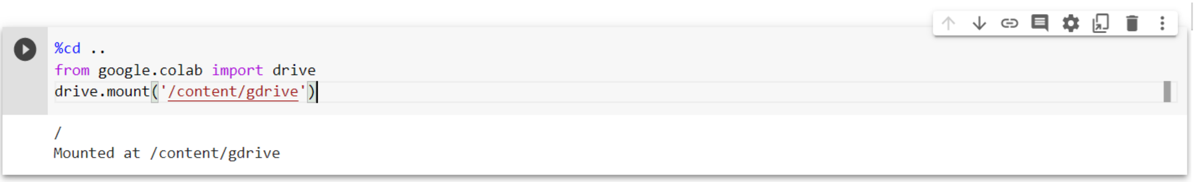
Nafith

Classification task

Muna Alsaber

❖ Uploading all required data on google drive

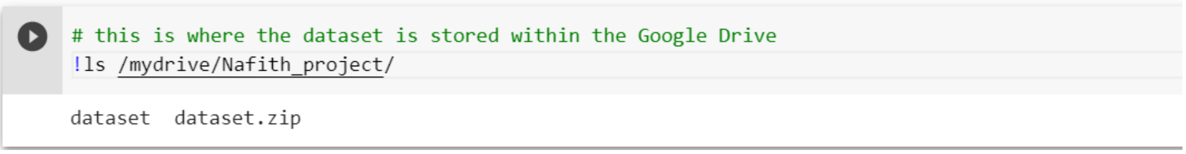
- Mount Google drive and Extract the Dataset



```
%cd ..
from google.colab import drive
drive.mount('/content/gdrive')
```

```
/
Mounted at /content/gdrive
```

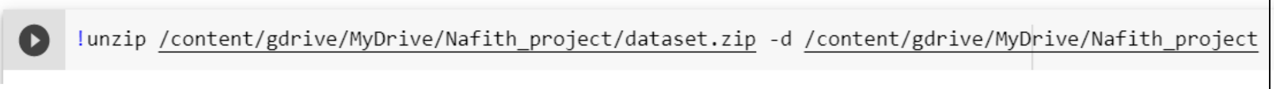
- Where the dataset is stored



```
# this is where the dataset is stored within the Google Drive
!ls /mydrive/Nafith_project/
```

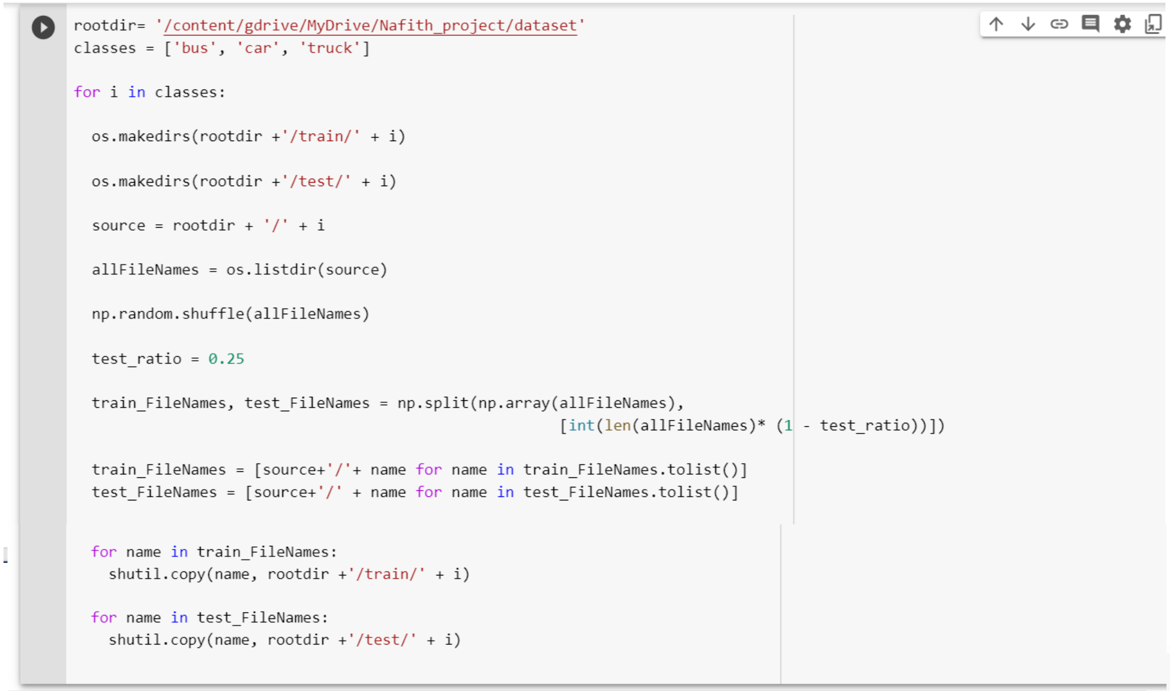
```
dataset  dataset.zip
```

- Extract the compressed dataset file

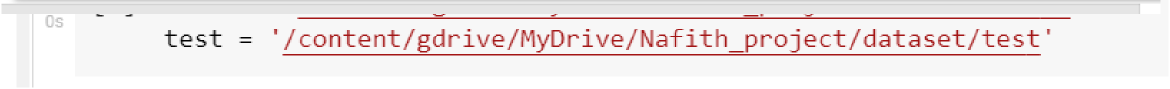


```
!unzip /content/gdrive/MyDrive/Nafith_project/dataset.zip -d /content/gdrive/MyDrive/Nafith_project
```

