report

by R R

1. INTRODUCTION

The project we have chosen is to help the hearing and speech impaired people by creating a deep-learning based Sign Language Interpreter which would help in communicating their language to us in a more efficient and comfortable way. The work comprises of collecting video dataset from more than 12 people and for 50 signs we would be building a model and training it and making it learn the signs we have collected all the signs are dynamic and not static which makes our work even more unique and practical. We would be dividing the video into frames and using different CNN layers optimizers and activators to extract features from the video as model training is not possible directly on the videos itself. Then we deploy our model on the cloud and make a web app/ mobile app so that everyone can use it.

Basic Objectives:

Collecting data of dynamic videos

Developing a model

Training the model

Testing and predicting

A web/ mobile based sign language interpreter

2. PROBLEM STATEMENT

In the world roughly 15-20% of the population is suffering from hearing or speech impairment and in India the World Health Organization says 2.42 million people are suffering from it that’s a lot of people in the Indian society.

There is an unknown fact that there is not just a single sign language used across the globe but there are different sign languages in different parts of the country some even differ from state to state. The number of sign languages are more than 300 and counting till now.

The problems these people face is immense as there is no proper way for them to communicate in the public places, railway stations, etc. The only option most of them use is having to carry a translator with them which is obviously not fusible for all and are very expensive and would have to wait for them to arrive and have to adjust the timetable according to their arrival.

If you look for schools the number of schools and colleges available for them is mere negligible for the population they are and are located in far away places and not in every state of the country which makes them inaccessible to lead an independent life. The number of deaf and dumb schools in India are just 388 all in cities and none in the villages which it an even bigger problem.

The online resources available for them are mostly limited and irrelevant from what they are looking for but, one few online platform does exist which provides education to them is:

https://indiansignlanguage.org/

http://www.islrtc.nic.in/ (Indian sign language research and training center)

There are just two resources available for them which is just not fair.

3. EXISTING SYSTEMS

3.1 Introduction

There are number of mobile applications and webapps available for ASL(American Sign Language) but when it comes to ISL (Indian Sign Language) there is only one available in the Indian market till now that is GnoSys.

3.2 Existing Software

GnoSys:

GnoSys is the only Indian Sign Language Interpreter available on web till now used across India. They are in collaboration with the NAD (National Deaf Association) for collection of sign and data for deploying the model.

This is an mobile application which make is accessible anywhere they want its inexpensive and offers high quality outputs making it the best and the only choice till now in the market.

They developers use Neural network with AI (Artificial Intelligence) and computer vision to record the signs the person is trying to say and then translates it and gives output as a audio so that everyone can understand.

3.3 Difference in our Project

Our method is to collect data of 50 dynamic signs from more than 12 and each sign will be performed 50 times which makes it 50x30x[]= 10,000 videos of dataset of even more.

Each sign having 200 videos. Then we would be making our own deep learning model and deploying it and would perform our training and testing on it to give us a good enough output as we desire it to be.

4. PROBLEM ANALYSIS

4.1 Objectives

A. Collection and development of dataset

B. Propose a model for sign language recognition system

C. Training the model

D. Testing the model

E. Creating a web/mobile based sign language interpreter

4.2 Scope

Currently in the market there is only one app dedicated to interpreting the Indian Sign language to the people around them. There are number of options available in other countries but that's not the case here so we plan to make this project opensource so that students and all the tech enthusiasts can contribute to this work and make the lives of the needy more independent and self-centered

.

In the recent times there are a number of research work done for which approach is best to deploy the models and what kind of models gives us what kind of accuracy. The best model could be picked up from there and the work could be expanded more making it give much better results when we have R&D done by researchers.

Making this service free would be a stroke of luck in disguise for them and this would be first ever Indian Sign Language Interpreter app developed in India by Indian as the other services and applications available were developed in foreign soils.

5. SOFTWARE REQUIREMENT ANALYSIS

This project is developed on Google Collaboratory is which allows anyone to use it to deploy machine learning or deep learning model to train and test them for free of cost.

It allows python code to be run through the browser with out downloading any extensions or software's on the system. It runs irrespective of your system configuration; the only thing required is a good internet connectivity and a browser to access it

5.1 User Profile

We are targeting people from 3 – 100< age group; it is not based on gender. The user needs to install our app or access the web application which will be available on Google Play Store or on the internet. After installing the app, the user can use the app. There is no need of logging in or signing up for the App or website. The user would express his/ her signs in the camera and it would be translation it into speech.

5.2 Assumptions / Dependencies

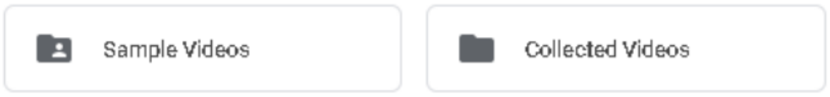
- Good internet connection
- Access to google collaboratory
- Access to play store/ website
- The code of model will be written in python
- The app will be written in XML and python.
- Deploying the model on a cloud server
- Linking of external websites is required for the app as different parts of the module works through the WebView method.

5.3 Functional & Non-Functional Requirements

- Active internet connection
- Knowledge of google colab
- Knowledge of deep learning
- Python GPU3 engine, 12GB RAM

6. DESIGN

6.1 Collection and creation of dataset

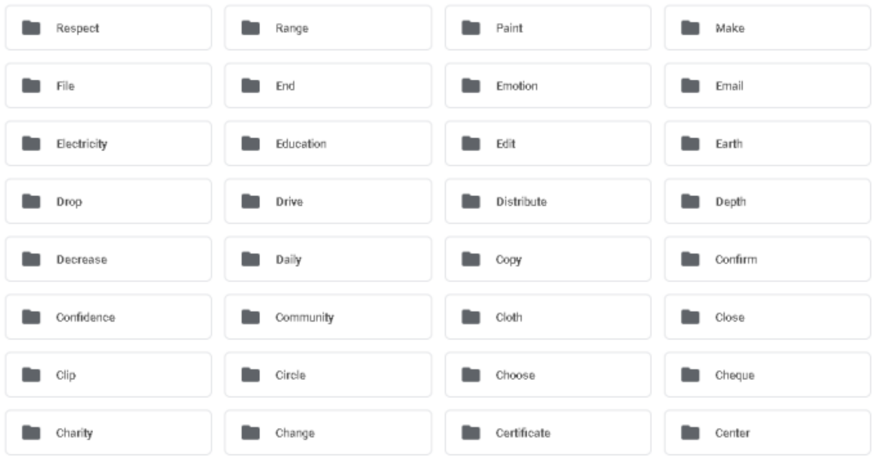


Sample Videos

Collected Videos

Fig.1. Sample videos and collected videos

6.2 Segregating them into folders



Respect	Range	Paint	Make
File	End	Emotion	Email
Electricity	Education	Edit	Earth
Drop	Drive	Distribute	Depth
Decrease	Daily	Copy	Confirm
Confidence	Community	Cloth	Close
Clip	Circle	Choose	Cheque
Charity	Change	Certificate	Center

