```
# Discard the output of this cell.

# Install the required libraries.
!pip install youtube-dl moviepy==1.0.3
!pip install git+https://github.com/TahaAnwar/pafy.git#egg=pafy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting youtube-dl
  Downloading youtube_dl-2021.12.17-py2.py3-none-any.whl (1.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 14.7 MB/s eta 0:00:00
Collecting moviepy==1.0.3
  Downloading moviepy-1.0.3.tar.gz (388 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 388.3/388.3 KB 13.8 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: decorator<5.0,>=4.0.2 in /usr/local/lib/python3.9/dist-packages (from moviepy==1.0.3) (4.4.2)
Requirement already satisfied: tqdm<5.0,>=4.11.2 in /usr/local/lib/python3.9/dist-packages (from moviepy==1.0.3) (4.65.0)
Requirement already satisfied: requests<3.0,>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from moviepy==1.0.3) (2.27.1)
Collecting proglog<=1.0.0
  Downloading proglog-0.1.10-py3-none-any.whl (6.1 kB)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.9/dist-packages (from moviepy==1.0.3) (1.24.2)
Requirement already satisfied: imageio<3.0,>=2.5 in /usr/local/lib/python3.9/dist-packages (from moviepy==1.0.3) (2.25.1)
Collecting imageio_ffmpeg>=0.2.0
  Downloading imageio_ffmpeg-0.4.8-py3-none-manylinux2010_x86_64.whl (26.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 26.9/26.9 MB 23.4 MB/s eta 0:00:00
Requirement already satisfied: pillow>=8.3.2 in /usr/local/lib/python3.9/dist-packages (from imageio<3.0,>=2.5->moviepy==1.0.3) (8.4.0)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests<3.0,>=2.8.1->moviepy==1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests<3.0,>=2.8.1->moviepy==1.0.3) (
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests<3.0,>=2.8.1->moviepy==1.0.3) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests<3.0,>=2.8.1->moviepy==1.0.3
Building wheels for collected packages: moviepy
  Building wheel for moviepy (setup.py) ... done
  Created wheel for moviepy: filename=moviepy-1.0.3-py3-none-any.whl size=110743 sha256=9c5965bfc59bd55188008ba7e05f0e237b38f23448b5f9cd9
  Stored in directory: /root/.cache/pip/wheels/29/15/e4/4f790bec6acd51a00b67e8ee1394f0bc6e0135c315f8ff399a
Successfully built moviepy
Installing collected packages: youtube-dl, proglog, imageio_ffmpeg, moviepy
  Attempting uninstall: moviepy
    Found existing installation: moviepy 0.2.3.5
    Uninstalling moviepy-0.2.3.5:
      Successfully uninstalled moviepy-0.2.3.5
Successfully installed imageio_ffmpeg-0.4.8 moviepy-1.0.3 proglog-0.1.10 youtube-dl-2021.12.17
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pafy
  Cloning https://github.com/TahaAnwar/pafy.git to /tmp/pip-install-_73y3izw/pafy_107356f2e7914e43af778d8e8963b47e
  Running command git clone --filter=blob:none --quiet https://github.com/TahaAnwar/pafy.git /tmp/pip-install-_73y3izw/pafy_107356f2e7914
  Resolved https://github.com/TahaAnwar/pafy.git to commit 2f3c473b3df7961721d07e1504675313afd1d2cb
  Preparing metadata (setup.py) ... done
```

```
Building wheels for collected packages: pafy
  Building wheel for pafy (setup.py) ... done
  Created wheel for pafy: filename=pafy-0.5.5-py2.py3-none-any.whl size=35706 sha256=a14c7536e6dd158dea09cbe5862e96849bc4f061fffc34e307b4
  Stored in directory: /tmp/pip-ephem-wheel-cache-mh7hq7rc/wheels/fe/93/42/b1f9b93e4ae72fc640e33a4586f943a0f02aa6e5e2f1891b71
Successfully built pafy
Installing collected packages: pafy
Successfully installed pafy-0.5.5
```

```
# Import the required libraries.
import os
import cv2
import pafy
import math
import random
import numpy as np
import datetime as dt
import tensorflow as tf
from collections import deque
import matplotlib.pyplot as plt

from moviepy.editor import *
%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
```