❖ Splitting dataset into train and test

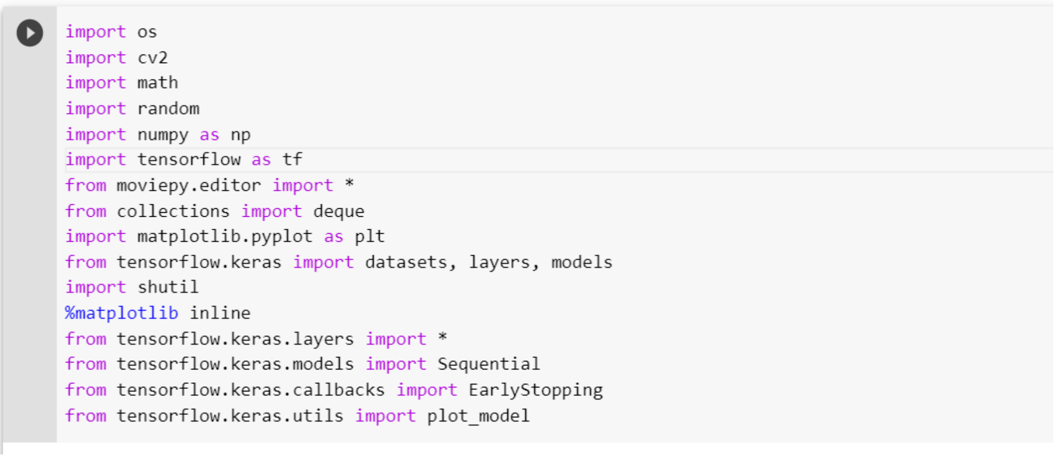


```
rootdir= '/content/gdrive/MyDrive/Nafith_project/dataset'
classes = ['bus', 'car', 'truck']

for i in classes:

  os.makedirs(rootdir +'/train/' + i)

  os.makedirs(rootdir +'/test/' + i)

  source = rootdir + '/' + i

  allFileNames = os.listdir(source)

  np.random.shuffle(allFileNames)

  test_ratio = 0.25

  train_FileNames, test_FileNames = np.split(np.array(allFileNames),
                                                      [int(len(allFileNames)* (1 - test_ratio))])

  train_FileNames = [source+'/'+ name for name in train_FileNames.tolist()]
  test_FileNames = [source+'/' + name for name in test_FileNames.tolist()]


  for name in train_FileNames:
    shutil.copy(name, rootdir +'/train/' + i)

  for name in test_FileNames:
    shutil.copy(name, rootdir +'/test/' + i)
```



```
    test = '/content/gdrive/MyDrive/Nafith_project/dataset/test'
```

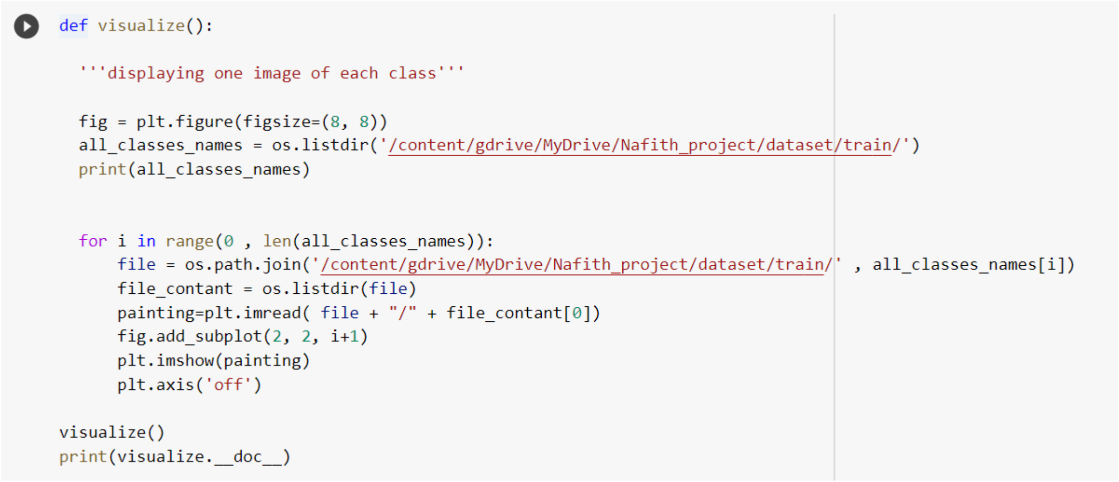
❖ Importing all important libraries



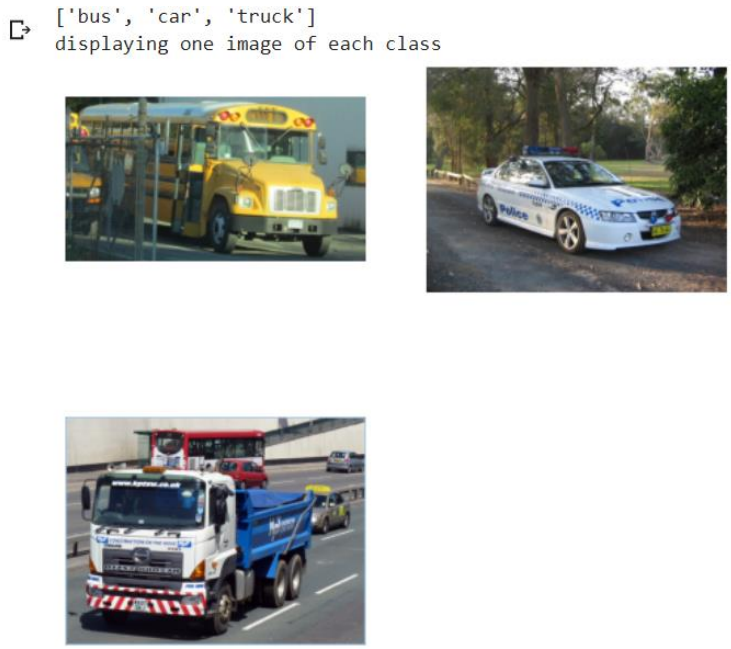
```
import os
import cv2
import math
import random
import numpy as np
import tensorflow as tf
from moviepy.editor import *
from collections import deque
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models
import shutil
%matplotlib inline
from tensorflow.keras.layers import *
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import plot_model
```