Fig.2.Dataset arranged into folders

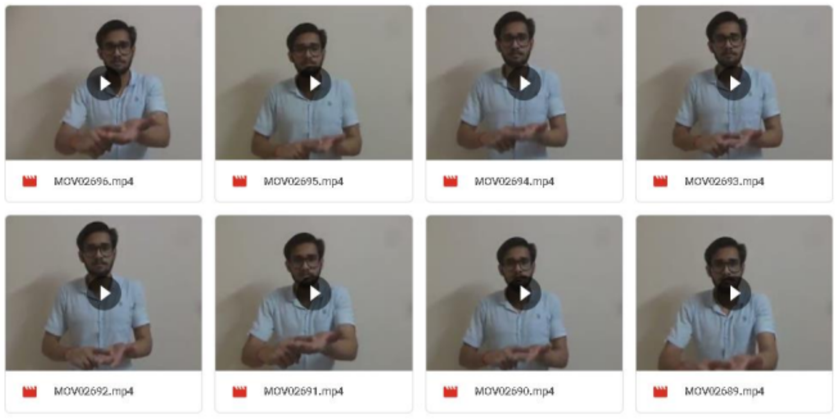


Fig.3. Folder contains files like this

6.3 Creating our model

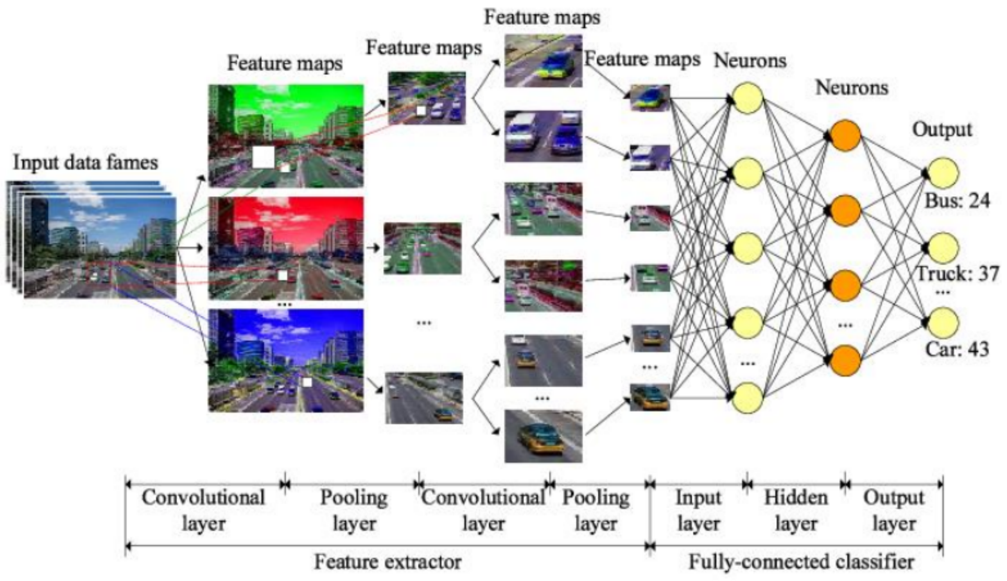


Fig.4. Example of model

6.4 Training and Testing

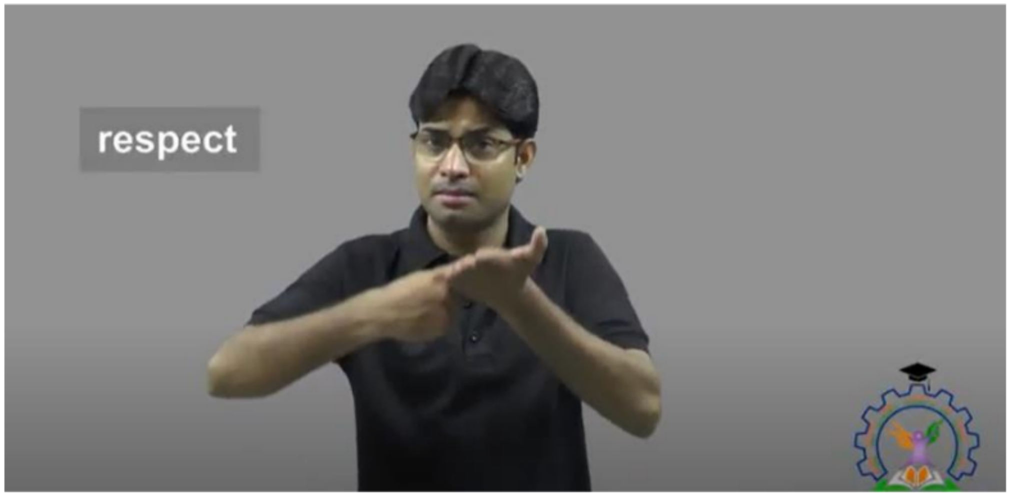


Fig.5 . Testing sample

7. IMPLEMENATION

1. The dataset is collected from different people for 50 signs and each sign repeated for 30 times
2. Then the data is segregated into specific folders
3. Then the data is uploaded onto google drive to make it accessible to all.
4. Using Google colab
5. Loading the dataset
6. Listing the dataset
7. Importing libraries
8. Read and preprocess the list of class
9. Resizing and normalizing the dataset
10. Creating dataset
11. Splitting into training and test sets
12. Creating model
13. Training model
14. Getting model accuracy
15. Saving the model
16. Plotting the loss and accuracy curves
17. Making predictions

8. TESTING.

8.1 Functional Testing

Each module is tested multiple times using the google colab python3 GPU which gives us 12.69GB RAM and 107.72GB ROM but having an additional GPU and accelerator would boost the performance even more

8.2 Structural Testing

The internal structure has succeeded in producing expected outcome and has not exhibited any kind of glitch during manual structural testing.

8.3 Levels of Testing

Type of testing: Manual

a) Module-wise testing: Success

Testing outcome: Each module is individually tested for their functionality and performance and the testing completed with no glitches and bugs.

b) Integrated-module testing: Success

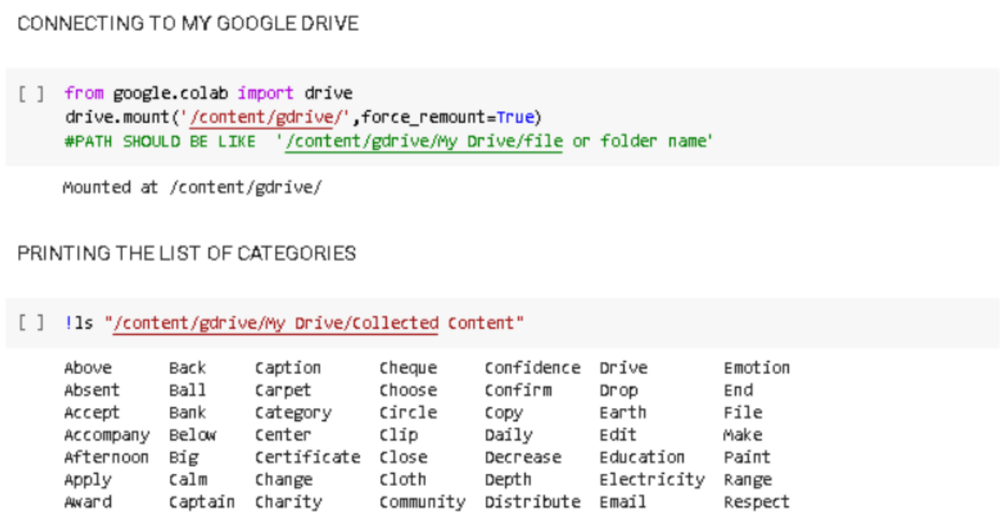
Testing outcome: All the modules are seamlessly working well with all the integrated modules with no glitches.

c) Model testing: Success

Testing outcome: A variation is seen every time the code is run on collab due to resources but if run on a physical hardware the code runs fine and gives the expected outcomes

9. SOURCECODE AND SYSTEM SNAPSHOTS

1. Loading the dataset
2. Listing the dataset



CONNECTING TO MY GOOGLE DRIVE

```
[ ] from google.colab import drive
    drive.mount('/content/gdrive/',force_remount=True)
    #PATH SHOULD BE LIKE  '/content/gdrive/My Drive/file or folder name'
```

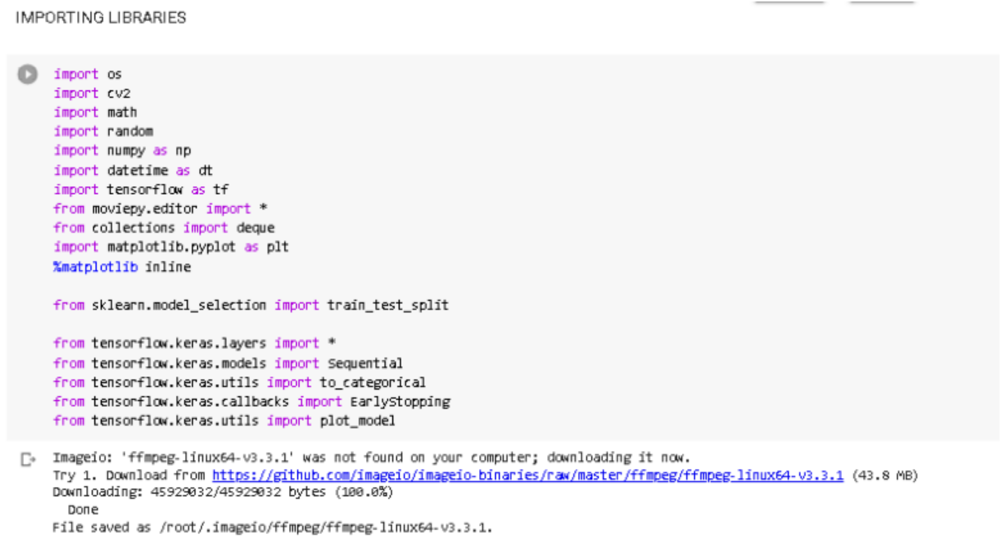
```
Mounted at /content/gdrive/
```

PRINTING THE LIST OF CATEGORIES

```
[ ] !ls "/content/gdrive/My Drive/Collected Content"
```

```
Above      Back     Caption     Cheque     Confidence  Drive        Emotion
Absent     Ball     Carpet      Choose     Confirm     Drop         End
Accept     Bank     Category    Circle     Copy        Earth        File
Accompany  Below    Center      Clip       Daily       Edit         Make
Afternoon  Big      Certificate Close      Decrease    Education    Paint
Apply      Calm     Change      Cloth      Depth       Electricity  Range
Award      Captain  Charity     Community  Distribute  Email        Respect
```

3. Importing libraries

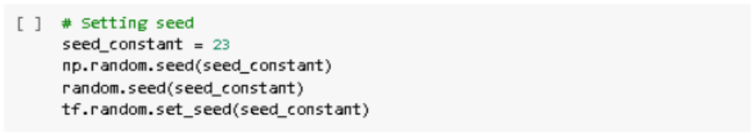


IMPORTING LIBRARIES

```
import os
import cv2
import math
import random
import numpy as np
import datetime as dt
import tensorflow as tf
from moviepy.editor import *
from collections import deque
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split

from tensorflow.keras.layers import *
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import plot_model
```