And will set `Numpy`, `Python`, and `Tensorflow` seeds to get consistent results on every execution.

```
seed_constant = 27
np.random.seed(seed_constant)
random.seed(seed_constant)
tf.random.set_seed(seed_constant)
```

Step 1: Download and Visualize the Data with its Labels

In the first step, we will download and visualize the data along with labels to get an idea about what we will be dealing with. We will be using the UCF50 - Action Recognition Dataset, consisting of realistic videos taken from youtube which differentiates this data set from most of the other available action recognition data sets as they are not realistic and are staged by actors. The Dataset contains:

- `50` Action Categories
- `25` Groups of Videos per Action Category
- `133` Average Videos per Action Category
- `199` Average Number of Frames per Video
- `320` Average Frames Width per Video
- `240` Average Frames Height per Video
- `26` Average Frames Per Seconds per Video

Let's download and extract the dataset.

```
# Discard the output of this cell.

# Downlaod the UCF50 Dataset
!wget --no-check-certificate https://www.crcv.ucf.edu/data/UCF50.rar

#Extract the Dataset
!unrar x UCF50.rar
```

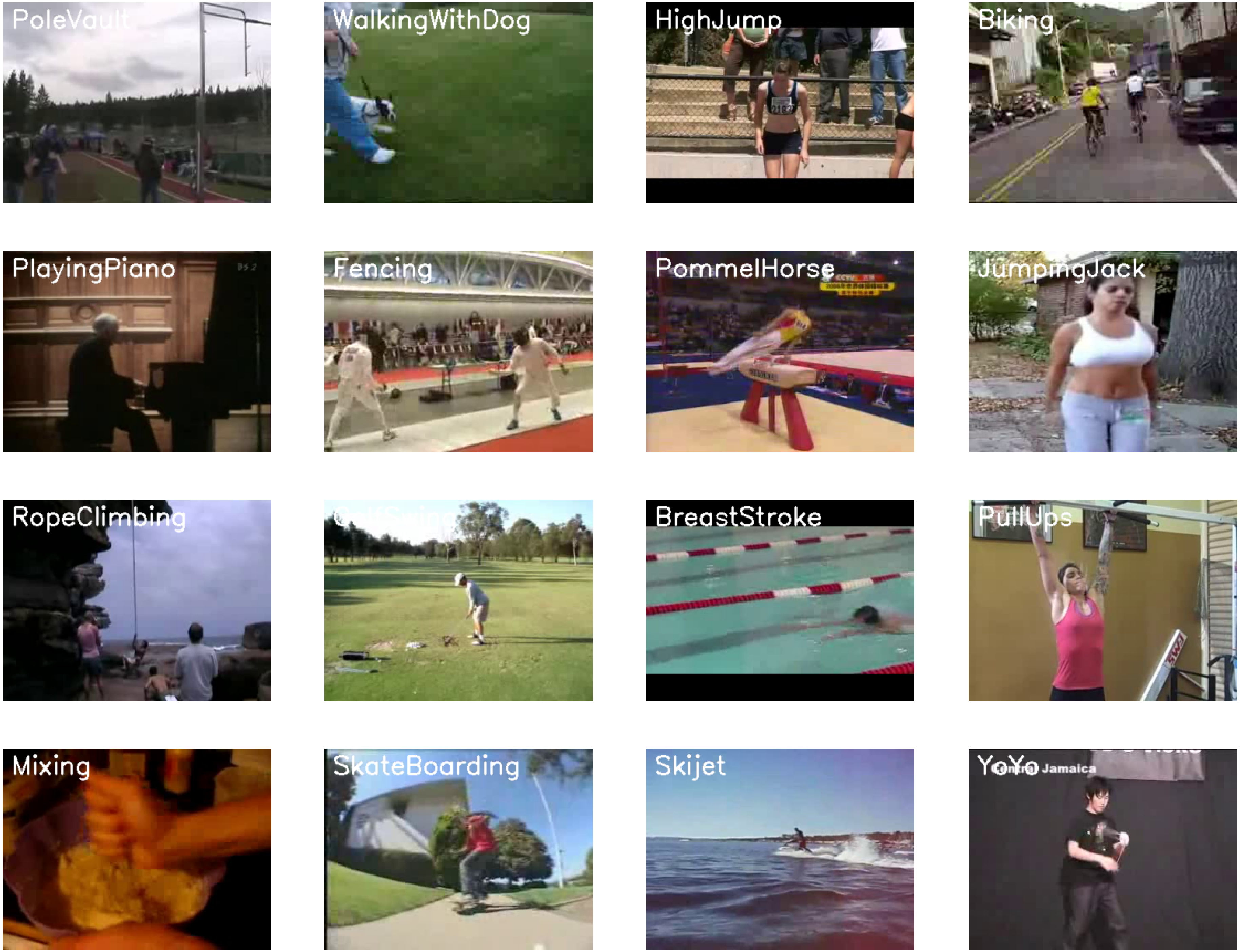
Streaming output truncated to the last 5000 lines.

```
Extracting  UCF50/HorseRace/v_HorseRace_g16_c03.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g16_c04.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g16_c05.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g17_c01.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g17_c02.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g17_c03.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g17_c04.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g17_c05.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g18_c01.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g18_c02.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g18_c03.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g18_c04.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g18_c05.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g18_c06.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g19_c01.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g19_c02.avi                   OK
Extracting  UCF50/HorseRace/v_HorseRace_g19_c03.avi                   OK
```

```
Extracting  UCF50/HorseRace/v_HorseRace_g19_c04.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g19_c05.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g19_c06.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g20_c01.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g20_c02.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g20_c03.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g20_c04.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g20_c05.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g21_c01.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g21_c02.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g21_c03.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g21_c04.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g21_c05.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g22_c01.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g22_c02.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g22_c03.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g22_c04.