❖ Visualizing the Data

- displaying one image of each class from the train file



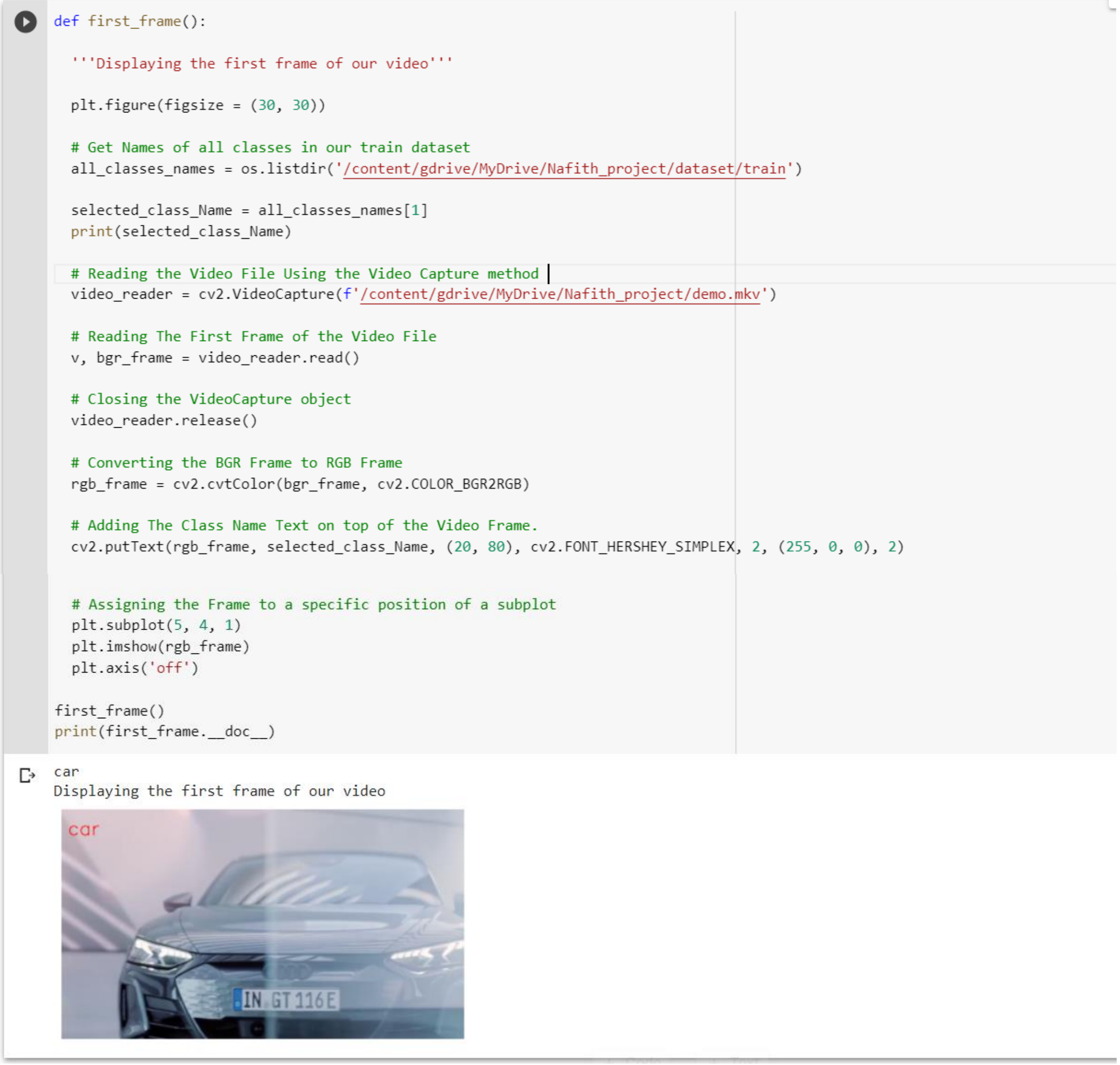
```
def visualize():

  '''displaying one image of each class'''

  fig = plt.figure(figsize=(8, 8))
  all_classes_names = os.listdir('/content/gdrive/MyDrive/Nafith_project/dataset/train/')
  print(all_classes_names)


  for i in range(0 , len(all_classes_names)):
      file = os.path.join('/content/gdrive/MyDrive/Nafith_project/dataset/train/' , all_classes_names[i])
      file_contant = os.listdir(file)
      painting=plt.imread( file + "/" + file_contant[0])
      fig.add_subplot(2, 2, i+1)
      plt.imshow(painting)
      plt.axis('off')

visualize()
print(visualize.__doc__)
```



```
['bus', 'car', 'truck']
displaying one image of each class
```

