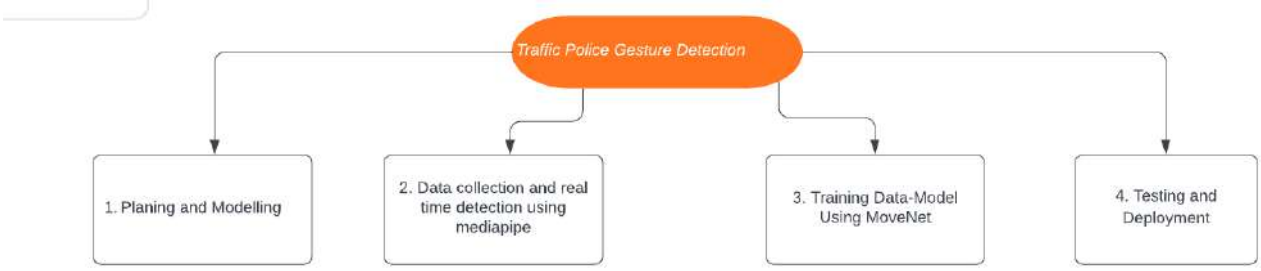
Timeline - Gantt Chart

The divison of project in sub phases using a Gant chart.

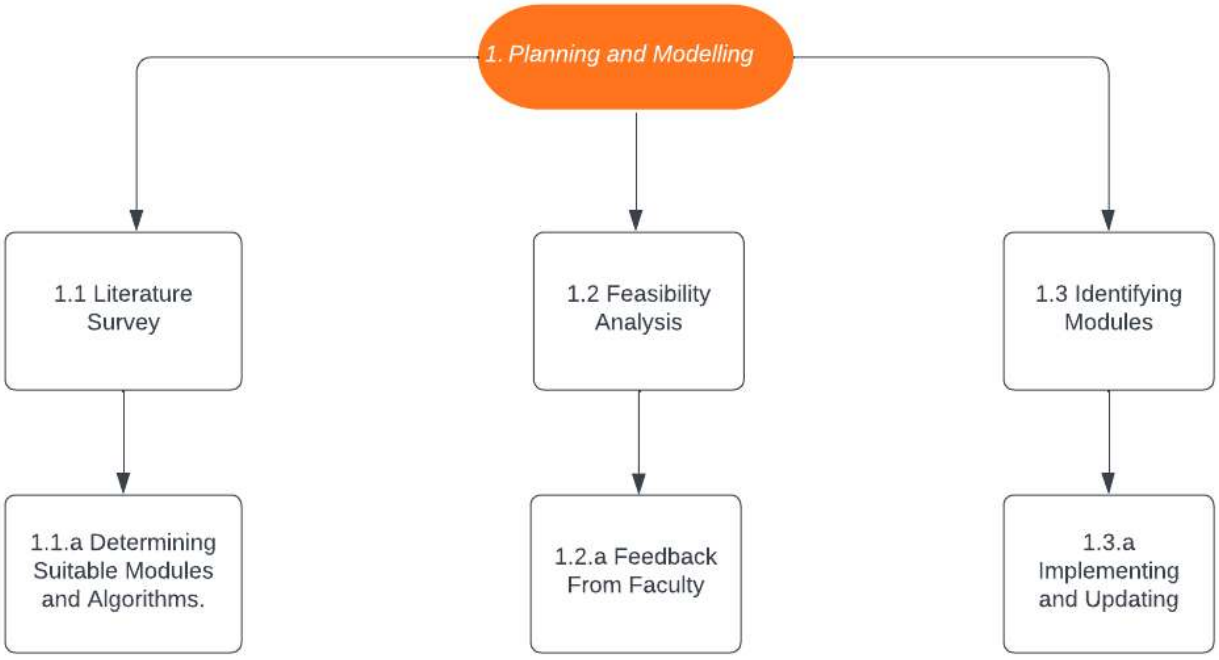


Workflow

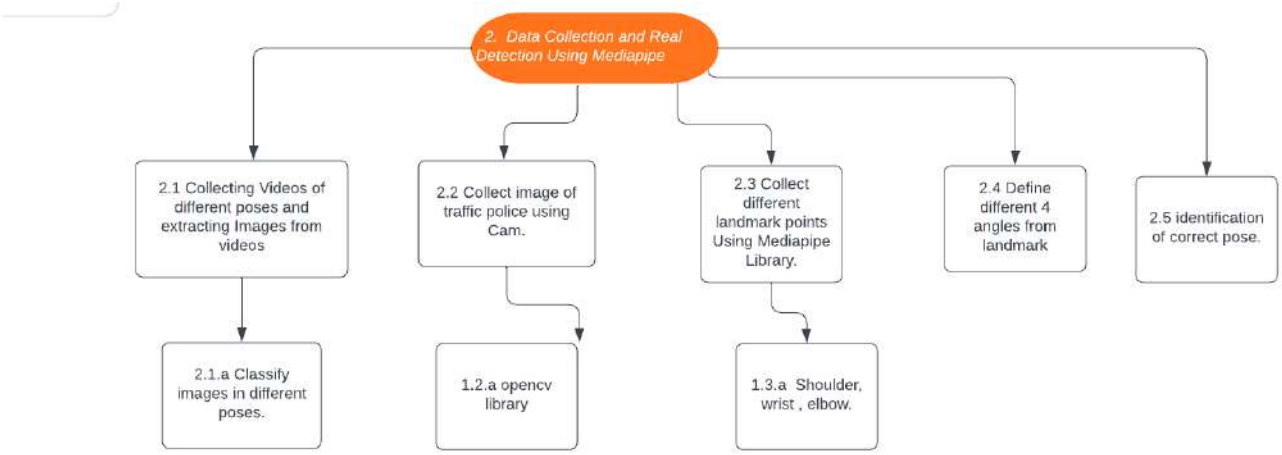
Tasks in various phases using flowchart diagram



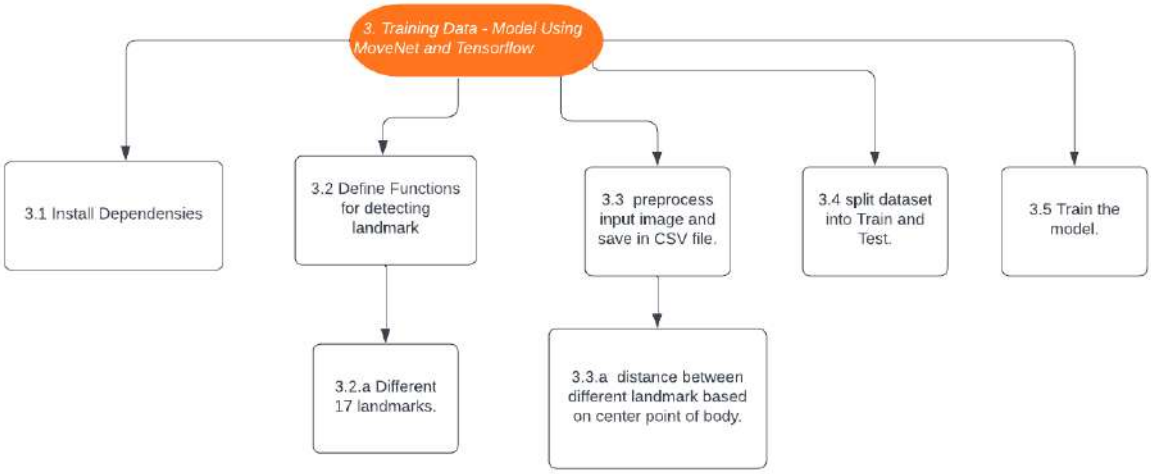
1.Planning and Modelling



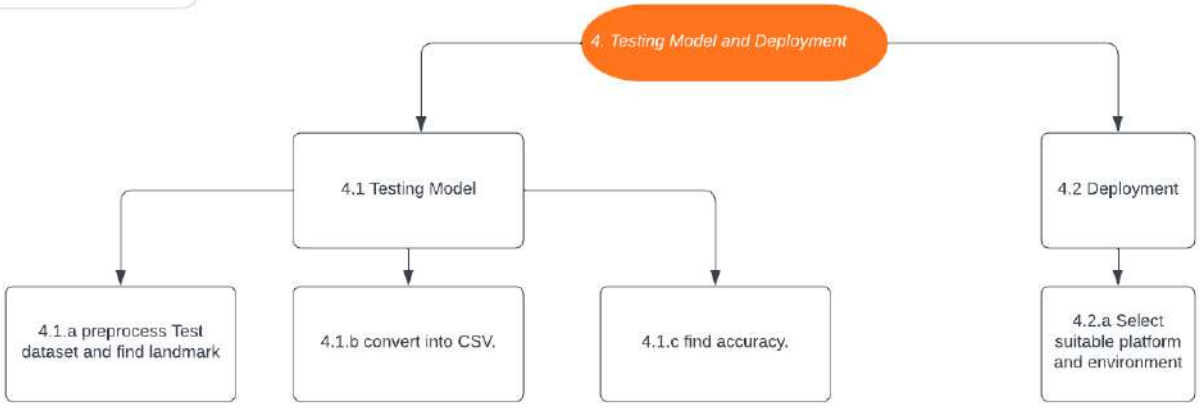
2.DataCollection and Real time detection usinf mediapipe