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g23_c01.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g23_c02.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g23_c03.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g23_c04.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g23_c05.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g24_c01.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g24_c02.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g24_c03.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g24_c04.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g24_c05.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g24_c06.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g24_c07.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g24_c08.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g24_c09.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g24_c10.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g25_c01.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g25_c02.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g25_c03.avi              OK
Extracting  UCF50/HorseRace/v_HorseRace_g25_c04.avi              OK
Creating    UCF50/HorseRiding                                    OK
Extracting  UCF50/HorseRiding/v_HorseRiding_g01_c01.avi          OK
Extracting  UCF50/HorseRiding/v_HorseRiding_g01_c02.avi          OK
Extracting  UCF50/HorseRiding/v_HorseRiding_g01_c03.avi          OK
```

For visualization, we will pick `20` random categories from the dataset and a random video from each selected category and will visualize the first frame of the selected videos with their associated labels written. This way we'll be able to visualize a subset (`20` random videos) of the dataset.

```
# Create a Matplotlib figure and specify the size of the figure.
plt.figure(figsize = (20, 20))

# Get the names of all classes/categories in UCF50.
all_classes_names = os.listdir('UCF50')

# Generate a list of 20 random values. The values will be between 0-50,
# where 50 is the total number of class in the dataset.
random_range = random.sample(range(len(all_classes_names)), 20)

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

    # Retrieve a Class Name using the Random Index.
    selected_class_Name = all_classes_names[random_index]

    # Retrieve the list of all the video files present in the randomly selected Class Directory.
    video_files_names_list = os.listdir(f'UCF50/{selected_class_Name}')

    # Randomly select a video file from the list retrieved from the randomly selected Class Directory.
    selected_video_file_name = random.choice(video_files_names_list)

    # Initialize a VideoCapture object to read from the video File.
    video_reader = cv2.VideoCapture(f'UCF50/{selected_class_Name}/{selected_video_file_name}')

    # Read the first frame of the video file.
    _, bgr_frame = video_reader.read()

    # Release the VideoCapture object.
    video_reader.release()

    # Convert the frame from BGR into RGB format.
    rgb_frame = cv2.cvtColor(bgr_frame, cv2.COLOR_BGR2RGB)

    # Write the class name on the video frame.
    cv2.putText(rgb_frame, selected_class_Name, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)

    # Display the frame.
    plt.subplot(5, 4, counter);plt.imshow(rgb_frame);plt.axis('off')
```