di

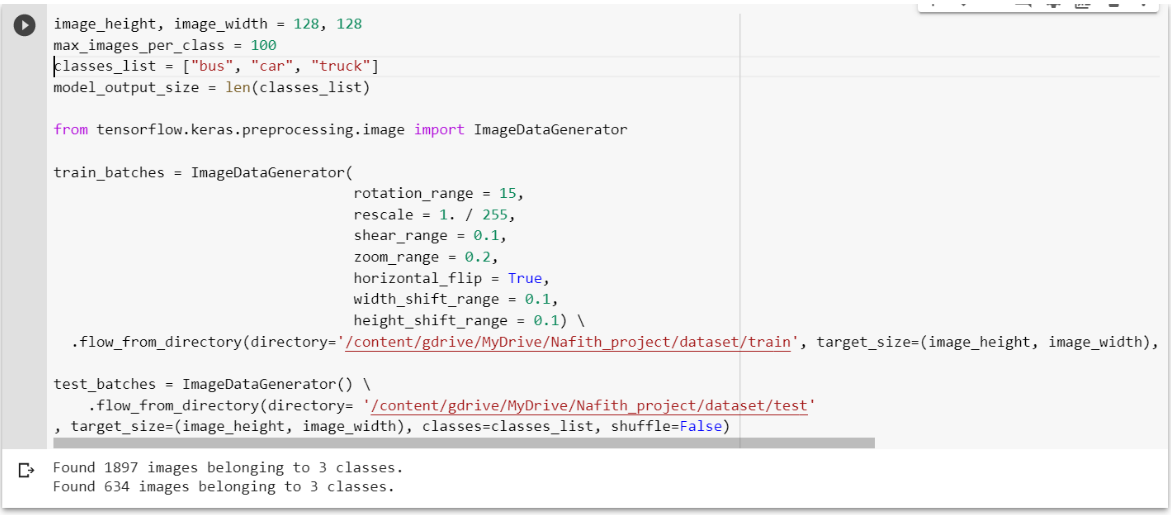
- Displaying the first frame of our video



```
def first_frame():

  '''Displaying the first frame of our video'''

  plt.figure(figsize = (30, 30))

  # Get Names of all classes in our train dataset
  all_classes_names = os.listdir('/content/gdrive/MyDrive/Nafith_project/dataset/train')

  selected_class_Name = all_classes_names[1]
  print(selected_class_Name)

  # Reading the Video File Using the Video Capture method
  video_reader = cv2.VideoCapture(f'/content/gdrive/MyDrive/Nafith_project/demo.mkv')

  # Reading The First Frame of the Video File
  v, bgr_frame = video_reader.read()

  # Closing the VideoCapture object
  video_reader.release()

  # Converting the BGR Frame to RGB Frame
  rgb_frame = cv2.cvtColor(bgr_frame, cv2.COLOR_BGR2RGB)

  # Adding The Class Name Text on top of the Video Frame.
  cv2.putText(rgb_frame, selected_class_Name, (20, 80), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 0, 0), 2)


  # Assigning the Frame to a specific position of a subplot
  plt.subplot(5, 4, 1)
  plt.imshow(rgb_frame)
  plt.axis('off')

first_frame()
print(first_frame.__doc__)
```

```
car
Displaying the first frame of our video
```

❖ Preprocessing the Dataset

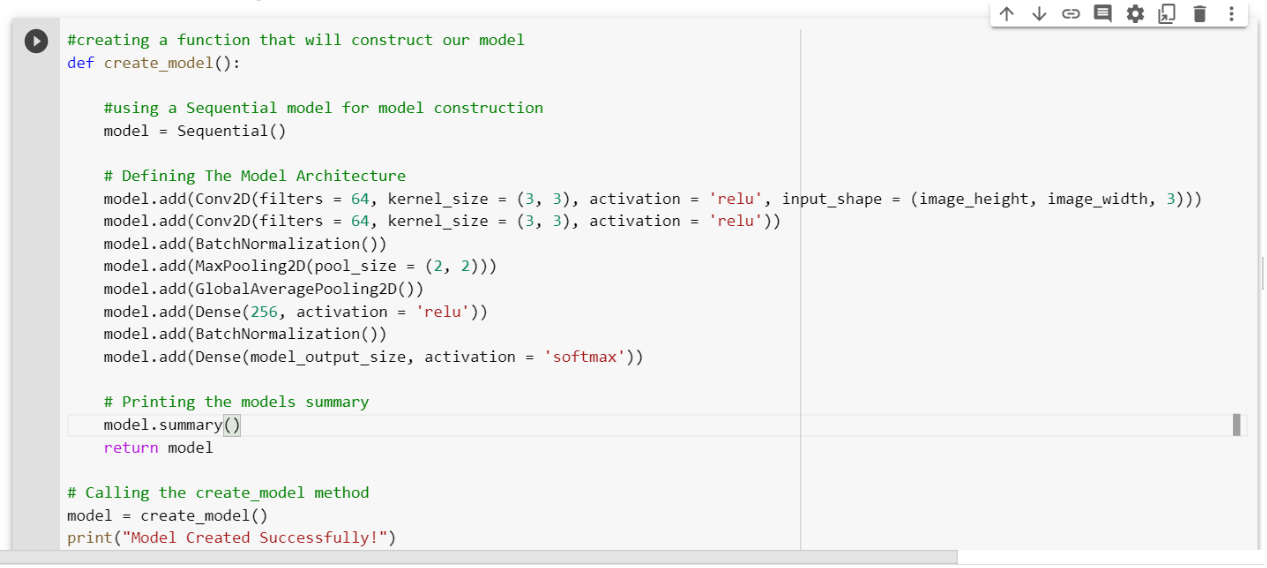


```
image_height, image_width = 128, 128
max_images_per_class = 100
classes_list = ["bus", "car", "truck"]
model_output_size = len(classes_list)

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_batches = ImageDataGenerator(
                                 rotation_range = 15,
                                 rescale = 1. / 255,
                                 shear_range = 0.1,
                                 zoom_range = 0.2,
                                 horizontal_flip = True,
                                 width_shift_range = 0.1,
                                 height_shift_range = 0.1) \
  .flow_from_directory(directory='/content/gdrive/MyDrive/Nafith_project/dataset/train', target_size=(image_height, image_width),

test_batches = ImageDataGenerator() \
    .flow_from_directory(directory= '/content/gdrive/MyDrive/Nafith_project/dataset/test'
, target_size=(image_height, image_width), classes=classes_list, shuffle=False)
```

```
Found 1897 images belonging to 3 classes.
Found 634 images belonging to 3 classes.
```