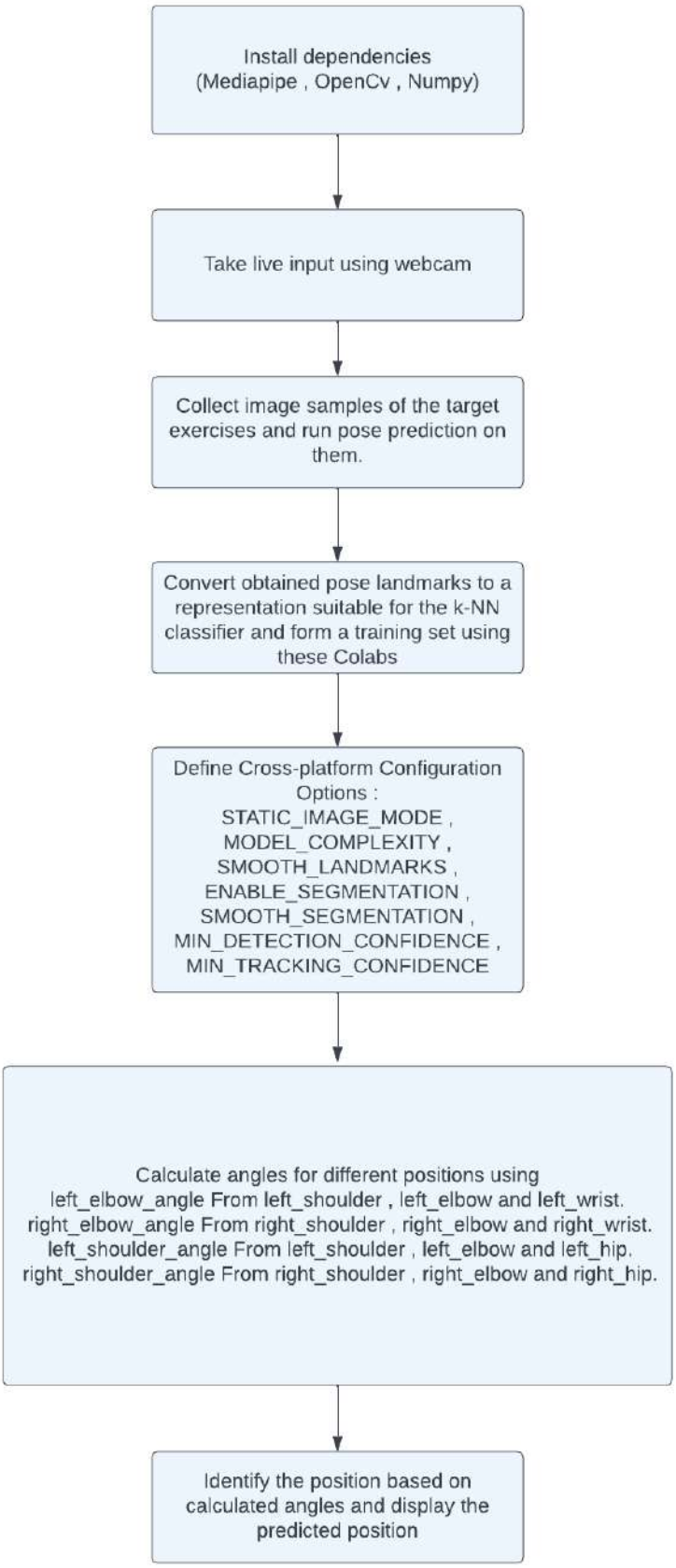
3.Training Data model using MoveNet



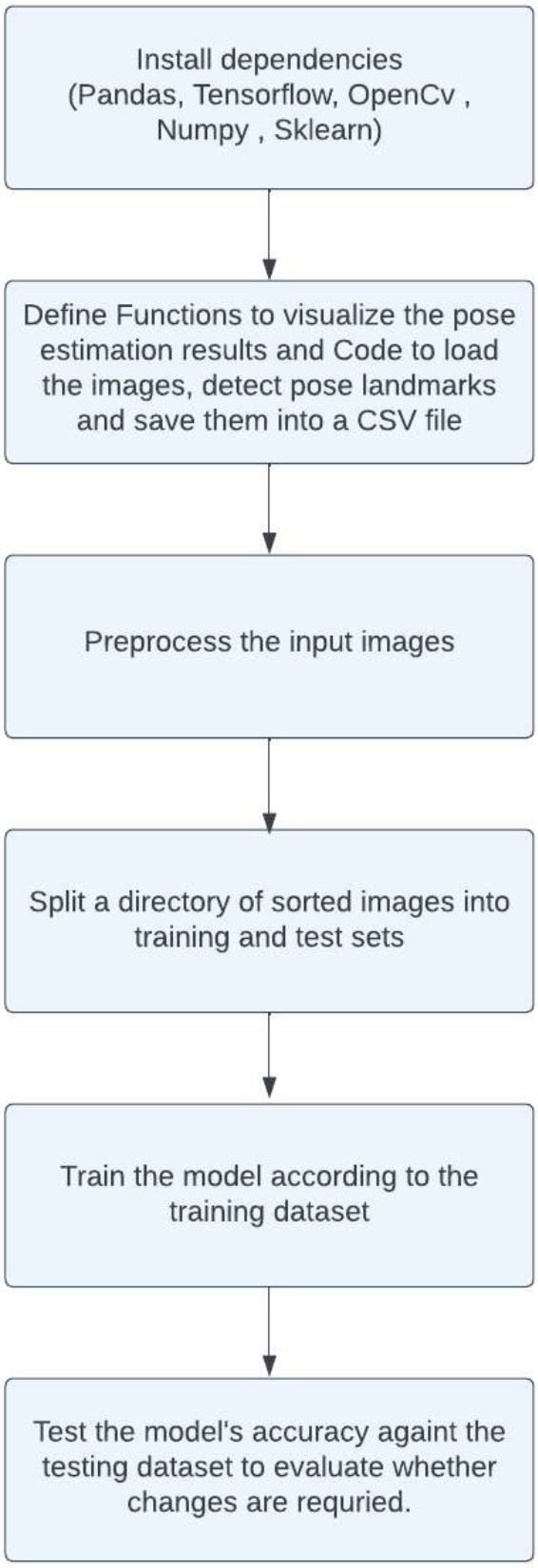
4.Testing and Deployment



Workflow Breakdown(Using Mediapipe):



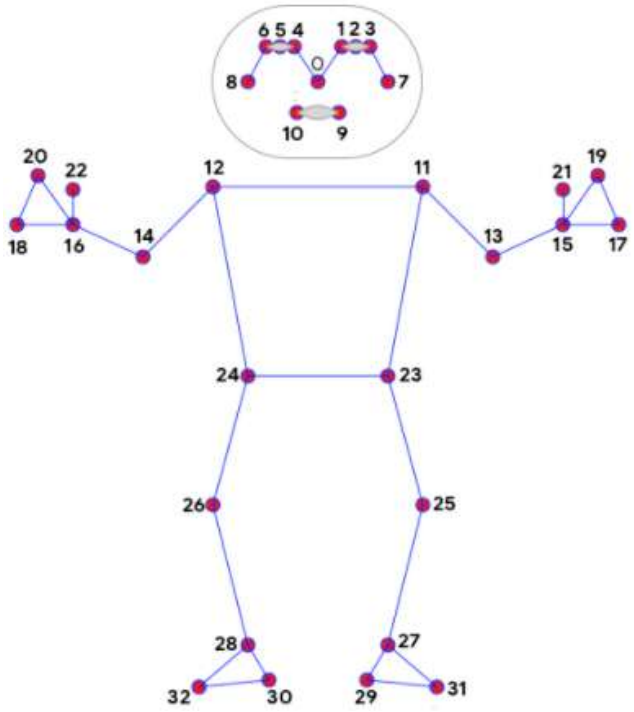
Workflow Breakdown(Using MoveNet Model):



Tools and Software: Implementation

1. Mediapipe Library(for Realtime Detection)

- MediaPipe is a Framework for building machine learning pipelines for processing time-series data like video, audio, etc.
- This cross-platform Framework works in Desktop/Server, Android, iOS, and embedded devices like Raspberry Pi and Jetson Nano.
- MediaPipe Toolkit comprises the Framework and the Solutions.
- Handpose recognition is a deep learning technique that allows you to detect different points on your hand.
- These points on your hand are commonly referred to as landmarks.
- These landmarks consist of joints, tips, and bases of your fingers.

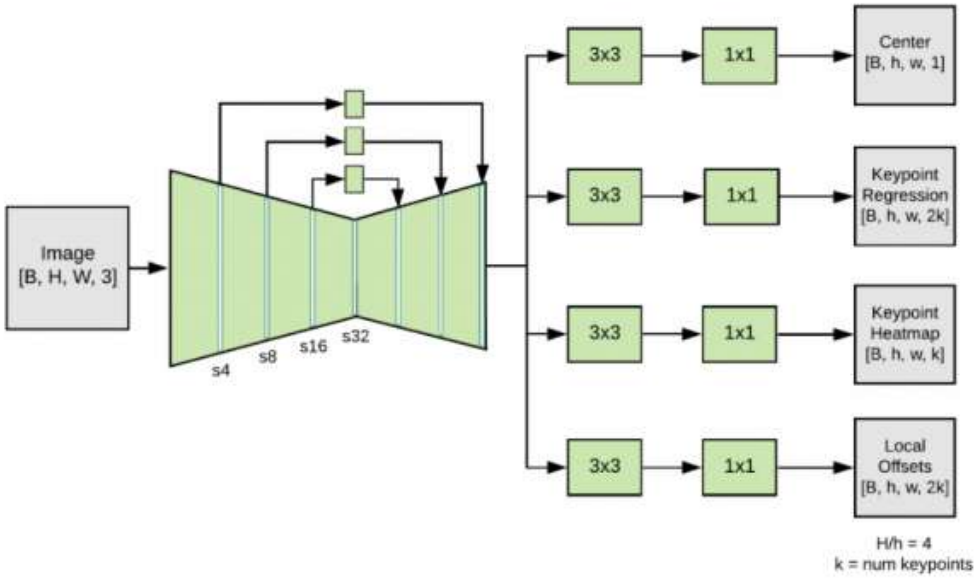


0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

1. left_elbow_angle From left_shoulder , left_elbow and left_wrist.
2. right_elbow_angle From right_shoulder , right_elbow and right_wrist.
3. left_shoulder_angle From left_shoulder , left_elbow and left_hip.
4. right_shoulder_angle From right_shoulder , right_elbow and right_hip.

2. MoveNet Model(Test and Train Dataset)

- MoveNet is a model that recognises 17 key spots on a body very quickly and precisely.
- Two variations of the model, dubbed Lightning and Thunder, are available on TF Hub.
- Thunder is designed for applications requiring great precision, whereas Lightning is designed for applications where latency is crucial.
- For the majority of contemporary computers, laptops, and phones, both models operate faster than real time (30+ FPS), which is essential for live fitness, health, and wellness applications.