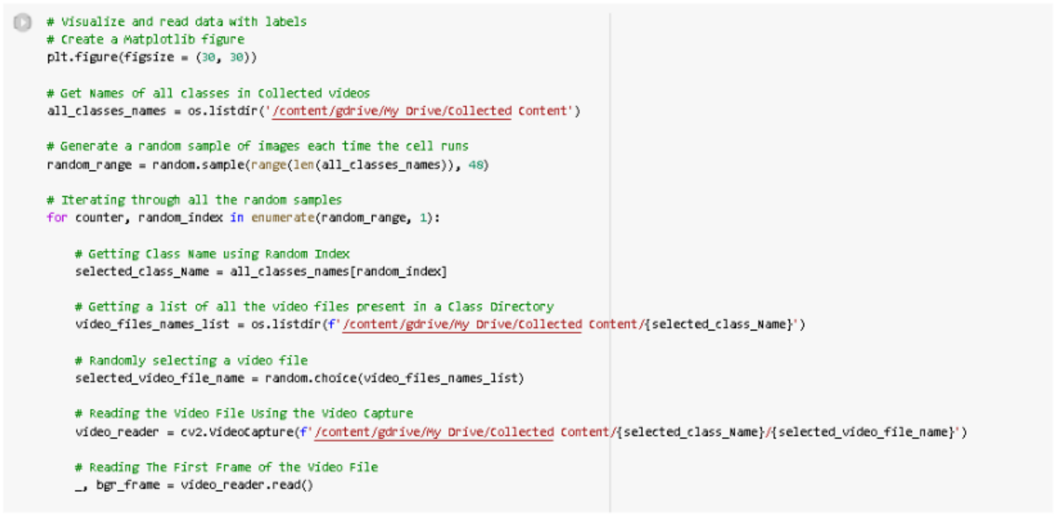
```
Imageio: 'ffmpeg-linux64-v3.3.1' was not found on your computer; downloading it now.
Try 1. Download from https://github.com/imageio/imageio-binaries/raw/master/ffmpeg/ffmpeg-linux64-v3.3.1 (43.8 MB)
Downloading: 45929032/45929032 bytes (100.0%)
  Done
File saved as /root/.imageio/ffmpeg/ffmpeg-linux64-v3.3.1.
```

4. Setting seed



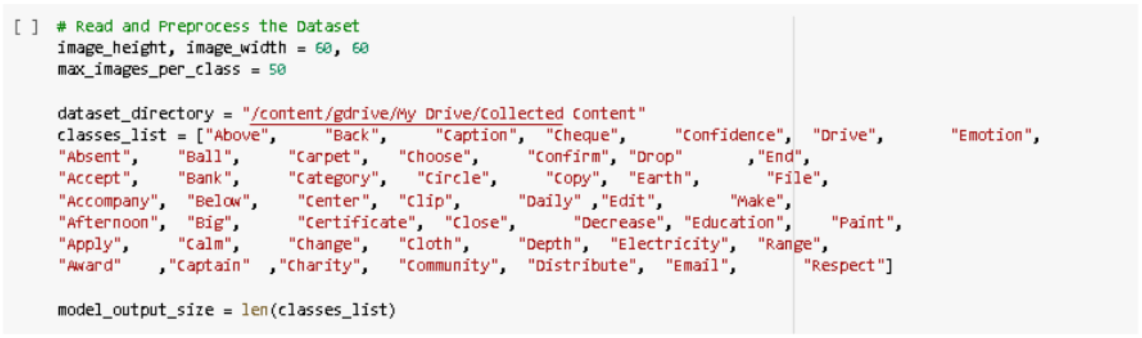
```
[ ]  # Setting seed
     seed_constant = 23
     np.random.seed(seed_constant)
     random.seed(seed_constant)
     tf.random.set_seed(seed_constant)
```

5. Creating grid of images with labels on top



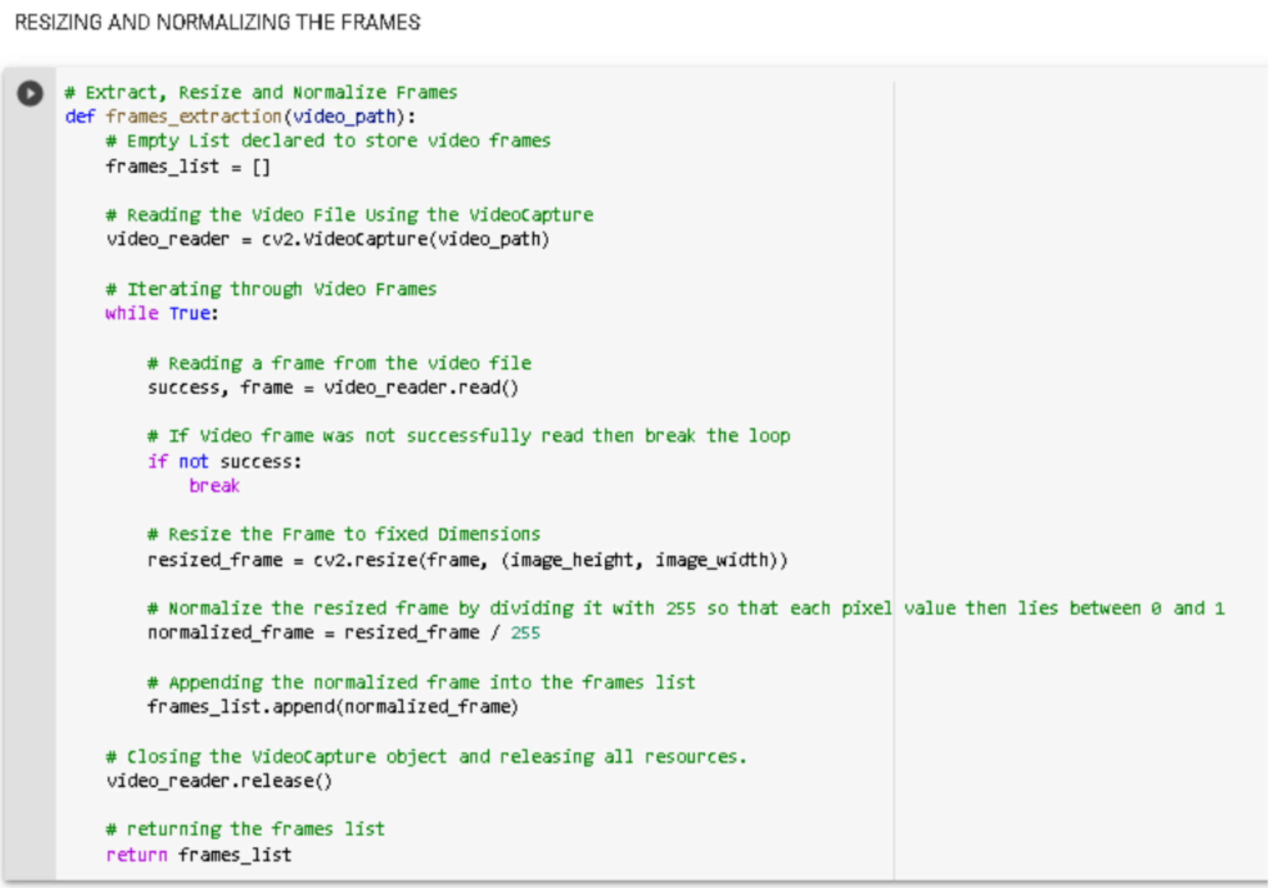
```
# Visualize and read data with labels
# Create a Matplotlib figure
plt.figure(figsize = (30, 30))

# Get Names of all classes in Collected videos
all_classes_names = os.listdir('/content/gdrive/My Drive/Collected Content')

# Generate a random sample of images each time the cell runs
random_range = random.sample(range(len(all_classes_names)), 40)

# Iterating through all the random samples
for counter, random_index in enumerate(random_range, 1):

    # Getting Class Name using Random Index
    selected_class_Name = all_classes_names[random_index]

    # Getting a list of all the video files present in a Class Directory
    video_files_names_list = os.listdir(f'/content/gdrive/My Drive/Collected Content/{selected_class_Name}')

    # Randomly selecting a video file
    selected_video_file_name = random.choice(video_files_names_list)

    # Reading the Video File Using the Video Capture
    video_reader = cv2.VideoCapture(f'/content/gdrive/My Drive/Collected Content/{selected_class_Name}/{selected_video_file_name}')

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

6. Read and preprocess the data