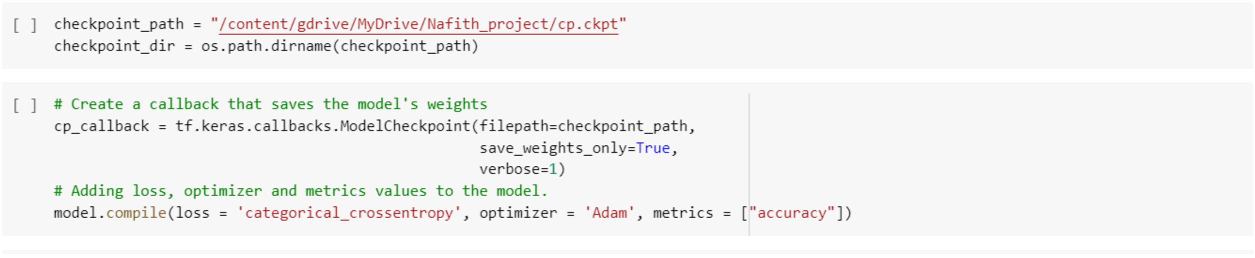
❖ The CNN Model

- Building the Model Architecture



```
#creating a function that will construct our model
def create_model():

    #using a Sequential model for model construction
    model = Sequential()

    # Defining The Model Architecture
    model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu', input_shape = (image_height, image_width, 3)))
    model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size = (2, 2)))
    model.add(GlobalAveragePooling2D())
    model.add(Dense(256, activation = 'relu'))
    model.add(BatchNormalization())
    model.add(Dense(model_output_size, activation = 'softmax'))

    # Printing the models summary
    model.summary()
    return model

# Calling the create_model method
model = create_model()
print("Model Created Successfully!")
```

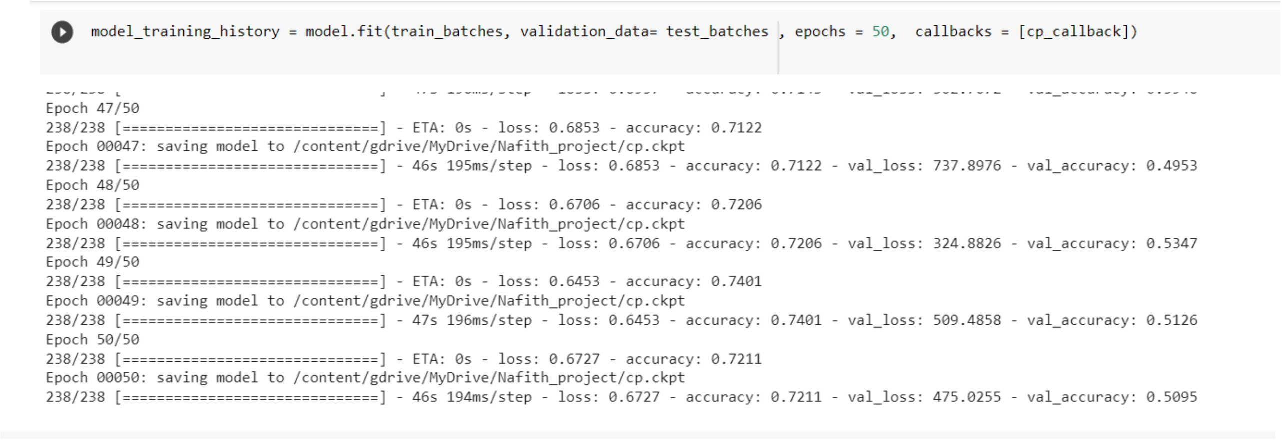
- checkpoint to Save the mode



```
[ ] checkpoint_path = "/content/gdrive/MyDrive/Nafith_project/cp.ckpt"
    checkpoint_dir = os.path.dirname(checkpoint_path)
```

```
[ ] # Create a callback that saves the model's weights
    cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                     save_weights_only=True,
                                                     verbose=1)
    # Adding loss, optimizer and metrics values to the model.
    model.compile(loss = 'categorical_crossentropy', optimizer = 'Adam', metrics = ["accuracy"])
```