- Numpy and Pandas Library for CSV files.
- opencv (cv2) for realtime video detection and extraction of landmark points.
- tensorflow : MovenetModel Training and Testing.
- sklearn
- Keras Model : Pose Classification

How MoveNet Model Works?

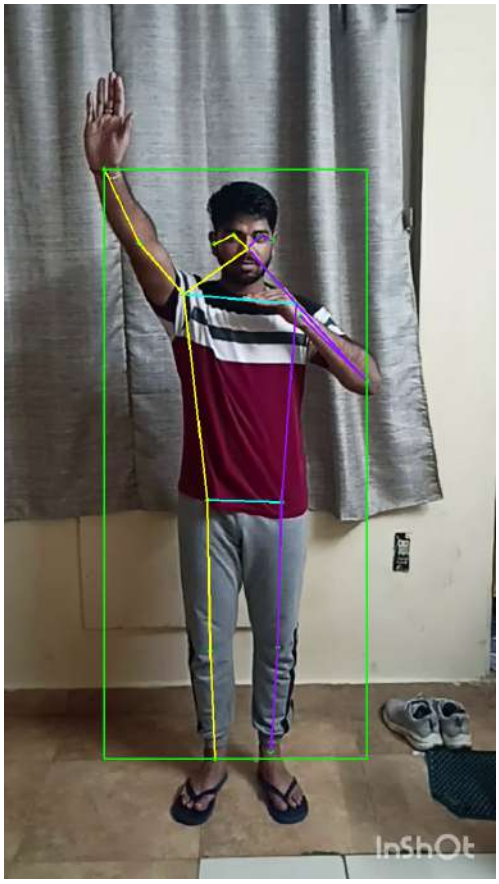
1. **MoveNet Thunder:** The model chosen for the application is the Thunder variant of the Movenet model. MoveNet thunder, even though is slightly slower compared to its counterpart – lightening, is highly accurate, suitable for the proposed application.
2. **detect(input_tensor, inference_count=3):** Runs detection on an input image.
   Args :
   input_tensor : A [height, width, 3] Tensor of type tf.float32.Note that height and width can be anything since the image will be immediately resized according to the needs of the model within thisfunction.
   inference_count : Number of times the model should run repeatly on the same input image to improve detection accuracy.
   Returns :
   A Person entity detected by the MoveNet.SinglePose.
3. **draw_prediction_on_image(image,person,crop_region=None,close_figure=True, keep_input_size=False):** Draws the keypoint predictions on image.
   Args :
   image: An numpy array with shape [height, width, channel] representing the pixel values of the input

image.
person: A person entity returned from the MoveNet.SinglePose model.
close_figure: Whether to close the plt figure after the function returns.
keep_input_size: Whether to keep the size of the input image.
Returns:An numpy array with shape [out_height, out_width, channel] representing the image overlaid with keypoint predictions.

4. **Class MoveNetPreprocessor(object):Helper class to preprocess pose sample images for classification.Creates a preprocessor to detection pose from images and save as CSV.**
5. **Split Dataset into train and test.**
6. **load pose landmarks from images and store into csv files.**
7. **get center point for all landmarks and store it;s distance from every landmarks into csv file.**
8. **Now,normalize every coordinates by moving center to (0,0).**
9. **Define the Model and train Datset.**

Example of Dataset:

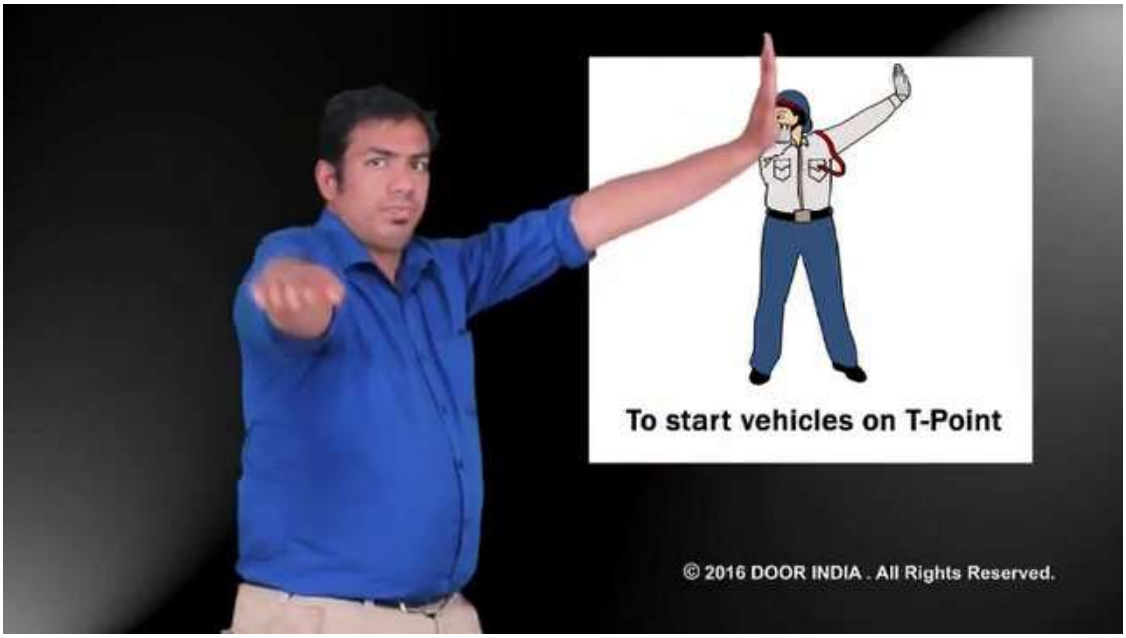
Dataset with overlapping model:



By identifying the 32 description points on the dataset image we are able to identify the angles and position the subject is forming which helps determine its gesture.

NoPose

Start Vehicle on T point



Start Vehicle From Left

Start Vehicle From Right

Stop Vehicles From simultaneously From Front and Behind

Stop Vehicles From Behind

Stop Vehicles From Front

Stop Vehicles From Left and Right

Link to Project Demo(Video Presentation):

https://drive.google.com/file/d/190LRb8DeG4hW6hkD--nzLw6wbiDAH813/view?usp=share_link

Code:

Link:

https://colab.research.google.com/drive/12VW66qOZcqRNwm26GZmab0se73G3cnDf

```
#os, random, shutil, and csv:
#These libraries provide functionality to work with the file system, randomly generate values, and

#numpy, pandas, matplotlib, and cv2:
#These libraries are commonly used in data science and machine learning workflows to manipulate, a

#machine learning libraries such as tensorflow and keras.
#These libraries provide functionality to build, train, and evaluate machine learning models.
#The script also imports sklearn which provides additional machine learning utilities such as trai

import os
import random
import shutil
import csv
import cv2
import itertools
import numpy as np
import pandas as pd
import sys
import tempfile
import tqdm
import time

from matplotlib import pyplot as plt
from matplotlib.collections import LineCollection

import tensorflow as tf
import tensorflow_hub as hub
from tensorflow import keras
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
#setting up the path to a specific example in the TensorFlow Lite library for pose estimation on R
#The first line uses the os library to join the current working directory with a subdirectory path
#The resulting path is stored in the pose_sample_rpi_path variable.

#The second line of code uses the sys library to append the pose_sample_rpi_path to the system pat
#allowing Python to locate and import modules from this directory.
pose_sample_rpi_path = os.path.join(os.getcwd(), 'examples/lite/examples/pose_estimation/raspberry
sys.path.append(pose_sample_rpi_path)
```

```
#The code also imports the Movenet class from the ml module, which is likely a wrapper class aroun
#The Movenet class takes a string argument specifying the type of Movenet model to use. In this ca
import utils
from data import BodyPart
from ml import Movenet
movenet = Movenet('movenet_thunder')
```

```
def detect(input_tensor, inference_count=3):
  """Runs detection on an input image.
  Args:
    input_tensor: A [height, width, 3] Tensor of type tf.float32.
      Note that height and width can be anything since the image will be
      immediately resized according to the needs of the model within this
      function.
    inference_count: Number of times the model should run repeatly on the
      same input image to improve detection accuracy.

  Returns:
    A Person entity detected by the MoveNet.SinglePose.
  """
  channel,image_height, image_width = input_tensor.shape

  # Detect pose using the full input image
  movenet.detect(input_tensor.numpy(), reset_crop_region=True)

  # Repeatedly using previous detection result to identify the region of
  # interest and only croping that region to improve detection accuracy
  for _ in range(inference_count - 1):
    person = movenet.detect(input_tensor.numpy(),
                            reset_crop_region=False)

  return person


def draw_prediction_on_image(
    image, person, crop_region=None, close_figure=True,
    keep_input_size=False):
  """Draws the keypoint predictions on image.

  Args:
    image: An numpy array with shape [height, width, channel] representing the
      pixel values of the input image.
    person: A person entity returned from the MoveNet.SinglePose model.
    close_figure: Whether to close the plt figure after the function returns.
    keep_input_size: Whether to keep the size of the input image.

  Returns:
    An numpy array with shape [out_height, out_width, channel] representing the
    image overlaid with keypoint predictions.
  """
  # Draw the detection result on top of the image.
  image_np = utils.visualize(image, [person])

  # Plot the image with detection results.
  height, width, channel = image.shape
  aspect_ratio = float(width) / height
  fig, ax = plt.subplots(figsize=(12 * aspect_ratio, 12))
  im = ax.imshow(image_np)

  if close_figure:
    plt.close(fig)
```

```
  if not keep_input_size:
    image_np = utils.keep_aspect_ratio_resizer(image_np, (512, 512))

  return image_np


class MoveNetPreprocessor(object):
  """Helper class to preprocess pose sample images for classification."""

  def __init__(self,
               images_in_folder,
               images_out_folder,
               csvs_out_path):
    """Creates a preprocessor to detection pose from images and save as CSV.

    Args:
      images_in_folder: Path to the folder with the input images. It should
        follow this structure:
        yoga_poses
        |__ downdog
            |______ 00000128.jpg
            |______ 00000181.bmp
            |______ ...
        |__ goddess
            |______ 00000243.jpg
            |______ 00000306.jpg
            |______ ...
        ...
      images_out_folder: Path to write the images overlay with detected
        landmarks. These images are useful when you need to debug accuracy
        issues.
      csvs_out_path: Path to write the CSV containing the detected landmark
        coordinates and label of each image that can be used to train a pose
        classification model.
    """
    self._images_in_folder = images_in_folder
    self._images_out_folder = images_out_folder
    self._csvs_out_path = csvs_out_path
    self._messages = []

    # Create a temp dir to store the pose CSVs per class
    self._csvs_out_folder_per_class = tempfile.mkdtemp()

    # Get list of pose classes and print image statistics
    self._pose_class_names = sorted(
        [n for n in os.listdir(self._images_in_folder) if not n.startswith('.')]
        )

  def process(self, per_pose_class_limit=None, detection_threshold=0.1):
    """Preprocesses images in the given folder.
    Args:
      per_pose_class_limit: Number of images to load. As preprocessing usually
        takes time, this parameter can be specified to make the reduce of the
        dataset for testing.
      detection_threshold: Only keep images with all landmark confidence score
        above this threshold.
```

```
    """
    # Loop through the classes and preprocess its images
    for pose_class_name in self._pose_class_names:
      print('Preprocessing', pose_class_name, file=sys.stderr)

      # Paths for the pose class.
      images_in_folder = os.path.join(self._images_in_folder, pose_class_name)
      images_out_folder = os.path.join(self._images_out_folder, pose_class_name)
      csv_out_path = os.path.join(self._csvs_out_folder_per_class,
                                  pose_class_name + '.csv')
      if not os.path.exists(images_out_folder):
        os.makedirs(images_out_folder)

      # Detect landmarks in each image and write it to a CSV file
      with open(csv_out_path, 'w') as csv_out_file:
        csv_out_writer = csv.writer(csv_out_file,
                                    delimiter=',',
                                    quoting=csv.QUOTE_MINIMAL)
        # Get list of images
        image_names = sorted(
            [n for n in os.listdir(images_in_folder) if not n.startswith('.')])
        if per_pose_class_limit is not None:
          image_names = image_names[:per_pose_class_limit]

        valid_image_count = 0

        # Detect pose landmarks from each image
        for image_name in tqdm.tqdm(image_names):
          image_path = os.path.join(images_in_folder, image_name)

          try:
            image = tf.io.read_file(image_path)
            image = tf.io.decode_jpeg(image)
          except:
            self._messages.append('Skipped ' + image_path + '. Invalid image.')
            continue
          else:
            image = tf.io.read_file(image_path)
            image = tf.io.decode_jpeg(image)
            image_height, image_width, channel = image.shape

          # Skip images that isn't RGB because Movenet requires RGB images
          if channel != 3:
            self._messages.append('Skipped ' + image_path +
                                  '. Image isn\'t in RGB format.')
            continue
          person = detect(image)

          # Save landmarks if all landmarks were detected
          min_landmark_score = min(
              [keypoint.score for keypoint in person.keypoints])
          should_keep_image = min_landmark_score >= detection_threshold
          if not should_keep_image:
            self._messages.append('Skipped ' + image_path +
                                  '. No pose was confidentlly detected.')
            continue
```

```
        valid_image_count += 1

        # Draw the prediction result on top of the image for debugging later
        output_overlay = draw_prediction_on_image(
            image.numpy().astype(np.uint8), person,
            close_figure=True, keep_input_size=True)

        # Write detection result into an image file
        output_frame = cv2.cvtColor(output_overlay, cv2.COLOR_RGB2BGR)
        cv2.imwrite(os.path.join(images_out_folder, image_name), output_frame)

        # Get landmarks and scale it to the same size as the input image
        pose_landmarks = np.array(
            [[keypoint.coordinate.x, keypoint.coordinate.y, keypoint.score]
              for keypoint in person.keypoints],
            dtype=np.float32)

        # Write the landmark coordinates to its per-class CSV file
        coordinates = pose_landmarks.flatten().astype(str).tolist()
        # coordinates = pose_landmarks.flatten().astype(np.str).tolist()
        csv_out_writer.writerow([image_name] + coordinates)

      if not valid_image_count:
        raise RuntimeError(
            'No valid images found for the "{}" class.'
            .format(pose_class_name))

  # Print the error message collected during preprocessing.
  print('\n'.join(self._messages))

  # Combine all per-class CSVs into a single output file
  all_landmarks_df = self._all_landmarks_as_dataframe()
  all_landmarks_df.to_csv(self._csvs_out_path, index=False)

def class_names(self):
  """List of classes found in the training dataset."""
  return self._pose_class_names

def _all_landmarks_as_dataframe(self):
  """Merge all per-class CSVs into a single dataframe."""
  total_df = None
  for class_index, class_name in enumerate(self._pose_class_names):
    csv_out_path = os.path.join(self._csvs_out_folder_per_class,
                                class_name + '.csv')
    per_class_df = pd.read_csv(csv_out_path, header=None)

    # Add the labels
    per_class_df['class_no'] = [class_index]*len(per_class_df)
    per_class_df['class_name'] = [class_name]*len(per_class_df)

    # Append the folder name to the filename column (first column)
    per_class_df[per_class_df.columns[0]] = (os.path.join(class_name, '')
      + per_class_df[per_class_df.columns[0]].astype(str))

    if total_df is None:
```

```
        # For the first class, assign its data to the total dataframe
        total_df = per_class_df
      else:
        # Concatenate each class's data into the total dataframe
        total_df = pd.concat([total_df, per_class_df], axis=0)

    list_name = [[bodypart.name + '_x', bodypart.name + '_y',
                  bodypart.name + '_score'] for bodypart in BodyPart]
    header_name = []
    for columns_name in list_name:
      header_name += columns_name
    header_name = ['file_name'] + header_name
    header_map = {total_df.columns[i]: header_name[i]
                  for i in range(len(header_name))}

    total_df.rename(header_map, axis=1, inplace=True)

    return total_df


#read image from given dir and store content of image into tensor.pass it from detect function to
image = tf.io.read_file('Dataset//Stop Vehicles From Front//img996.jpg')
image = tf.io.decode_jpeg(image)
person = detect(image)= draw_prediction_on_image(image.numpy(), person, crop_region=None,
                                close_figure=False, keep_input_size=True)
```

```
def processTensorImage(path):
    if(type(path) == str):
        image = tf.io.read_file(path)
        image = tf.image.decode_jpeg(image)
    else:
        image = tf.convert_to_tensor(path, dtype=tf.float32)
    return image


# def processTensorImageFromNumpy(image):
#     image = tf.convert_to_tensor(image, dtype=tf.float32)
#     return image


use_custom_dataset = True
is_skip_step_1 = False
dataset_is_split = False


def split_into_train_test(images_origin, images_dest, test_split):
  """Splits a directory of sorted images into training and test sets.

  Args:
    images_origin: Path to the directory with your images. This directory
      must include subdirectories for each of your labeled classes. For example:
      yoga_poses/
```

```
      |__ downdog/
          |______ 00000128.jpg
          |______ 00000181.jpg
          |______ ...
      |__ goddess/
          |______ 00000243.jpg
          |______ 00000306.jpg
          |______ ...
      ...
    images_dest: Path to a directory where you want the split dataset to be
      saved. The results looks like this:
      split_yoga_poses/
      |__ train/
          |__ downdog/
              |______ 00000128.jpg
              |______ ...
      |__ test/
          |__ downdog/
              |______ 00000181.jpg
              |______ ...
    test_split: Fraction of data to reserve for test (float between 0 and 1).
  """
  _, dirs, _ = next(os.walk(images_origin))

  TRAIN_DIR = os.path.join(images_dest, 'train')
  TEST_DIR = os.path.join(images_dest, 'test')
  os.makedirs(TRAIN_DIR, exist_ok=True)
  os.makedirs(TEST_DIR, exist_ok=True)

  for dir in dirs:
    # Get all filenames for this dir, filtered by filetype
    filenames = os.listdir(os.path.join(images_origin, dir))
    filenames = [os.path.join(images_origin, dir, f) for f in filenames if (
        f.endswith('.png') or f.endswith('.jpg') or f.endswith('.jpeg') or f.endswith('.bmp'))]
    # Shuffle the files, deterministically
    filenames.sort()
    random.seed(42)
    random.shuffle(filenames)
    # Divide them into train/test dirs
    os.makedirs(os.path.join(TEST_DIR, dir), exist_ok=True)
    os.makedirs(os.path.join(TRAIN_DIR, dir), exist_ok=True)
    test_count = int(len(filenames) * test_split)
    for i, file in enumerate(filenames):
      if i < test_count:
        destination = os.path.join(TEST_DIR, dir, os.path.split(file)[1])
      else:
        destination = os.path.join(TRAIN_DIR, dir, os.path.split(file)[1])
      shutil.copyfile(file, destination)
    print(
        f'Moved {test_count} of {len(filenames)} from class "{dir}" into test.')
  print(f'Your split dataset is in "{images_dest}"')



#SPLIT DATASET INTO TEST AND TRAIN
```

```
# dataset_in = 'Dataset'
#  # You can leave the rest alone:
# if not os.path.isdir(dataset_in):
#    raise Exception("dataset_in is not a valid directory")
# IMAGES_ROOT = dataset_in


dataset_in = 'Dataset'

if not os.path.isdir(dataset_in):
   raise Exception("dataset_in is not a valid directory")

dataset_out = 'split_' + dataset_in
split_into_train_test(dataset_in, dataset_out, test_split=0.2)
IMAGES_ROOT = dataset_out



#TRAINING DATASET

images_in_train_folder = os.path.join(IMAGES_ROOT, 'train') #path to train dataset
images_out_train_folder = 'poses_images_out_train' #tensor files for train datset
csvs_out_train_path = 'train_data.csv' #path to csv file of train dataset

#training model using MOVENETPREPROCESSOR
preprocessor = MoveNetPreprocessor(
    images_in_folder=images_in_train_folder,
    images_out_folder=images_out_train_folder,
    csvs_out_path=csvs_out_train_path,
)

#preprocessor.process(per_pose_class_limit=None, detection_threshold=0.0)
preprocessor.process(per_pose_class_limit=None)



#Now TESTING the Model Using Test Dataset

images_in_test_folder = os.path.join(IMAGES_ROOT, 'test')
images_out_test_folder = 'poses_images_out_test'
csvs_out_test_path = 'test_data.csv'

preprocessor = MoveNetPreprocessor(
    images_in_folder=images_in_test_folder,
    images_out_folder=images_out_test_folder,
    csvs_out_path=csvs_out_test_path,
)

#preprocessor.process(per_pose_class_limit=None,detection_threshold=0.0)
preprocessor.process(per_pose_class_limit=None)



def load_pose_landmarks(csv_path):
  """Loads a CSV created by MoveNetPreprocessor.

  Returns:
    X: Detected landmark coordinates and scores of shape (N, 17 * 3)
```

```
    y: Ground truth labels of shape (N, label_count)
    classes: The list of all class names found in the dataset
    dataframe: The CSV loaded as a Pandas dataframe features (X) and ground
      truth labels (y) to use later to train a pose classification model.
  """

  # Load the CSV file
  dataframe = pd.read_csv(csv_path)
  df_to_process = dataframe.copy()

  # Drop the file_name columns as you don't need it during training.
  df_to_process.drop(columns=['file_name'], inplace=True)

  # Extract the list of class names
  classes = df_to_process.pop('class_name').unique()

  # Extract the labels
  y = df_to_process.pop('class_no')

  # Convert the input features and labels into the correct format for training.
  X = df_to_process.astype('float64')
  y = keras.utils.to_categorical(y)

  return X, y, classes, dataframe


# Load the train data
X, y, class_names, _ = load_pose_landmarks(csvs_out_train_path)

# Split training data (X, y) into (X_train, y_train) and (X_val, y_val)
X_train, X_val, y_train, y_val = train_test_split(X, y,
                                                  test_size=0.15)


# Load the test data
X_test, y_test, _, df_test = load_pose_landmarks(csvs_out_test_path)


def get_center_point(landmarks, left_bodypart, right_bodypart):
  """Calculates the center point of the two given landmarks."""

  left = tf.gather(landmarks, left_bodypart.value, axis=1)
  right = tf.gather(landmarks, right_bodypart.value, axis=1)
  center = left * 0.5 + right * 0.5
  return center


def get_pose_size(landmarks, torso_size_multiplier=2.5):
  """Calculates pose size.

  It is the maximum of two values:
    * Torso size multiplied by `torso_size_multiplier`
    * Maximum distance from pose center to any pose landmark
  """
  # Hips center
```

```
  hips_center = get_center_point(landmarks, BodyPart.LEFT_HIP,
                                 BodyPart.RIGHT_HIP)

  # Shoulders center
  shoulders_center = get_center_point(landmarks, BodyPart.LEFT_SHOULDER,
                                      BodyPart.RIGHT_SHOULDER)

  # Torso size as the minimum body size
  torso_size = tf.linalg.norm(shoulders_center - hips_center)

  # Pose center
  pose_center_new = get_center_point(landmarks, BodyPart.LEFT_HIP,
                                     BodyPart.RIGHT_HIP)
  pose_center_new = tf.expand_dims(pose_center_new, axis=1)
  # Broadcast the pose center to the same size as the landmark vector to
  # perform substraction
  pose_center_new = tf.broadcast_to(pose_center_new,[tf.size(landmarks) // (17*2), 17, 2])

  # Dist to pose center
  d = tf.gather(landmarks - pose_center_new, 0, axis=0,
                name="dist_to_pose_center")
  # Max dist to pose center
  max_dist = tf.reduce_max(tf.linalg.norm(d, axis=0))

  # Normalize scale
  pose_size = tf.maximum(torso_size * torso_size_multiplier, max_dist)

  return pose_size


def normalize_pose_landmarks(landmarks):
  """Normalizes the landmarks translation by moving the pose center to (0,0) and
  scaling it to a constant pose size.
  """
  # Move landmarks so that the pose center becomes (0,0)
  pose_center = get_center_point(landmarks, BodyPart.LEFT_HIP,
                                 BodyPart.RIGHT_HIP)
  pose_center = tf.expand_dims(pose_center, axis=1)
  # Broadcast the pose center to the same size as the landmark vector to perform
  # substraction
  pose_center = tf.broadcast_to(pose_center,[tf.size(landmarks) // (17*2), 17, 2])
  landmarks = landmarks - pose_center

  # Scale the landmarks to a constant pose size
  pose_size = get_pose_size(landmarks)
  landmarks /= pose_size

  return landmarks


def landmarks_to_embedding(landmarks_and_scores):
  """Converts the input landmarks into a pose embedding."""
  # Reshape the flat input into a matrix with shape=(17, 3)
  reshaped_inputs = keras.layers.Reshape((17, 3))(landmarks_and_scores)
  # Normalize landmarks 2D
  landmarks = normalize_pose_landmarks(reshaped_inputs[:, :, :2])
```

```
  # Flatten the normalized landmark coordinates into a vector
  embedding = keras.layers.Flatten()(landmarks)
  return embedding



# Define the model - this is our keras - model (Using Movenet and Deep Learning)
inputs = tf.keras.Input(shape=(51))
embedding = landmarks_to_embedding(inputs)

#layer = keras.layers.Dense(128, activation=tf.nn.relu6)(embedding): This line creates a fully con
#layer = keras.layers.Dropout(0.5)(layer): This line applies a dropout layer to the output of the
#layer = keras.layers.Dense(64, activation=tf.nn.relu6)(layer): This line creates another fully co
#layer = keras.layers.Dropout(0.5)(layer): This line applies another dropout layer with a rate of
#outputs = keras.layers.Dense(len(class_names), activation="softmax")(layer): This line creates th
layer = keras.layers.Dense(128, activation=tf.nn.relu6)(embedding)
layer = keras.layers.Dropout(0.5)(layer)
layer = keras.layers.Dense(64, activation=tf.nn.relu6)(layer)
layer = keras.layers.Dropout(0.5)(layer)
outputs = keras.layers.Dense(len(class_names), activation="softmax")(layer)

model = keras.Model(inputs, outputs)
model.summary()
```

```
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Add a checkpoint callback to store the checkpoint that has the highest
# validation accuracy.
checkpoint_path = "weights.best.hdf5"
checkpoint = keras.callbacks.ModelCheckpoint(checkpoint_path,
                                             monitor='val_accuracy',
                                             verbose=1,
                                             save_best_only=True,
                                             mode='max')
earlystopping = keras.callbacks.EarlyStopping(monitor='val_accuracy',
                                              patience=20)

# Start training
history = model.fit(X_train, y_train,
                    epochs=200,
                    batch_size=16,
                    validation_data=(X_val, y_val),
                    callbacks=[checkpoint, earlystopping])



# Visualize the training history to see whether you're overfitting.
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
```

```
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['TRAIN', 'VAL'], loc='lower right')
plt.show()



# Evaluate the model using the TEST dataset
loss, accuracy = model.evaluate(X_test, y_test)



def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
  """Plots the confusion matrix."""
  if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    print("Normalized confusion matrix")
  else:
    print('Confusion matrix, without normalization')

  plt.imshow(cm, interpolation='nearest', cmap=cmap)
  plt.title(title)
  plt.colorbar()
  tick_marks = np.arange(len(classes))
  plt.xticks(tick_marks, classes, rotation=55)
  plt.yticks(tick_marks, classes)
  fmt = '.2f' if normalize else 'd'
  thresh = cm.max() / 2.
  for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
             color="white" if cm[i, j] > thresh else "black")

  plt.ylabel('True label')
  plt.xlabel('Predicted label')
  plt.tight_layout()

# Classify pose in the TEST dataset using the trained model
y_pred = model.predict(X_test)

# Convert the prediction result to class name
y_pred_label = [class_names[i] for i in np.argmax(y_pred, axis=1)]
y_true_label = [class_names[i] for i in np.argmax(y_test, axis=1)]

# Plot the confusion matrix
cm = confusion_matrix(np.argmax(y_test, axis=1), np.argmax(y_pred, axis=1))
plot_confusion_matrix(cm,
                      class_names,
                      title ='Confusion Matrix of Pose Classification Model')

# Print the classification report
print('\nClassification Report:\n', classification_report(y_true_label,
                                                         y_pred_label))
```

```
Classification Report:
                                precision    recall  f1-score   support

                        NoPose       0.96      1.00      0.98        26
          Start Vehicle On T Point       0.97      0.97      0.97        79
          Start Vehicles From Left       0.95      0.99      0.97       102
         Start Vehicles From Right       0.94      0.93      0.93        97
         Start Vehicles On T Point       0.96      0.94      0.95       109
         Stop Vehicles From Behind       0.94      0.97      0.95       106
Stop Vehicles From Left and Right       0.98      0.99      0.99       115
          Stop Vehicles from Front       0.99      0.92      0.95        95

                        accuracy                           0.96       729
                       macro avg       0.96      0.96      0.96       729
                    weighted avg       0.96      0.96      0.96       729
```

```python
def processTensorImageFromNumpy(image):
    image = tf.convert_to_tensor(image, dtype=tf.float32)
    return image

# def processTensorImage(path):
#     image = tf.io.read_file(path)
#     image = tf.image.decode_jpeg(image)
#     return image


# image = processTensorImage(image): This line calls the processTensorImage function to preprocess
#person = detect(image): This line calls a function detect to detect a person in the input image u
#pose_landmarks = np.array([[keypoint.coordinate.x, keypoint.coordinate.y, keypoint.score] for key
#coordinates = pose_landmarks.flatten().astype(str).tolist(): This line flattens the pose_landmark
#df = pd.DataFrame([coordinates]).reset_index(drop=True): This line creates a Pandas DataFrame df
#X = df.astype('float64'): This line converts the data type of the df DataFrame to float64, which
#y = model.predict(X): This line passes the input data X to the trained model model to get the pre
#y_pred = [class_names[i] for i in np.argmax(y, axis=1)]: This line finds the index of the maximum
#return y_pred[0]: This line returns the predicted pose classification label for the input image t
def classifyPose(image):
    image = processTensorImage(image)
    # image = processTensorImageFromNumpy(image)
    person = detect(image)
    pose_landmarks = np.array([[keypoint.coordinate.x, keypoint.coordinate.y, keypoint.score]
                               for keypoint in person.keypoints], dtype=np.float32)
    coordinates = pose_landmarks.flatten().astype(str).tolist()
    df = pd.DataFrame([coordinates]).reset_index(drop=True)
    X = df.astype('float64')
    y = model.predict(X)
    y_pred = [class_names[i] for i in np.argmax(y, axis=1)]
    return y_pred[0]
```

```python
camera_video = cv2.VideoCapture(0)
camera_video.set(3, 1280)
camera_video.set(4, 960)
```