```
[ ] # Read and Preprocess the Dataset
    image_height, image_width = 60, 60
    max_images_per_class = 50

    dataset_directory = "/content/gdrive/My Drive/Collected Content"
    classes_list = ["Above",     "Back",     "Caption",  "Cheque",     "Confidence",  "Drive",       "Emotion",
    "Absent",    "Ball",     "Carpet",   "Choose",     "Confirm", "Drop"       ,"End",
    "Accept",    "Bank",     "Category",   "Circle",     "Copy",  "Earth",       "File",
    "Accompany",  "Below",    "Center",  "Clip",       "Daily" ,"Edit",        "Make",
    "Afternoon",  "Big",      "Certificate",  "Close",      "Decrease", "Education",    "Paint",
    "Apply",     "Calm",     "Change",   "Cloth",      "Depth",  "Electricity",  "Range",
    "Award"    ,"Captain"  ,"Charity",   "Community",  "Distribute",  "Email",       "Respect"]

    model_output_size = len(classes_list)
```

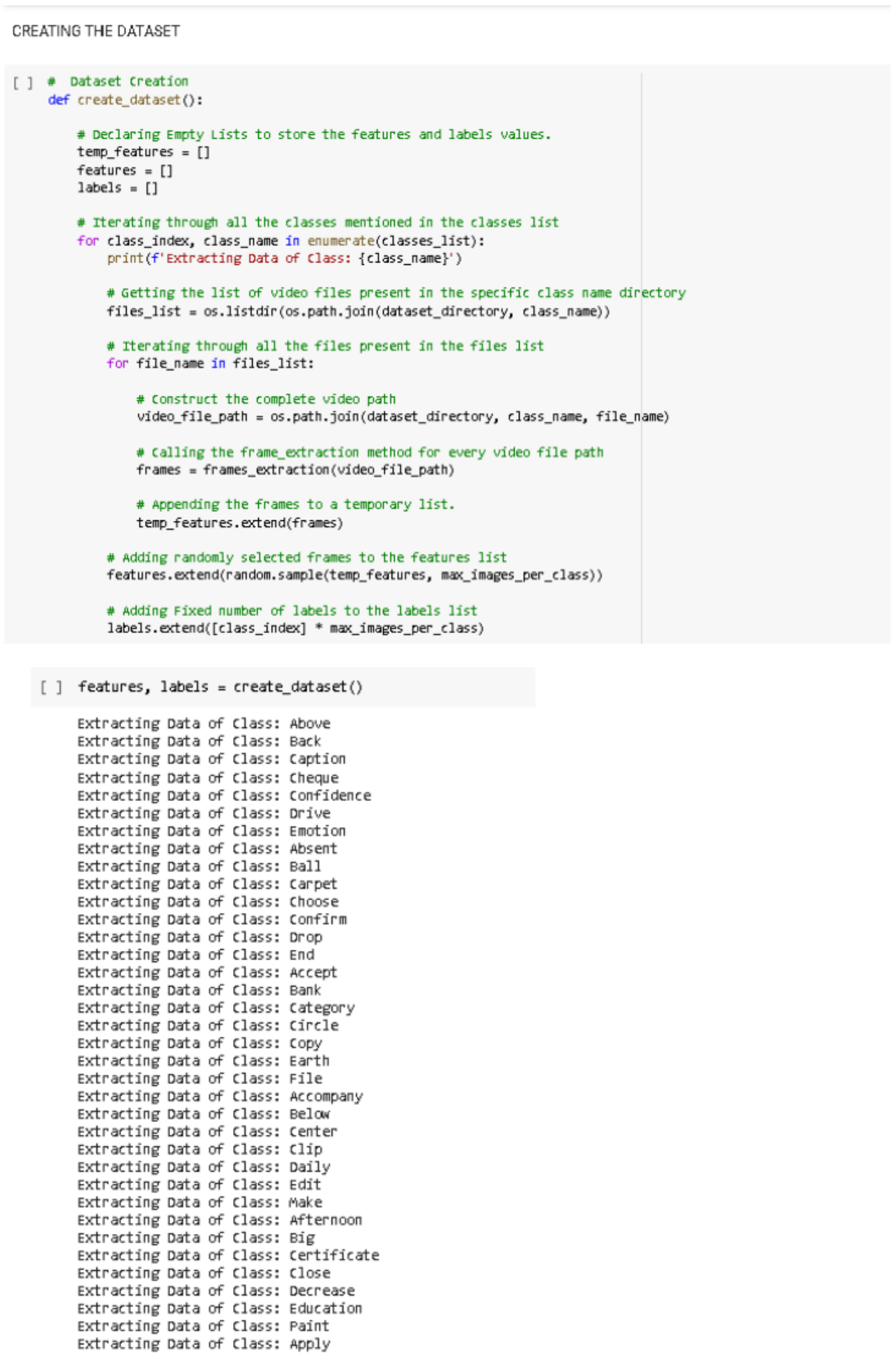
7. Resizing and normalizing the dataset



RESIZING AND NORMALIZING THE FRAMES

```
# Extract, Resize and Normalize Frames
def frames_extraction(video_path):
    # Empty List declared to store video frames
    frames_list = []

    # Reading the Video File Using the VideoCapture
    video_reader = cv2.VideoCapture(video_path)

    # Iterating through Video Frames
    while True:

        # Reading a frame from the video file
        success, frame = video_reader.read()

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

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

        # Normalize the resized frame by dividing it with 255 so that each pixel value then lies between 0 and 1
        normalized_frame = resized_frame / 255

        # Appending the normalized frame into the frames list
        frames_list.append(normalized_frame)

    # Closing the VideoCapture object and releasing all resources.
    video_reader.release()

    # returning the frames list
    return frames_list
```

8. Creating dataset



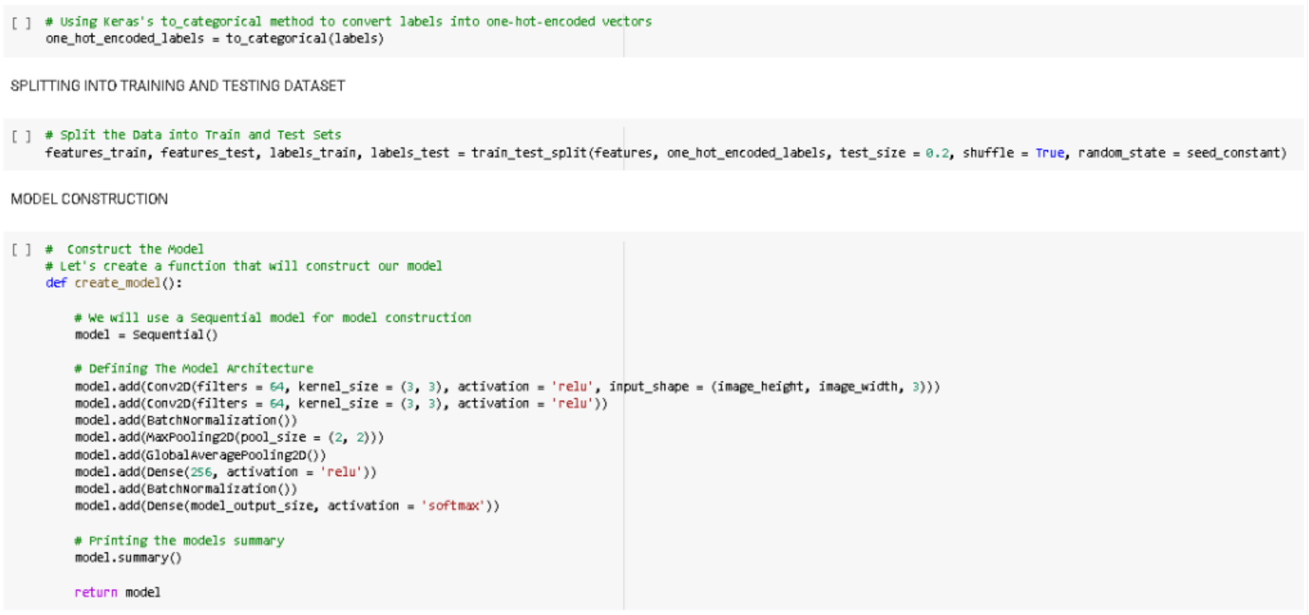
CREATING THE DATASET

```
[ ] #  Dataset Creation
    def create_dataset():

        # Declaring Empty Lists to store the features and labels values.
        temp_features = []
        features = []
        labels = []

        # Iterating through all the classes mentioned in the classes list
        for class_index, class_name in enumerate(classes_list):
            print(f'Extracting Data of Class: {class_name}')

            # Getting the list of video files present in the specific class name directory
            files_list = os.listdir(os.path.join(dataset_directory, class_name))

            # Iterating through all the files present in the files list
            for file_name in files_list:

                # Construct the complete video path
                video_file_path = os.path.join(dataset_directory, class_name, file_name)

                # Calling the frame_extraction method for every video file path
                frames = frames_extraction(video_file_path)

                # Appending the frames to a temporary list.
                temp_features.extend(frames)

            # Adding randomly selected frames to the features list
            features.extend(random.sample(temp_features, max_images_per_class))

            # Adding Fixed number of labels to the labels list
            labels.extend([class_index] * max_images_per_class)
```

```
[ ] features, labels = create_dataset()
```

```
Extracting Data of Class: Above
Extracting Data of Class: Back
Extracting Data of Class: Caption
Extracting Data of Class: Cheque
Extracting Data of Class: Confidence
Extracting Data of Class: Drive
Extracting Data of Class: Emotion
Extracting Data of Class: Absent
Extracting Data of Class: Ball
Extracting Data of Class: Carpet
Extracting Data of Class: Choose
Extracting Data of Class: Confirm
Extracting Data of Class: Drop
Extracting Data of Class: End
Extracting Data of Class: Accept
Extracting Data of Class: Bank
Extracting Data of Class: Category
Extracting Data of Class: Circle
Extracting Data of Class: Copy
Extracting Data of Class: Earth
Extracting Data of Class: File
Extracting Data of Class: Accompany
Extracting Data of Class: Below
Extracting Data of Class: Center
Extracting Data of Class: Clip
Extracting Data of Class: Daily
Extracting Data of Class: Edit
Extracting Data of Class: Make
Extracting Data of Class: Afternoon
Extracting Data of Class: Big
Extracting Data of Class: Certificate
Extracting Data of Class: Close
Extracting Data of Class: Decrease
Extracting Data of Class: Education
Extracting Data of Class: Paint
Extracting Data of Class: Apply
```

9. Splitting into training and test sets