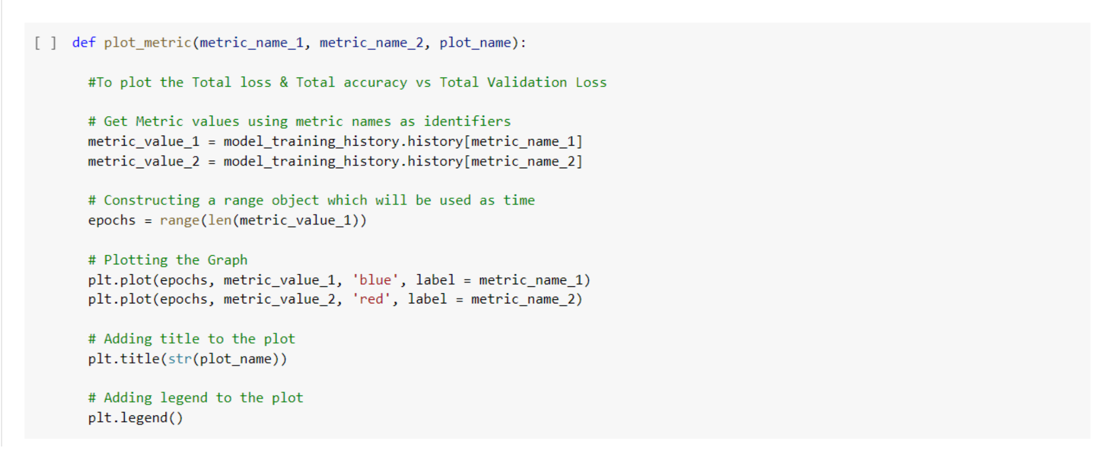
- Training process



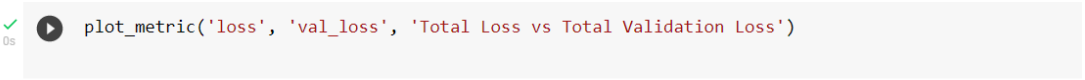
```
model_training_history = model.fit(train_batches, validation_data= test_batches , epochs = 50,  callbacks = [cp_callback])
```

```
Epoch 47/50
238/238 [==============================] - ETA: 0s - loss: 0.6853 - accuracy: 0.7122
Epoch 00047: saving model to /content/gdrive/MyDrive/Nafith_project/cp.ckpt
238/238 [==============================] - 46s 195ms/step - loss: 0.6853 - accuracy: 0.7122 - val_loss: 737.8976 - val_accuracy: 0.4953
Epoch 48/50
238/238 [==============================] - ETA: 0s - loss: 0.6706 - accuracy: 0.7206
Epoch 00048: saving model to /content/gdrive/MyDrive/Nafith_project/cp.ckpt
238/238 [==============================] - 46s 195ms/step - loss: 0.6706 - accuracy: 0.7206 - val_loss: 324.8826 - val_accuracy: 0.5347
Epoch 49/50
238/238 [==============================] - ETA: 0s - loss: 0.6453 - accuracy: 0.7401
Epoch 00049: saving model to /content/gdrive/MyDrive/Nafith_project/cp.ckpt
238/238 [==============================] - 47s 196ms/step - loss: 0.6453 - accuracy: 0.7401 - val_loss: 509.4858 - val_accuracy: 0.5126
Epoch 50/50
238/238 [==============================] - ETA: 0s - loss: 0.6727 - accuracy: 0.7211
Epoch 00050: saving model to /content/gdrive/MyDrive/Nafith_project/cp.ckpt
238/238 [==============================] - 46s 194ms/step - loss: 0.6727 - accuracy: 0.7211 - val_loss: 475.0255 - val_accuracy: 0.5095
```

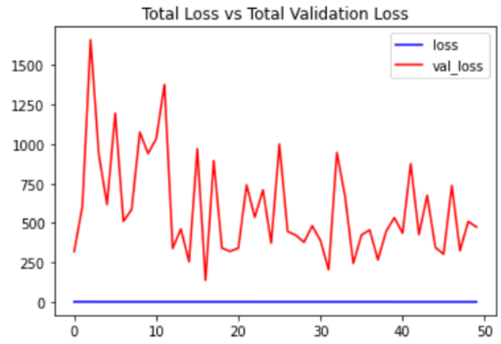
❖ Plotting the model loss and accuracy versus total validation loss

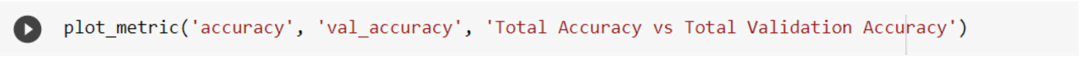


```
def plot_metric(metric_name_1, metric_name_2, plot_name):

    #To plot the Total loss & Total accuracy vs Total Validation Loss

    # Get Metric values using metric names as identifiers
    metric_value_1 = model_training_history.history[metric_name_1]
    metric_value_2 = model_training_history.history[metric_name_2]

    # Constructing a range object which will be used as time
    epochs = range(len(metric_value_1))

    # Plotting the Graph
    plt.plot(epochs, metric_value_1, 'blue', label = metric_name_1)
    plt.plot(epochs, metric_value_2, 'red', label = metric_name_2)

    # Adding title to the plot
    plt.title(str(plot_name))

    # Adding legend to the plot
    plt.legend()
```