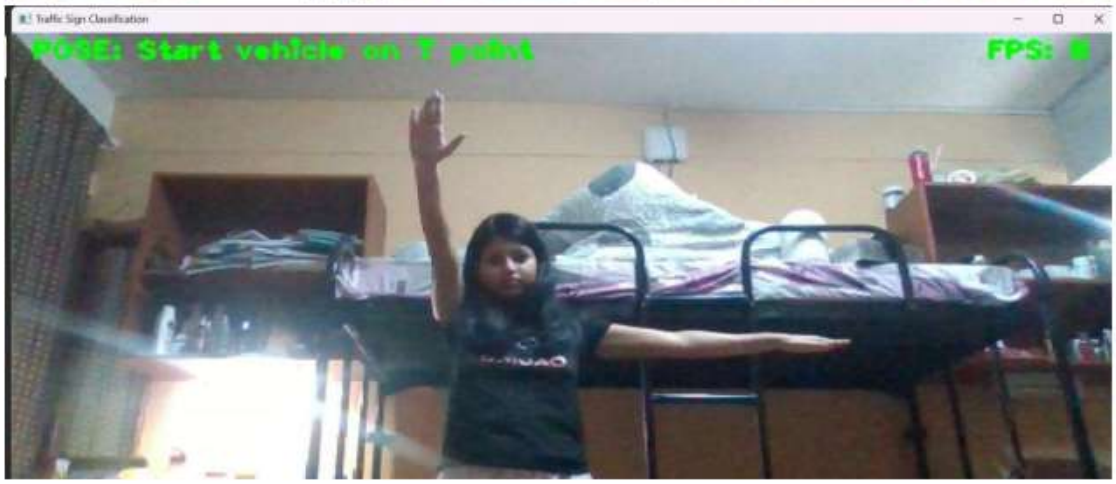
```
cv2.namedWindow('Traffic Sign Classification', cv2.WINDOW_NORMAL)
while camera_video.isOpened():
    ok, frame = camera_video.read()
    time1 = time.time()
    if not ok:
        continue
    frame_height, frame_width, _ = frame.shape
    frame = cv2.resize(
        frame, (int(frame_width * (320 / frame_height)), 320))
    pose = classifyPose(frame)
    frame = cv2.flip(frame, 1)
    time2 = time.time()
    fps = 0
    if (time2 - time1) > 0:
        fps = 1.0 / (time2 - time1)
    cv2.putText(frame, 'FPS: {}'.format(int(fps)), (500, 15),
                cv2.FONT_HERSHEY_PLAIN, 1, (0, 255, 0), 2)
    if pose=="NoPose":
        cv2.putText(frame, 'POSE: NoPose', (10, 15),
                cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 2)
    else:
        cv2.putText(frame, 'POSE: {}'.format(pose), (10, 15),
                cv2.FONT_HERSHEY_PLAIN, 1, (0, 255, 0), 2)
    time1 = time2

    cv2.imshow('Traffic Sign Classification', frame)
    k = cv2.waitKey(1) & 0xFF
    if(k == 27):
        break
camera_video.release()
cv2.destroyAllWindows()
```