PoleVault
WalkingWithDog
HighJump
Biking
PlayingPiano
Fencing
PommelHorse
JumpingJack
RopeClimbing
BreastStroke
PullUps
Mixing
SkateBoarding
Skijet
YoYo

Step 2: Preprocess the Dataset

Next, we will perform some preprocessing on the dataset. First, we will read the video files from the dataset and resize the frames of the videos to a fixed width and height, to reduce the computations and normalized the data to range `[0-1]` by dividing the pixel values with `255`, which makes convergence faster while training the network.

But first, let's initialize some constants.

```
# Specify the height and width to which each video frame will be resized in our dataset.
IMAGE_HEIGHT , IMAGE_WIDTH = 64, 64

# Specify the number of frames of a video that will be fed to the model as one sequence.
SEQUENCE_LENGTH = 20

# Specify the directory containing the UCF50 dataset.
DATASET_DIR = "UCF50"

# Specify the list containing the names of the classes used for training. Feel free to choose any set of classes.
CLASSES_LIST = ["WalkingWithDog", "TaiChi", "Swing", "HorseRace"]
```

Note: *The **`IMAGE_HEIGHT`**, **`IMAGE_WIDTH`** and **`SEQUENCE_LENGTH`** constants can be increased for better results, although increasing the sequence length is only effective to a certain point, and increasing the values will result in the process being more computationally expensive.*

Create a Function to Extract, Resize & Normalize Frames

We will create a function **`frames_extraction()`** that will create a list containing the resized and normalized frames of a video whose path is passed to it as an argument. The function will read the video file frame by frame, although not all frames are added to the list as we will only need an evenly distributed sequence length of frames.

```
def frames_extraction(video_path):
    '''
    This function will extract the required frames from a video after resizing and normalizing them.
    Args:
        video_path: The path of the video in the disk, whose frames are to be extracted.
    Returns:
        frames_list: A list containing the resized and normalized frames of the video.
    '''
```

```
    # Declare a list to store video frames.
    frames_list = []

    # Read the Video File using the VideoCapture object.
    video_reader = cv2.VideoCapture(video_path)

    # Get the total number of frames in the video.
    video_frames_count = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))

    # Calculate the the interval after which frames will be added to the list.
    skip_frames_window = max(int(video_frames_count/SEQUENCE_LENGTH), 1)

    # Iterate through the Video Frames.
    for frame_counter in range(SEQUENCE_LENGTH):

        # Set the current frame position of the video.
        video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_counter * skip_frames_window)

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

        # Check if Video frame is not successfully read then break the loop
        if not success:
            break

        # Resize the Frame to fixed height and width.
        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

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

    # Release the VideoCapture object.
    video_reader.release()

    # Return the frames list.
    return frames_list
```