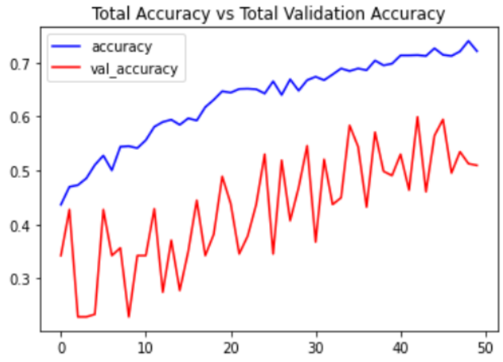


```
plot_metric('loss', 'val_loss', 'Total Loss vs Total Validation Loss')
```

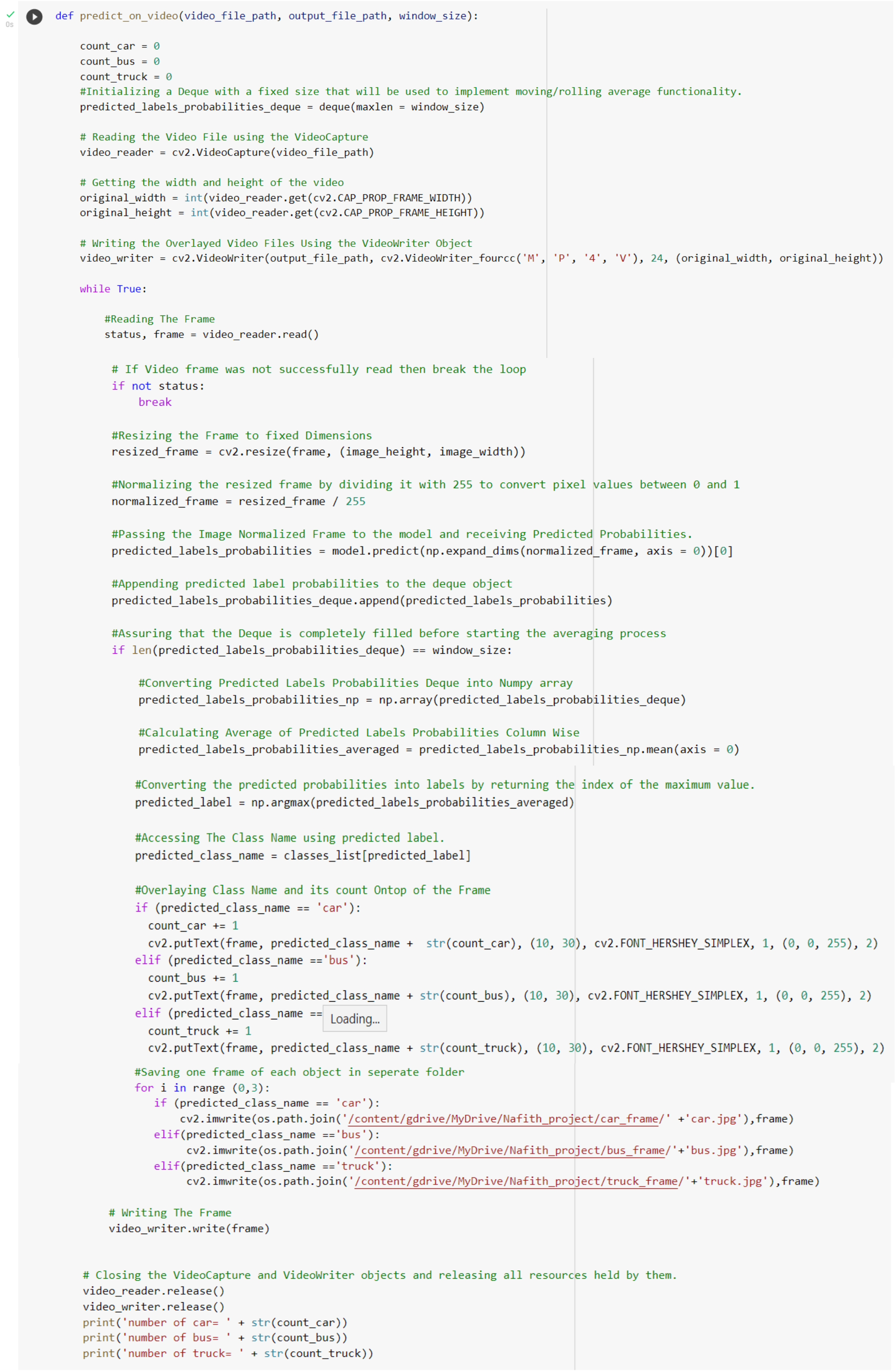




```
plot_metric('accuracy', 'val_accuracy', 'Total Accuracy vs Total Validation Accuracy')
```