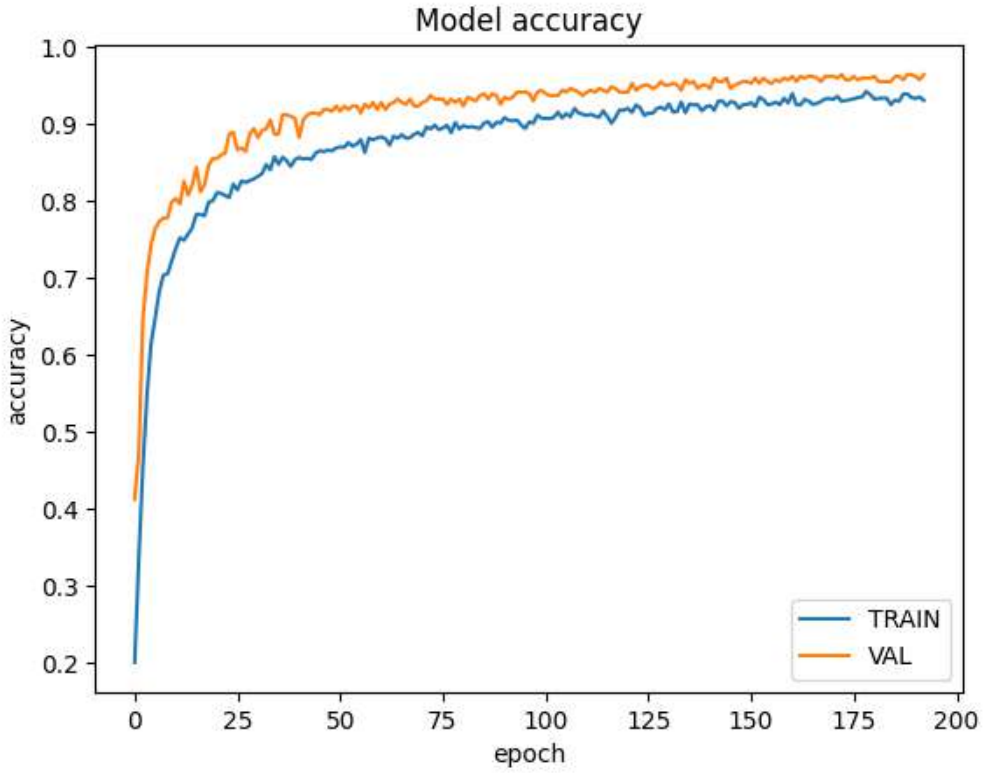
Result and Discussion on Findings:

Description about dataset used:
Since the application is novel, no such dataset exits for the proposed application. Hence, we've prepared our own dataset consisting of around 4000 images, that are classified and captures multiple environment scenarios for maximum accuracy.

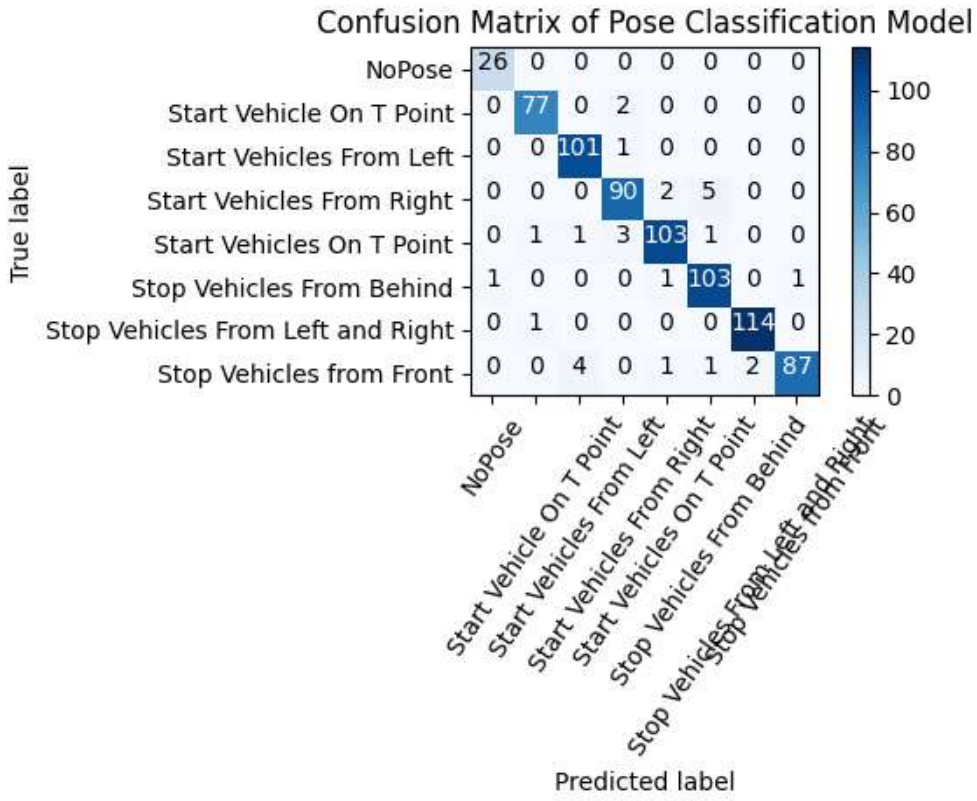
Experimental Results/Output:



Accuracy



Confusion Matrix



Classification Report

```
Classification Report:
                                precision    recall  f1-score   support

                        NoPose       0.96      1.00      0.98        26
         Start Vehicle On T Point       0.97      0.97      0.97        79
         Start Vehicles From Left       0.95      0.99      0.97       102
        Start Vehicles From Right       0.94      0.93      0.93        97
        Start Vehicles On T Point       0.96      0.94      0.95       109
        Stop Vehicles From Behind       0.94      0.97      0.95       106
Stop Vehicles From Left and Right       0.98      0.99      0.99       115
         Stop Vehicles from Front       0.99      0.92      0.95        95

                        accuracy                           0.96       729
                       macro avg       0.96      0.96      0.96       729
                    weighted avg       0.96      0.96      0.96       729
```

Issues in Existing System:

- Hand gesture recognition is an important task for traffic police as it enables them to control traffic flow more efficiently. In order to develop a deep learning model for hand gesture recognition, one approach is to use popular object detection models like YOLO. However, a slight change in the background can entirely change the image for the model, leading to false positives or true negatives in the predictions.To overcome this limitation, we need to train the model on a diverse

set of images that capture different backgrounds, lighting conditions, and camera angles. This will enable the model to generalize better to new images that it has not seen before.