Create a Function for Dataset Creation

Now we will create a function `create_dataset()` that will iterate through all the classes specified in the `CLASSES_LIST` constant and will call the function `frame_extraction()` on every video file of the selected classes and return the frames (`features`), class index (`labels`), and video file path (`video_files_paths`).

```
def create_dataset():
    '''
    This function will extract the data of the selected classes and create the required dataset.
    Returns:
        features:          A list containing the extracted frames of the videos.
        labels:            A list containing the indexes of the classes associated with the videos.
        video_files_paths: A list containing the paths of the videos in the disk.
    '''

    # Declared Empty Lists to store the features, labels and video file path values.
    features = []
    labels = []
    video_files_paths = []

    # Iterating through all the classes mentioned in the classes list
    for class_index, class_name in enumerate(CLASSES_LIST):

        # Display the name of the class whose data is being extracted.
        print(f'Extracting Data of Class: {class_name}')

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

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

            # Get the complete video path.
            video_file_path = os.path.join(DATASET_DIR, class_name, file_name)

            # Extract the frames of the video file.
            frames = frames_extraction(video_file_path)

            # Check if the extracted frames are equal to the SEQUENCE_LENGTH specified above.
            # So ignore the vides having frames less than the SEQUENCE_LENGTH.
            if len(frames) == SEQUENCE_LENGTH:

                # Append the data to their repective lists.
                features.append(frames)
                labels.append(class_index)
```

```
                video_files_paths.append(video_file_path)

    # Converting the list to numpy arrays
    features = np.asarray(features)
    labels = np.array(labels)

    # Return the frames, class index, and video file path.
    return features, labels, video_files_paths
```

Now we will utilize the function **`create_dataset()`** created above to extract the data of the selected classes and create the required dataset.

```
# Create the dataset.
features, labels, video_files_paths = create_dataset()
```

```
Extracting Data of Class: WalkingWithDog
Extracting Data of Class: TaiChi
Extracting Data of Class: Swing
Extracting Data of Class: HorseRace
```

Now we will convert `labels` (class indexes) into one-hot encoded vectors.

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

Step 3: Split the Data into Train and Test Set

As of now, we have the required **`features`** (a NumPy array containing all the extracted frames of the videos) and **`one_hot_encoded_labels`** (also a Numpy array containing all class labels in one hot encoded format). So now, we will split our data to create training and testing sets. We will also shuffle the dataset before the split to avoid any bias and get splits representing the overall distribution of the data.

```
# Split the Data into Train ( 75% ) and Test Set ( 25% ).
features_train, features_test, labels_train, labels_test = train_test_split(features, one_hot_encoded_labels,
                                                                            test_size = 0.25, shuffle = True,
                                                                            random_state = seed_constant)
```

CovLSTM Approach

```
def create_convlstm_model():
    '''
    This function will construct the required convlstm model.
    Returns:
        model: It is the required constructed convlstm model.
    '''

    # We will use a Sequential model for model construction
    model = Sequential()

    # Define the Model Architecture.
    ########################################################################################################################

    model.add(ConvLSTM2D(filters = 4, kernel_size = (3, 3), activation = 'tanh',data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True, input_shape = (SEQUENCE_LENGTH,
                                                                                      IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(ConvLSTM2D(filters = 8, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(ConvLSTM2D(filters = 14, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(ConvLSTM2D(filters = 16, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    #model.add(TimeDistributed(Dropout(0.2)))

    model.add(Flatten())

    model.add(Dense(len(CLASSES_LIST), activation = "softmax"))

    ########################################################################################################################
```

```
    # Display the models summary.
    model.summary()

    # Return the constructed convlstm model.
    return model
```

Now we will utilize the function **`create_convlstm_model()`** created above, to construct the required `convlstm` model.

```
# Construct the required convlstm model.
convlstm_model = create_convlstm_model()

# Display the success message.
print("Model Created Successfully!")
```