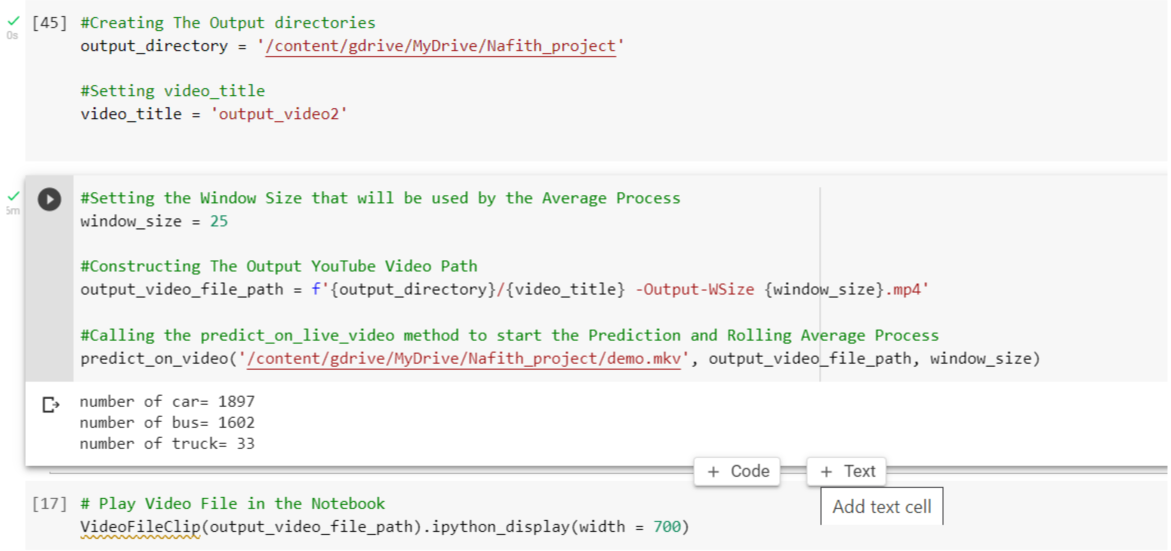


❖ Video classification and analysis



```
def predict_on_video(video_file_path, output_file_path, window_size):

    count_car = 0
    count_bus = 0
    count_truck = 0
    #Initializing a Deque with a fixed size that will be used to implement moving/rolling average functionality.
    predicted_labels_probabilities_deque = deque(maxlen = window_size)

    # Reading the Video File using the VideoCapture
    video_reader = cv2.VideoCapture(video_file_path)

    # Getting the width and height of the video
    original_width = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))
    original_height = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # Writing the Overlayed Video Files Using the VideoWriter Object
    video_writer = cv2.VideoWriter(output_file_path, cv2.VideoWriter_fourcc('M', 'P', '4', 'V'), 24, (original_width, original_height))

    while True:

        #Reading The Frame
        status, frame = video_reader.read()

        # If Video frame was not successfully read then break the loop
        if not status:
            break

        #Resizing the Frame to fixed Dimensions
        resized_frame = cv2.resize(frame, (image_height, image_width))

        #Normalizing the resized frame by dividing it with 255 to convert pixel values between 0 and 1
        normalized_frame = resized_frame / 255

        #Passing the Image Normalized Frame to the model and receiving Predicted Probabilities.
        predicted_labels_probabilities = model.predict(np.expand_dims(normalized_frame, axis = 0))[0]

        #Appending predicted label probabilities to the deque object
        predicted_labels_probabilities_deque.append(predicted_labels_probabilities)

        #Assuring that the Deque is completely filled before starting the averaging process
        if len(predicted_labels_probabilities_deque) == window_size:

            #Converting Predicted Labels Probabilities Deque into Numpy array
            predicted_labels_probabilities_np = np.array(predicted_labels_probabilities_deque)

            #Calculating Average of Predicted Labels Probabilities Column Wise
            predicted_labels_probabilities_averaged = predicted_labels_probabilities_np.mean(axis = 0)

            #Converting the predicted probabilities into labels by returning the index of the maximum value.
            predicted_label = np.argmax(predicted_labels_probabilities_averaged)

            #Accessing The Class Name using predicted label.
            predicted_class_name = classes_list[predicted_label]

            #Overlaying Class Name and its count Ontop of the Frame
            if (predicted_class_name == 'car'):
              count_car += 1
              cv2.putText(frame, predicted_class_name +  str(count_car), (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
            elif (predicted_class_name =='bus'):
              count_bus += 1
              cv2.putText(frame, predicted_class_name + str(count_bus), (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
            elif (predicted_class_name == Loading...
              count_truck += 1
              cv2.putText(frame, predicted_class_name + str(count_truck), (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
            #Saving one frame of each object in seperate folder
            for i in range (0,3):
               if (predicted_class_name == 'car'):
                   cv2.imwrite(os.path.join('/content/gdrive/MyDrive/Nafith_project/car_frame/' +'car.jpg'),frame)
               elif(predicted_class_name =='bus'):
                    cv2.imwrite(os.path.join('/content/gdrive/MyDrive/Nafith_project/bus_frame/'+'bus.jpg'),frame)
               elif(predicted_class_name =='truck'):
                    cv2.imwrite(os.path.join('/content/gdrive/MyDrive/Nafith_project/truck_frame/'+'truck.jpg'),frame)

        # Writing The Frame
        video_writer.write(frame)


    # Closing the VideoCapture and VideoWriter objects and releasing all resources held by them.
    video_reader.release()
    video_writer.release()
    print('number of car= ' + str(count_car))
    print('number of bus= ' + str(count_bus))
    print('number of truck= ' + str(count_truck))
```

❖ Applying predicting method on our video and download output video



```
[45] #Creating The Output directories
     output_directory = '/content/gdrive/MyDrive/Nafith_project'

     #Setting video_title
     video_title = 'output_video2'
```

```
#Setting the Window Size that will be used by the Average Process
window_size = 25

#Constructing The Output YouTube Video Path
output_video_file_path = f'{output_directory}/{video_title} -Output-WSize {window_size}.mp4'

#Calling the predict_on_live_video method to start the Prediction and Rolling Average Process
predict_on_video('/content/gdrive/MyDrive/Nafith_project/demo.mkv', output_video_file_path, window_size)
```

```
number of car= 1897
number of bus= 1602
number of truck= 33
```

```
[17] # Play Video File in the Notebook
     VideoFileClip(output_video_file_path).ipython_display(width = 700)
```