- Autonomous vehicles require traffic police gesture recognition. Current traffic police gesture identification systems frequently extract pixel-level characteristics from RGB photos, which are incoherent owing to the absence of gesture skeleton features and can result in erroneous results. Existing object detection algorithms enable the detection of automobiles, trees, people, bicycles, animals, and so forth (YOLO).
- Another approach for hand gesture recognition is to use heuristic-based models that detect gestures on the basis of the location of feature points as perceived by the camera. However, this approach fails to perform with slight deviations in the relative position of the object on the image, slight shift in angle with respect to the camera and also the distance the object is from the camera.
- To address this issue, we can use a combination of heuristic-based and deep learning-based approaches. The heuristic-based model can be used as a pre-processing step to detect the location of the hand in the image and estimate its orientation and distance from the camera. This information can then be used to crop the image and feed it to a deep learning model for gesture recognition.
- Moreover, data augmentation techniques such as random rotations, translations, and scaling can be applied to the training data to make the model more robust to variations in the input. In addition, we can also use techniques such as transfer learning, where a pretrained model on a large dataset can be fine-tuned on a smaller dataset of hand gestures to improve its performance.
- Overall, developing an accurate and robust hand gesture recognition model for traffic police requires careful consideration of various factors such as the choice of model, diversity of training data, data augmentation techniques, and pre-processing steps.

Novelty of proposed work:

- Nowdays In autonomous vehicles to detect different objects like other cars,trees,human,animals,water etc..YOLO algorithm is used,and this YOLO Algorithm has one drawback which is it can not detect traffic police gestures (Which is important in indian scenario) ,so our Movenet Model will overcome this issue and it is trained in such way that it can easily detect different traffic police gestures.
- As there are no dataset available for traffic police gestures we have collected around 5000 images and made our own custom dataset.We have collected this by making video of every pose and than extracted images from that video.
- In Present in indian scenario there is no algorithm available which can able to detect this type of traffic police gestures,we are the first one to introduce this.

Future Work:

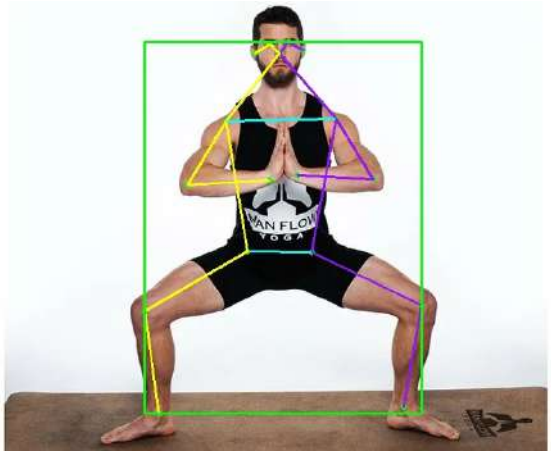
- As we are only detecting traffic police signals using movenet model,our model will only focus on the traffic police hand signals and it will not able to detect wheather the personal showing traffic

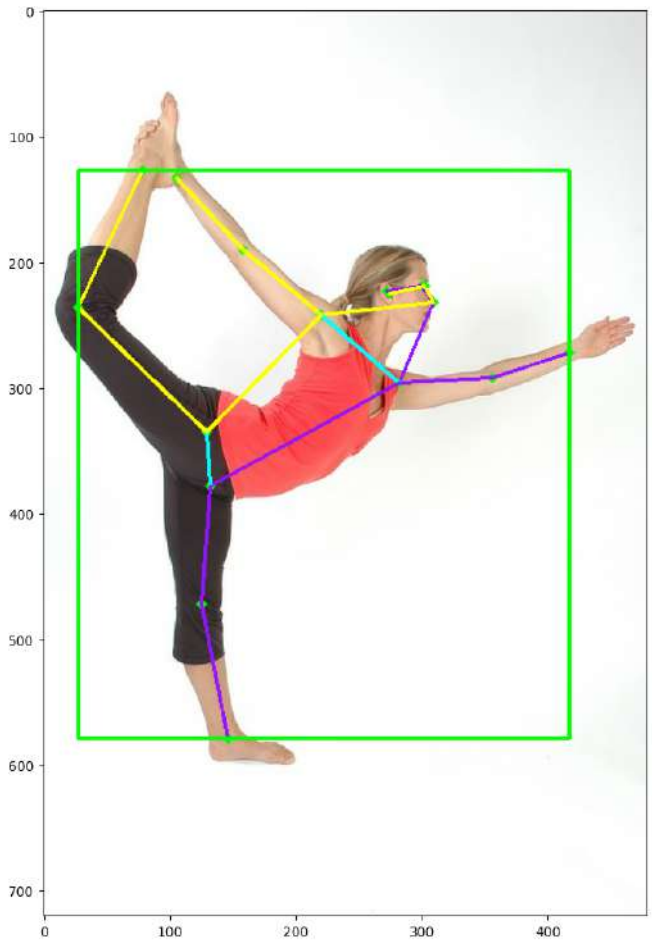
hand signals is traffic police or any normal person.so we have to train our model in such way that it can also detect traffic police by their uniform by expanding our dataset.

- Another Future work that can be done is we can merge our movenet model with YOLO Algorithm so it can detect other traffic objects like trees,cars,humans,traffic signs,signals etc. and it can be used as fully working model for autonomus vehicles.
- Add More images to Dataset to make our model more accurate in generating output and gussing the correct pose.
- For now our model can detect different signals perform by traffic police and it will inform/shown to autonomous vehicles about that signal.But we can modify it in such wawy that it can instruct autonomous vehicles about what signals traffic police showing as well as based on that what task that vehicle has to perform.
- We can make one software for training traffic police or cadets about different traffic rules and gestures.

Other Applications:

1. Yoga Pose Classification With TensorFlow's MoveNet Model





The process for yoga pose detection using the Movenet model can be broken down into the following steps:

1. Data Collection: Collect a dataset of yoga pose images or videos. The dataset should include a diverse range of individuals with different body types and clothing.

2. Data Preprocessing: Resize and normalize the images to a fixed size. This step ensures that the model can handle input of a consistent size and shape.

3. Model Training: Train the Movenet model on the preprocessed dataset. The model can be trained using supervised or unsupervised learning.

4. Inference: Use the trained model to detect yoga poses in real-time or on pre-recorded videos. Inference involves processing the input data using the trained model to output the predicted pose.

5. Post-processing: Clean up the predicted pose by removing outliers and smoothing the pose over time. This step ensures that the final pose is accurate and smooth.

6. Visualization: Visualize the predicted pose by overlaying it on top of the input image or video. This step provides a visual representation of the predicted pose for evaluation and analysis.

2. Instant Motion tracking using MediaPipe:

Instant motion tracking using MediaPipe involves the following steps:

1. Install MediaPipe: MediaPipe is an open-source framework developed by Google for building pipelines to process perceptual data. You can install MediaPipe by following the installation instructions provided on the official website.

2. Import MediaPipe and relevant libraries: Once you have installed MediaPipe, you can import it along with other relevant libraries such as OpenCV for image processing.

3. Define the input source: Define the input source, which could be a video file, a live video stream, or a series of images.

4. Create a MediaPipe pipeline: Create a MediaPipe pipeline that includes the motion tracking module. MediaPipe provides pre-trained models for motion tracking that you can use out-of-the-box.

5. Process the input data: Feed the input data to the pipeline and process it using the motion tracking module. MediaPipe will detect the motion and track it in real-time.

6. Visualize the output: Finally, visualize the output of the motion tracking module using OpenCV or any other visualization library.

Literature Survey

Sr.No	Title/Author	Techniques	Future Work
1.	Object Detection in Self Driving Cars Using Deep Learning Author : Prajwal P, Prajwal D, Harish D H, Gajanana R, Jayasri B S and S. Lokesh (IEE - 2021)	• Convolutional Neural Networks (CNNs): CNNs are a type of deep learning neural network that are commonly used for image recognition and classification tasks. In the context of self-driving cars, CNNs are used to identify and classify objects in the environment, such as other vehicles, pedestrians, and traffic signs. • Region Proposal Networks (RPNs): RPNs are a type of deep learning neural network that are used to identify potential object regions in an image. In the context of self-driving cars, RPNs are used to identify potential regions of interest in the environment, which can then be analyzed further using CNNs.	• It can only able to d car,person,animal,trees,Bus,divider,b etc.

- Anchor Boxes: Anchor boxes are predefined bounding boxes that are used to predict the location of objects in an image. In the context of self-driving cars, anchor boxes are used to predict the location and size of objects in the environment.
- Non-Maximum Suppression (NMS): NMS is a technique used to filter out redundant object detections. In the context of self-driving cars, NMS is used to eliminate duplicate object detections and select the most accurate and relevant detections.
- Transfer Learning: Transfer learning is a technique used to leverage pre-trained models to solve new problems. In the context of self-driving cars, transfer learning is used to fine-tune pre-trained CNNs for object

		detection tasks in the environment.	
2.	Traffic Police Gesture Recognition Based on Gesture Skeleton Extractor and Multichannel Dilated Graph Convolution Network Author : Xin Xiong , Haoyuan Wu , Weidong Min , Jianqiang Xu , Qiyan Fu and Chunjiang Peng (IEEE - 2021)	• GSE - extracts traffic police skeleton seq. From a video. (skeleton coordinate) • MD-GCN - take gesture skeleton seq. as input and construct a graph convolution.	• Due to the differences and chang the angle of view, the "left turn wa might be misclassified as "stop" "slow down" might be misclassifie "left turn"
3.	Pothole and Object Detection for an Autonomous Vehicle Using YOLO Author : Kavitha R , Nivetha S (IEEE - 2020)	• YOLOv3 • Camera capture the object as the input image.	• Detect the object classes like: car, pe truck, bus, pothole, wetland, traffic motorcycle.
4.	The Real-Time Detection of Traffic Participants Using YOLO Algorithm Author : Aleksa Ćorović, Velibor Ilić, Siniša Đurić, Mališa Marijan, and Bogdan Pavković (IEEE - 2021)	• YOLO. • First extract single image from video stream and resized. • Image goes to CNN and bounding box is o/p.	• It can only detect object like car, t pedestrian, traffic signs, and lights.
5.	Object Detection for Autonomous Vehicle using Single Camera with YOLOv4 and Mapping Algorithm Author : Mochammad	• YOLOv4 with CSPDarknet-53. • Mapping algorithm for location.	• It can only able to detect object animal,people,tree,Vehicle,tv.