```
[ ] # Using Keras's to_categorical method to convert labels into one-hot-encoded vectors
    one_hot_encoded_labels = to_categorical(labels)
```

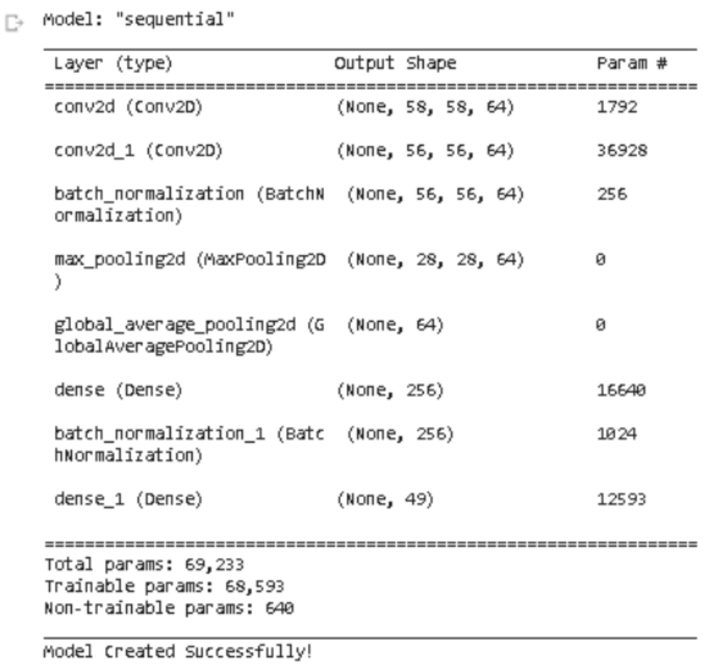
SPLITTING INTO TRAINING AND TESTING DATASET

```
[ ] # Split the Data into Train and Test Sets
    features_train, features_test, labels_train, labels_test = train_test_split(features, one_hot_encoded_labels, test_size = 0.2, shuffle = True, random_state = seed_constant)
```

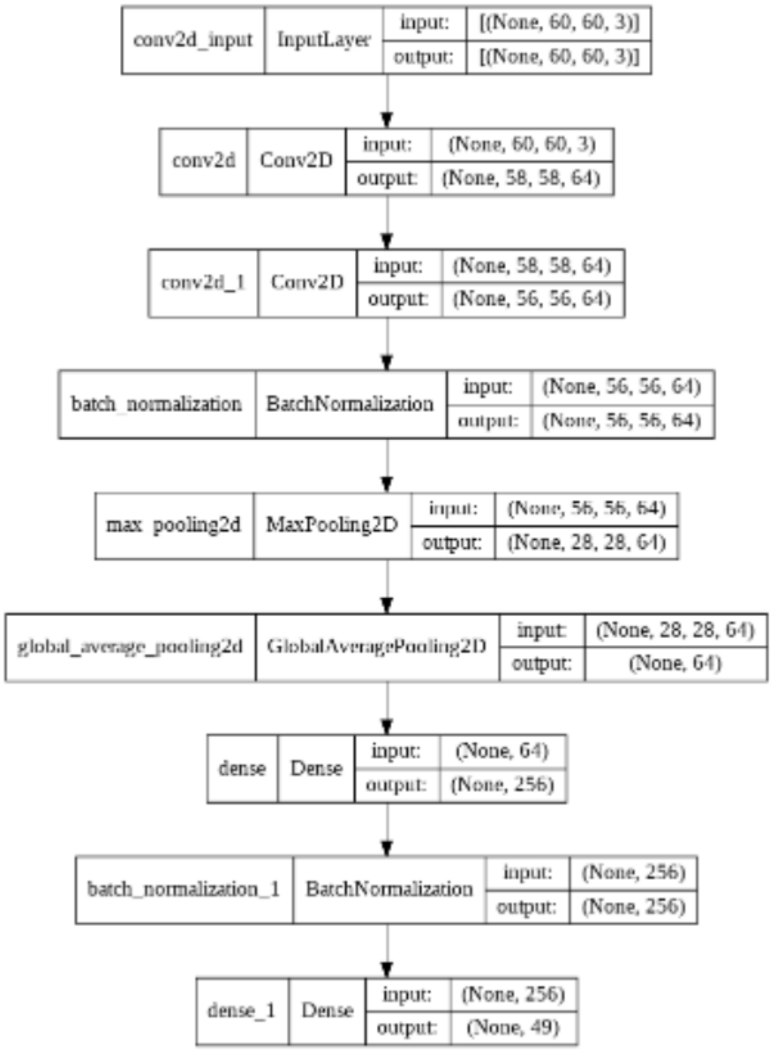
MODEL CONSTRUCTION

```
[ ] #  Construct the Model
    # Let's create a function that will construct our model
    def create_model():

        # We will use 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
```

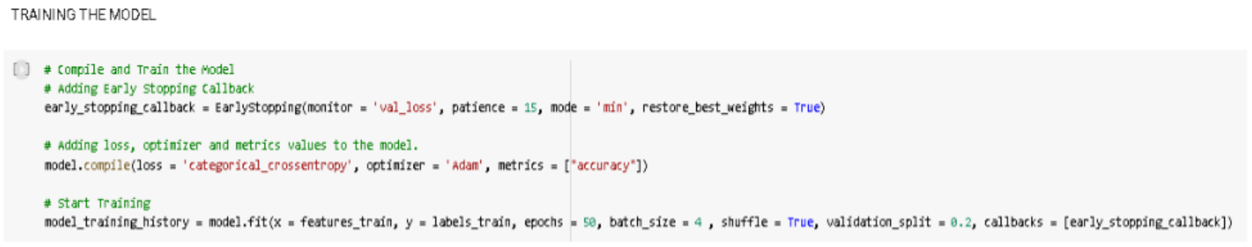
10. Creating   model



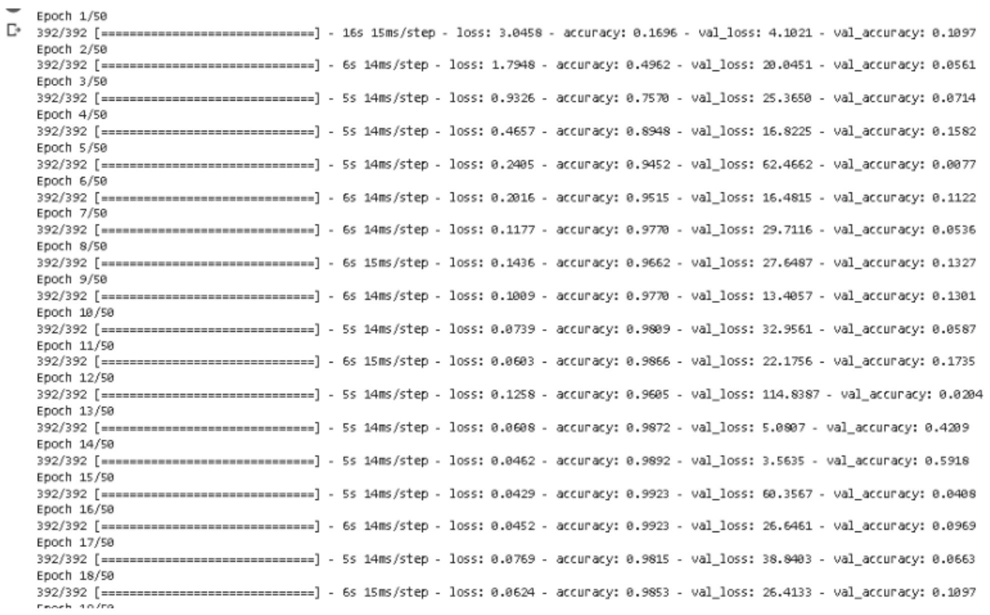
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 58, 58, 64)        1792

 conv2d_1 (Conv2D)           (None, 56, 56, 64)        36928

 batch_normalization (BatchN  (None, 56, 56, 64)       256
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 28, 28, 64)       0
 )

 global_average_pooling2d (G  (None, 64)               0
 lobalAveragePooling2D)

 dense (Dense)               (None, 256)               16640

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 dense_1 (Dense)             (None, 49)                12593

=================================================================
Total params: 69,233
Trainable params: 68,593
Non-trainable params: 640
_________________________________________________________________
Model Created Successfully!
```


conv2d_input | InputLayer | input: [(None, 60, 60, 3)] | output: [(None, 60, 60, 3)]
conv2d | Conv2D | input: (None, 60, 60, 3) | output: (None, 58, 58, 64)
conv2d_1 | Conv2D | input: (None, 58, 58, 64) | output: (None, 56, 56, 64)
batch_normalization | BatchNormalization | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
max_pooling2d | MaxPooling2D | input: (None, 56, 56, 64) | output: (None, 28, 28, 64)
global_average_pooling2d | GlobalAveragePooling2D | input: (None, 28, 28, 64) | output: (None, 64)
dense | Dense | input: (None, 64) | output: (None, 256)
batch_normalization_1 | BatchNormalization | input: (None, 256) | output: (None, 256)
dense_1 | Dense | input: (None, 256) | output: (None, 49)

11. Training model


TRAINING THE MODEL

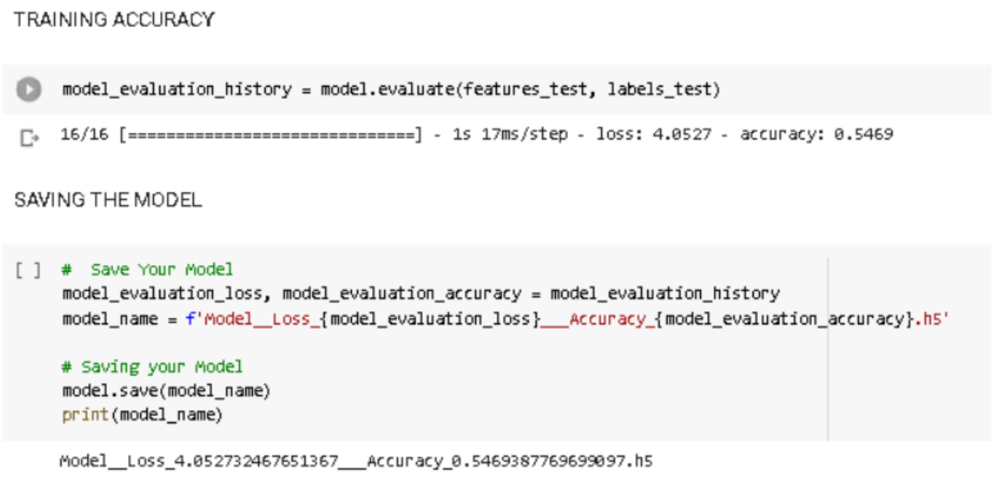
```
# Compile and Train the Model
# Adding Early Stopping Callback
early_stopping_callback = EarlyStopping(monitor = 'val_loss', patience = 15, mode = 'min', restore_best_weights = True)

# Adding loss, optimizer and metrics values to the model.
model.compile(loss = 'categorical_crossentropy', optimizer = 'Adam', metrics = ["accuracy"])

# Start Training
model_training_history = model.fit(x = features_train, y = labels_train, epochs = 50, batch_size = 4 , shuffle = True, validation_split = 0.2, callbacks = [early_stopping_callback])
```



```
Epoch 1/50
392/392 [==============================] - 16s 15ms/step - loss: 3.0458 - accuracy: 0.1696 - val_loss: 4.1021 - val_accuracy: 0.1097
Epoch 2/50
392/392 [==============================] - 6s 14ms/step - loss: 1.7948 - accuracy: 0.4962 - val_loss: 20.0451 - val_accuracy: 0.0561
Epoch 3/50
392/392 [==============================] - 5s 14ms/step - loss: 0.9326 - accuracy: 0.7570 - val_loss: 25.3650 - val_accuracy: 0.0714
Epoch 4/50
392/392 [==============================] - 5s 14ms/step - loss: 0.4657 - accuracy: 0.8948 - val_loss: 16.8225 - val_accuracy: 0.1582
Epoch 5/50
392/392 [==============================] - 5s 14ms/step - loss: 0.2405 - accuracy: 0.9452 - val_loss: 62.4662 - val_accuracy: 0.0077
Epoch 6/50
392/392 [==============================] - 6s 14ms/step - loss: 0.2016 - accuracy: 0.9515 - val_loss: 16.4815 - val_accuracy: 0.1122
Epoch 7/50
392/392 [==============================] - 6s 14ms/step - loss: 0.1177 - accuracy: 0.9770 - val_loss: 29.7116 - val_accuracy: 0.0536
Epoch 8/50
392/392 [==============================] - 6s 15ms/step - loss: 0.1436 - accuracy: 0.9662 - val_loss: 27.6487 - val_accuracy: 0.1327
Epoch 9/50
392/392 [==============================] - 6s 14ms/step - loss: 0.1009 - accuracy: 0.9770 - val_loss: 13.4057 - val_accuracy: 0.1301
Epoch 10/50
392/392 [==============================] - 5s 14ms/step - loss: 0.0739 - accuracy: 0.9809 - val_loss: 32.9561 - val_accuracy: 0.0587
Epoch 11/50
392/392 [==============================] - 6s 15ms/step - loss: 0.0603 - accuracy: 0.9866 - val_loss: 22.1756 - val_accuracy: 0.1735
Epoch 12/50
392/392 [==============================] - 5s 14ms/step - loss: 0.1258 - accuracy: 0.9605 - val_loss: 114.8387 - val_accuracy: 0.0204
Epoch 13/50
392/392 [==============================] - 5s 14ms/step - loss: 0.0608 - accuracy: 0.9872 - val_loss: 5.0807 - val_accuracy: 0.4209
Epoch 14/50
392/392 [==============================] - 5s 14ms/step - loss: 0.0462 - accuracy: 0.9892 - val_loss: 3.5635 - val_accuracy: 0.5918
Epoch 15/50
392/392 [==============================] - 5s 14ms/step - loss: 0.0429 - accuracy: 0.9923 - val_loss: 60.3567 - val_accuracy: 0.0408
Epoch 16/50
392/392 [==============================] - 6s 14ms/step - loss: 0.0452 - accuracy: 0.9923 - val_loss: 26.6461 - val_accuracy: 0.0969
Epoch 17/50
392/392 [==============================] - 5s 14ms/step - loss: 0.0769 - accuracy: 0.9815 - val_loss: 38.8403 - val_accuracy: 0.0663
Epoch 18/50
392/392 [==============================] - 6s 15ms/step - loss: 0.0624 - accuracy: 0.9853 - val_loss: 26.4133 - val_accuracy: 0.1097
```

12. Getting model accuracy

13. Saving the model



TRAINING ACCURACY

```
model_evaluation_history = model.evaluate(features_test, labels_test)
```

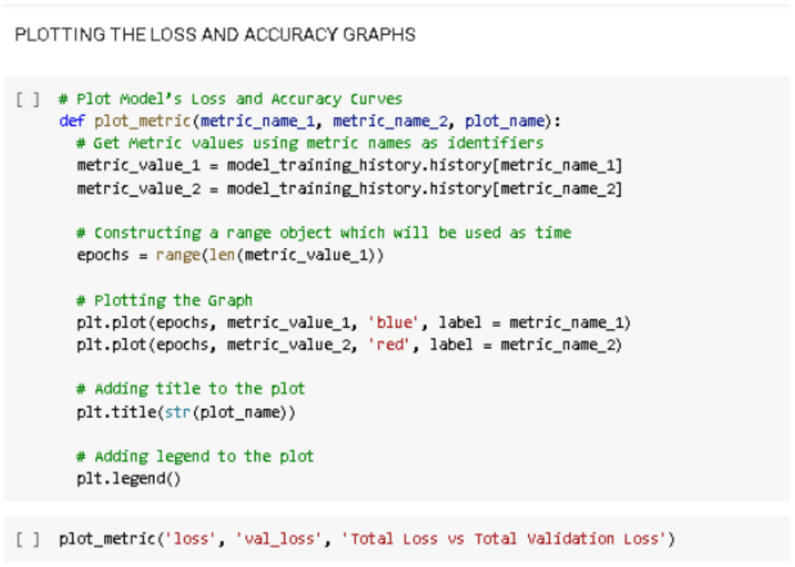
```
16/16 [==============================] - 1s 17ms/step - loss: 4.0527 - accuracy: 0.5469
```

SAVING THE MODEL

```
#  Save Your Model
model_evaluation_loss, model_evaluation_accuracy = model_evaluation_history
model_name = f'Model__Loss_{model_evaluation_loss}__Accuracy_{model_evaluation_accuracy}.h5'

# Saving your Model
model.save(model_name)
print(model_name)
```

```
Model__Loss_4.052732467651367__Accuracy_0.5469387769699097.h5
```