	Sahal,Ade Oktavianus Kurniawan (IEEE - 2022)		
6.	On-road object detection using Deep Neural Network Author : Huieun Kim, Youngwan Lee, Byeounghak Yim, Eunsoo Park, Hakil Kim (IEEE - 2018)	• SSD(Single Shot MultiBox Detector). • RL(Reinforcement Learning)	• If data is obtained from tradit methods such as loop detectors, the not provide accurate on-time predic • lack of clear policies, resistanc adopting new technologies.
7.	Application of Artificial Intelligence for Traffic Data Analysis, Simulations and Adaptation Author : Daniela Koltovska Nechoska , Renata Petrevska Nechkoska and Renata Duma (ICEST - 22)	• ANN(Artificial Neural Network) • FL(Fuzzy Logic) • RL(Reinforcement Learning)	• Focused their analysis on the cu transport fields that benefit fro based technologies and especiall automated traffic data collection
8.	Prediction of Metacarpophalangeal Joint Angles and Classification of Hand Configurations Based on Ultrasound Imaging of the Forearm Author : Keshav Bimbraw, Christopher J. Nycz, Matthew J. Schueler, Ziming Zhang and Haichong K. Zhang (IEEE - 2021)	• Hand Configuration Classification. • MCP Joint Angle Estimation • SVC and CNN Model	• It can inspire research in the prom domain of utilizing ultrasound predicting both continuous and dis hand movements,which can be usef intuitive and adaptable contro physical robots and non-physical d and AR/VR interfaces.
9.	Traffic control hand signal recognition using recurrent	• The police officer is localized to a center point for	• various adverse weather conditions, as fog, rain, and snow, can degrad

	neural networks Author : Taeseung Baek and Yong-Gu Lee (Journal of Computational Design and Engineering, 2022)	directions, and the pose of the arm is detected. • RNN • The sequence generator concatenats the directions of the poses into a sequence and sends to RNN for classification.	image quality and make it hard to c the gestures.
10.	Traffic Sign Detection using Clara and Yolo in Python Author :Yogesh Valeja, Shubham Pathare , Dipen patel , Mohandas Pawar (ICACCS - 2021)	• Clara based • Feature Extraction Techniques for Object Detection in real time. • YOLO	• The paper's proposed algorithm s and monitors one or more ob motion in a variable co (simultaneously). The experim results show: the use of two dime object features and their inte distribution solving the data associ problem efficiently during monit phase.
11.	Deep Learning based Traffic Analysis of Motor Cycles in Urban City Author : Abirami T, Nivas C, Naveen R, Nithishkumar T G (IEEE - 2022)	• YOLO v3 • the use of SORT tracker • R-CNN (better than CNN)	• horizontal scaling would allow for gr effective traffic-flow optimization a the metropolis.
12.	Yolo Target Detection Algorithm in Road Scene Based on Computer Vision Author : Haomin He (IEEE - 2020)	• implementation of YOLOv4 • use of DNN(Deep Neural network)	• It is capable of detecting person, ve bicycle ,motorbike, MAP FPS.
13.	Comparison of KNN, ANN, CNN and YOLO algorithms for detecting the accurate traffic flow and build an Intelligent	• KNN algorithm • YOLO algo. • CNN	• Working on it to detect small object as in current state.

	Transportation System Author : K. Pavani, P. Sriramya (IEEE - 2021)		
14.	Road Traffic Analysis Using Unmanned Aerial Vehicle and Image Processing Algorithms Author : Carmen Gheorghe, Nicolae Filip (IEEE - 2021)	• Data from Sky (DFS) • use of DNN(Deep Neural network)	• Limitations: i. majorly legislative nature, va from state to state and whi certain countries ii. Italy or Germany are permissive with the use of dr compared to other countries, su Romania, where the lifting c unmanned aerial vehicle, or a c requires preliminary actions iii. compared to those for authorization of small comm aircraft, Drone photography filming also requires addit authorization, independent of actual flight authorization.
15.	An Assessment of Resource Exploitation Using Artificial Intelligence Based traffic control strategies Author : Vincenzo Catania, Giuseppe Ficili, Daniela Panno (IEEE - 2019)	• Integrated Control for Policing and CAC.	• The use of NNs for the CAC funct particularly effective solution. proposal, in fact, requires the NN trained with patterns which take account the effect of policing actic the various traffic flows entering an access node.

Reference

1. S. Shiravandi, M. Rahmati and F. Mahmoudi, "Hand gestures recognition using dynamic Bayesian networks," 2013 3rd Joint Conference of AI & Robotics and 5th RoboCup Iran Open International Symposium, Tehran, Iran, 2013, pp. 1-6, doi: 10.1109/RIOS.2013.6595318.
2. N. Yadav, U. Thakur, A. Poonia and R. Chandel, "Post-Crash Detection and Traffic Analysis," 2021 8th International Conference on Signal Processing and Integrated Networks (SPIN), Noida, India, 2021,

pp. 1092-1097, doi: 10.1109/SPIN52536.2021.9565964.

3. R. Kulkarni, S. Dhavalikar and S. Bangar, "Traffic Light Detection and Recognition for Self Driving Cars Using Deep Learning," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, 2018, pp. 1-4, doi: 10.1109/ICCUBEA.2018.8697819.
4. Xiong, X.; Wu, H.; Min, W.; Xu, J.; Fu, Q.; Peng, C. Traffic Police Gesture Recognition Based on Gesture Skeleton Extractor and Multichannel Dilated Graph Convolution Network. Electronics 2021, 10, 551. https://doi.org/10.3390/electronics10050551
5. K. R and N. S, "Pothole and Object Detection for an Autonomous Vehicle Using YOLO," 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2021, pp. 1585-1589, doi: 10.1109/ICICCS51141.2021.9432186.
6. A. Ćorović, V. Ilić, S. Đurić, M. Marijan and B. Pavković, "The Real-Time Detection of Traffic Participants Using YOLO Algorithm," 2018 26th Telecommunications Forum (TELFOR), Belgrade, Serbia, 2018, pp. 1-4, doi: 10.1109/TELFOR.2018.8611986.
7. M. Sahal, A. O. Kurniawan and R. E. A. Kadir, "Object Detection for Autonomous Vehicle using Single Camera with YOLOv4 and Mapping Algorithm," 2021 4th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), Yogyakarta, Indonesia, 2021, pp. 144-149, doi: 10.1109/ISRITI54043.2021.9702764.
8. H. Kim, Y. Lee, B. Yim, E. Park and H. Kim, "On-road object detection using deep neural network," 2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia), Seoul, Korea (South), 2016, pp. 1-4, doi: 10.1109/ICCE-Asia.2016.7804765.
9. P. Das, T. Ahmed and M. F. Ali, "Static Hand Gesture Recognition for American Sign Language using Deep Convolutional Neural Network," 2020 IEEE Region 10 Symposium (TENSYMP), Dhaka, Bangladesh, 2020, pp. 1762-1765, doi: 10.1109/TENSYMP50017.2020.9230772.
10. Taeseung Baek, Yong-Gu Lee, Traffic control hand signal recognition using convolution and recurrent neural networks, Journal of Computational Design and Engineering, Volume 9, Issue 2, April 2022, Pages 296–309, https://doi.org/10.1093/jcde/qwab080
11. K. Bimbraw, C. J. Nycz, M. J. Schueler, Z. Zhang and H. K. Zhang, "Prediction of Metacarpophalangeal Joint Angles and Classification of Hand Configurations Based on Ultrasound Imaging of the Forearm," 2022 International Conference on Robotics and Automation (ICRA), Philadelphia, PA, USA, 2022, pp. 91-97, doi: 10.1109/ICRA46639.2022.9812287.
12. Y. Valeja, S. Pathare, D. Patel and M. Pawar, "Traffic Sign Detection using Clara and Yolo in Python," 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2021, pp. 367-371, doi: 10.1109/ICACCS51430.2021.9442065.
13. K. Pavani and P. Sriramya, "Comparison of KNN, ANN, CNN and YOLO algorithms for detecting the accurate traffic flow and build an Intelligent Transportation System," 2022 2nd International Conference on Innovative Practices in Technology and Management (ICIPTM), Gautam Buddha Nagar, India, 2022, pp. 628-633, doi: 10.1109/ICIPTM54933.2022.9753900.
14. C. Gheorghe and N. Filip, "Road Traffic Analysis Using Unmanned Aerial Vehicle and Image Processing Algorithms," 2022 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), Cluj-Napoca, Romania, 2022, pp. 1-5, doi: 10.1109/AQTR55203.2022.9802058.
15. M. Mostafa and M. Ghantous, "A YOLO Based Approach for Traffic Light Recognition for ADAS Systems," 2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC), Cairo, Egypt, 2022, pp. 225-229, doi: 10.1109/MIUCC55081.2022.9781682.

16. L. H. Duong et al., "ODAR: A Lightweight Object Detection Framework for Autonomous Driving Robots," 2021 Digital Image Computing: Techniques and Applications (DICTA), Gold Coast, Australia, 2021, pp. 01-08, doi: 10.1109/DICTA52665.2021.9647256.
17. H. He, "Yolo Target Detection Algorithm in Road Scene Based on Computer Vision," 2022 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC), Dalian, China, 2022, pp. 1111-1114, doi: 10.1109/IPEC54454.2022.9777571.