14. Plotting the loss and accuracy curves

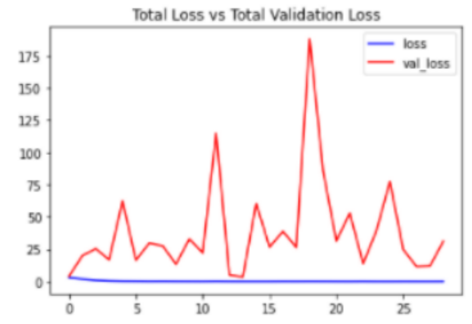


PLOTTING THE LOSS AND ACCURACY GRAPHS

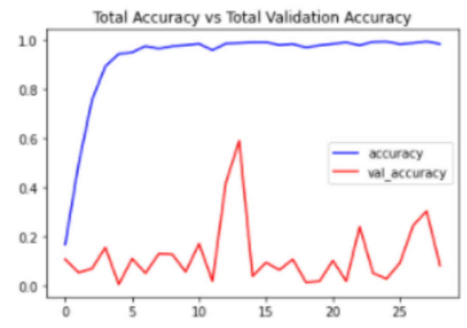
```
[ ] # Plot Model's Loss and Accuracy Curves
    def plot_metric(metric_name_1, metric_name_2, plot_name):
      # Get Metric values using metric names as identifiers
      metric_value_1 = model_training_history.history[metric_name_1]
      metric_value_2 = model_training_history.history[metric_name_2]

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

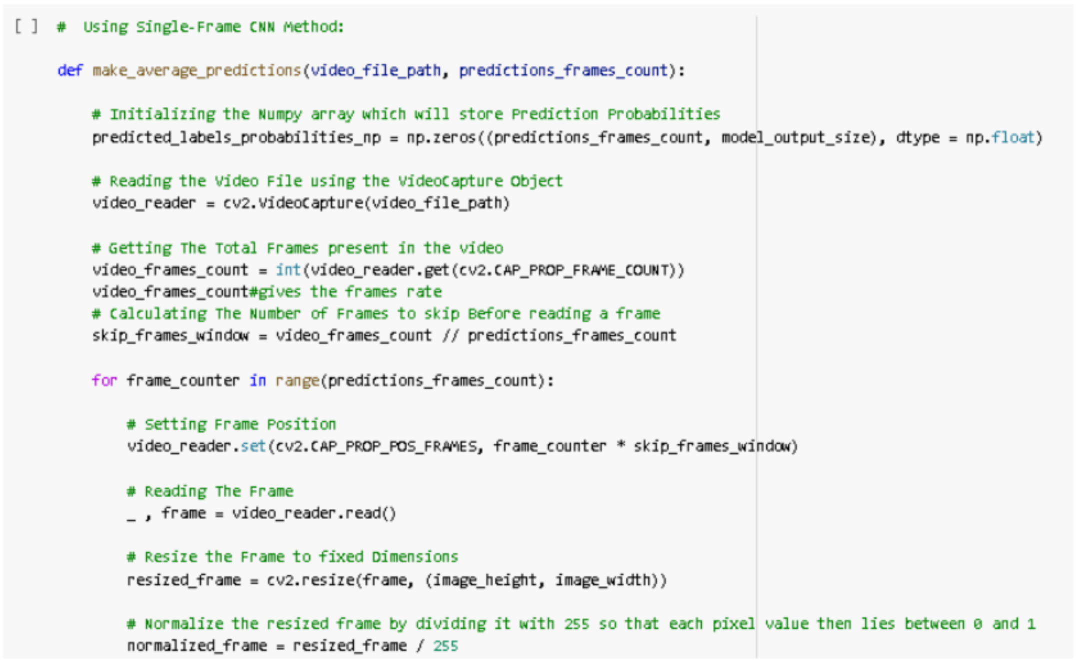


```
[ ] plot_metric('accuracy', 'val_accuracy', 'Total Accuracy vs Total Validation Accuracy')
```

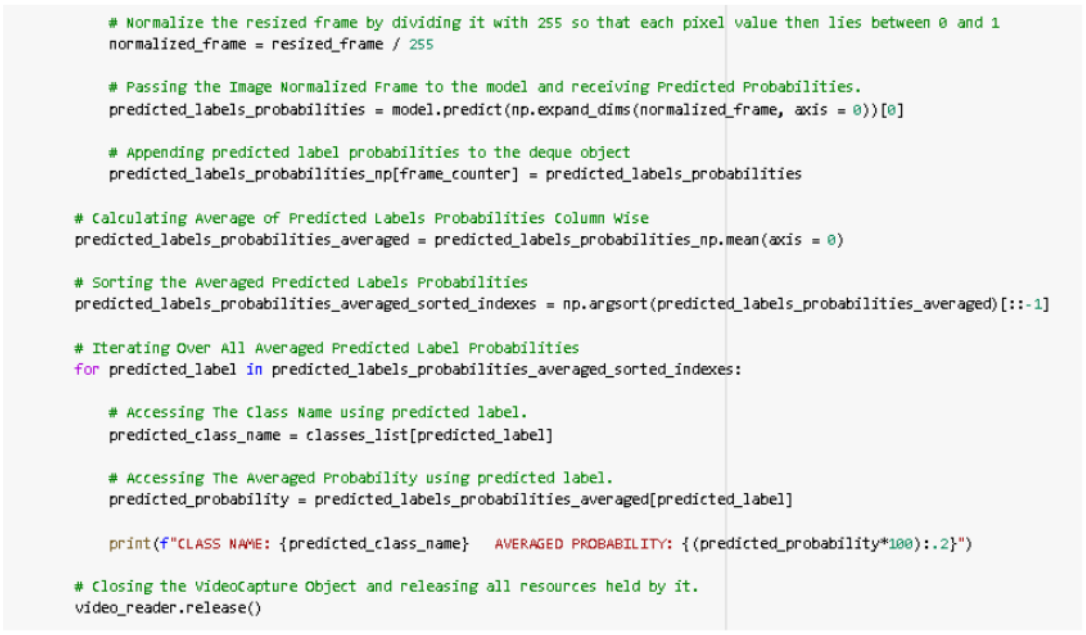


15. Making predictions

USING SINGLE-FRAME CNN METHORD



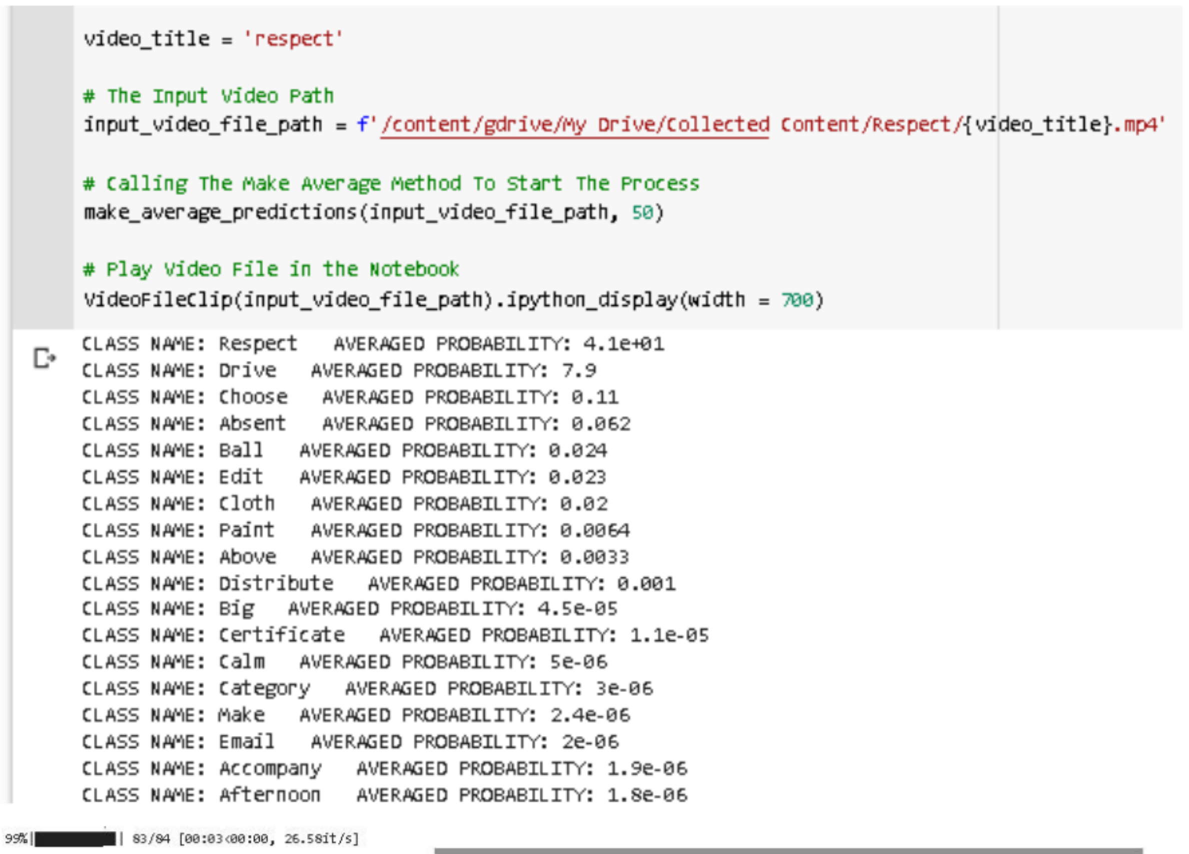
```
[ ] #  Using Single-Frame CNN Method:

    def make_average_predictions(video_file_path, predictions_frames_count):

        # Initializing the Numpy array which will store Prediction Probabilities
        predicted_labels_probabilities_np = np.zeros((predictions_frames_count, model_output_size), dtype = np.float)

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

        # Getting The Total Frames present in the video
        video_frames_count = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))
        video_frames_count#gives the frames rate
        # Calculating The Number of Frames to skip Before reading a frame
        skip_frames_window = video_frames_count // predictions_frames_count

        for frame_counter in range(predictions_frames_count):

            # Setting Frame Position
            video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_counter * skip_frames_window)

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

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

            # Normalize the resized frame by dividing it with 255 so that each pixel value then lies between 0 and 1
            normalized_frame = resized_frame / 255
```



```
            # Normalize the resized frame by dividing it with 255 so that each pixel value then lies between 0 and 1
            normalized_frame = resized_frame / 255
```

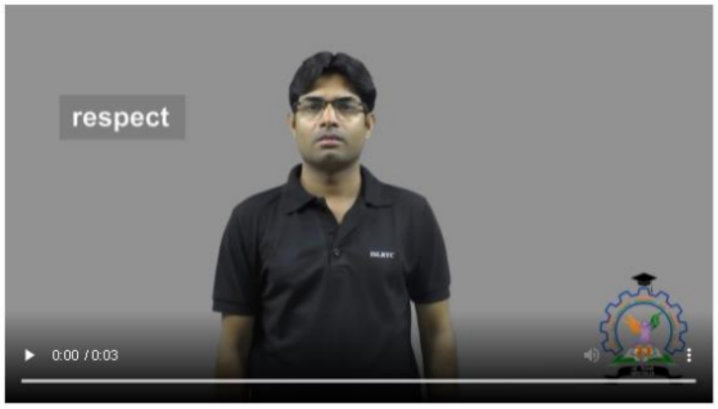
```
            # Passing the Image Normalized Frame to the model and receiving Predicted Probabilities.
            predicted_labels_probabilities = model.predict(np.expand_dims(normalized_frame, axis = 0))[0]

            # Appending predicted label probabilities to the deque object
            predicted_labels_probabilities_np[frame_counter] = predicted_labels_probabilities

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

        # Sorting the Averaged Predicted Labels Probabilities
        predicted_labels_probabilities_averaged_sorted_indexes = np.argsort(predicted_labels_probabilities_averaged)[::-1]

        # Iterating Over All Averaged Predicted Label Probabilities
        for predicted_label in predicted_labels_probabilities_averaged_sorted_indexes:

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

            # Accessing The Averaged Probability using predicted label.
            predicted_probability = predicted_labels_probabilities_averaged[predicted_label]

            print(f"CLASS NAME: {predicted_class_name}   AVERAGED PROBABILITY: {(predicted_probability*100):.2}")

        # Closing the VideoCapture Object and releasing all resources held by it.
        video_reader.release()
```

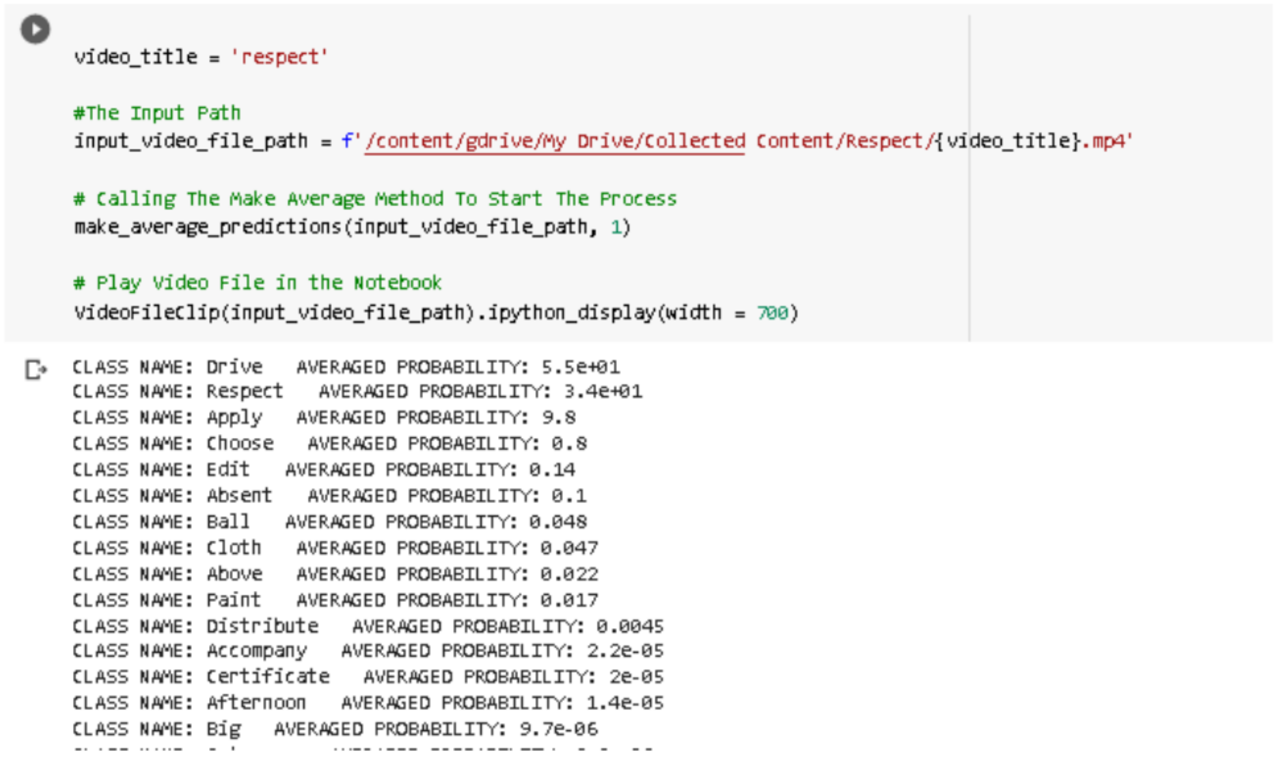
16. Results



```
video_title = 'respect'

# The Input Video Path
input_video_file_path = f'/content/gdrive/My Drive/Collected Content/Respect/{video_title}.mp4'

# Calling The Make Average Method To Start The Process
make_average_predictions(input_video_file_path, 50)

# Play Video File in the Notebook
VideoFileClip(input_video_file_path).ipython_display(width = 700)
```

```
CLASS NAME: Respect   AVERAGED PROBABILITY: 4.1e+01
CLASS NAME: Drive   AVERAGED PROBABILITY: 7.9
CLASS NAME: Choose   AVERAGED PROBABILITY: 0.11
CLASS NAME: Absent   AVERAGED PROBABILITY: 0.062
CLASS NAME: Ball   AVERAGED PROBABILITY: 0.024
CLASS NAME: Edit   AVERAGED PROBABILITY: 0.023
CLASS NAME: Cloth   AVERAGED PROBABILITY: 0.02
CLASS NAME: Paint   AVERAGED PROBABILITY: 0.0064
CLASS NAME: Above   AVERAGED PROBABILITY: 0.0033
CLASS NAME: Distribute   AVERAGED PROBABILITY: 0.001
CLASS NAME: Big   AVERAGED PROBABILITY: 4.5e-05
CLASS NAME: Certificate   AVERAGED PROBABILITY: 1.1e-05
CLASS NAME: Calm   AVERAGED PROBABILITY: 5e-06
CLASS NAME: Category   AVERAGED PROBABILITY: 3e-06
CLASS NAME: Make   AVERAGED PROBABILITY: 2.4e-06
CLASS NAME: Email   AVERAGED PROBABILITY: 2e-06
CLASS NAME: Accompany   AVERAGED PROBABILITY: 1.9e-06
CLASS NAME: Afternoon   AVERAGED PROBABILITY: 1.8e-06

99%|█████████| 83/84 [00:03<00:00, 26.58it/s]
```





```
video_title = 'respect'

#The Input Path
input_video_file_path = f'/content/gdrive/My Drive/Collected Content/Respect/{video_title}.mp4'

# Calling The Make Average Method To Start The Process
make_average_predictions(input_video_file_path, 1)

# Play Video File in the Notebook
VideoFileClip(input_video_file_path).ipython_display(width = 700)
```

```
CLASS NAME: Drive   AVERAGED PROBABILITY: 5.5e+01
CLASS NAME: Respect   AVERAGED PROBABILITY: 3.4e+01
CLASS NAME: Apply   AVERAGED PROBABILITY: 9.8
CLASS NAME: Choose   AVERAGED PROBABILITY: 0.8
CLASS NAME: Edit   AVERAGED PROBABILITY: 0.14
CLASS NAME: Absent   AVERAGED PROBABILITY: 0.1
CLASS NAME: Ball   AVERAGED PROBABILITY: 0.048
CLASS NAME: Cloth   AVERAGED PROBABILITY: 0.047
CLASS NAME: Above   AVERAGED PROBABILITY: 0.022
CLASS NAME: Paint   AVERAGED PROBABILITY: 0.017
CLASS NAME: Distribute   AVERAGED PROBABILITY: 0.0045
CLASS NAME: Accompany   AVERAGED PROBABILITY: 2.2e-05
CLASS NAME: Certificate   AVERAGED PROBABILITY: 2e-05
CLASS NAME: Afternoon   AVERAGED PROBABILITY: 1.4e-05
CLASS NAME: Big   AVERAGED PROBABILITY: 9.7e-06
```

10. CONCLUSION

The model we made achieved an accuracy of 55.7% accuracy we need to apply different model of deep learning to get better accuracy and better predictions.

We need to increase the size of dataset also as each video is of 2-3-4 seconds only.

report

ORIGINALITY REPORT

0%	0%	0%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		