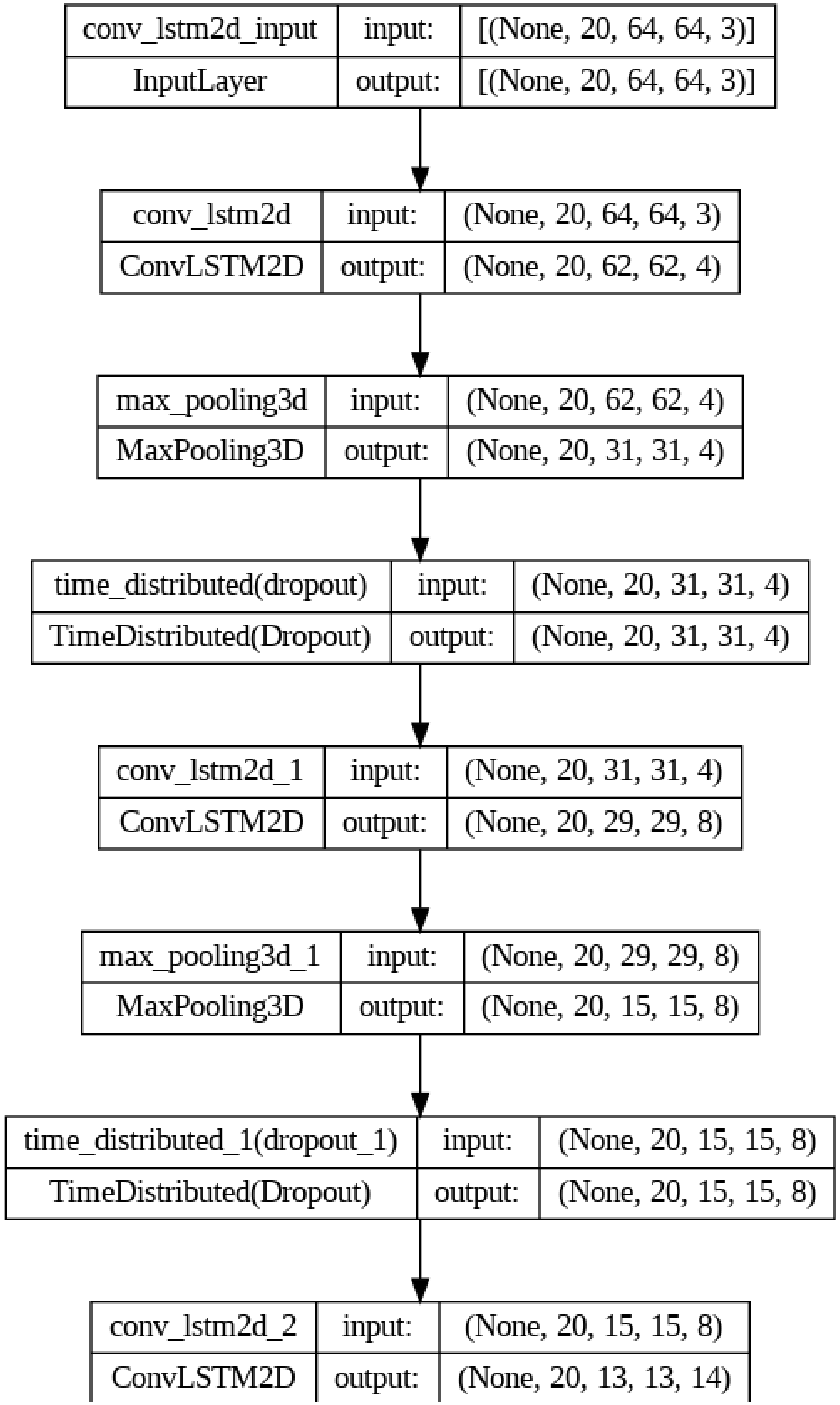
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv_lstm2d (ConvLSTM2D)    (None, 20, 62, 62, 4)     1024      
                                                                 
 max_pooling3d (MaxPooling3D  (None, 20, 31, 31, 4)    0         
 )                                                               
                                                                 
 time_distributed (TimeDistr  (None, 20, 31, 31, 4)    0         
 ibuted)                                                         
                                                                 
 conv_lstm2d_1 (ConvLSTM2D)  (None, 20, 29, 29, 8)     3488      
                                                                 
 max_pooling3d_1 (MaxPooling  (None, 20, 15, 15, 8)    0         
 3D)                                                             
                                                                 
 time_distributed_1 (TimeDis  (None, 20, 15, 15, 8)    0         
 tributed)                                                       
                                                                 
 conv_lstm2d_2 (ConvLSTM2D)  (None, 20, 13, 13, 14)    11144     
                                                                 
 max_pooling3d_2 (MaxPooling  (None, 20, 7, 7, 14)     0         
 3D)                                                             
                                                                 
 time_distributed_2 (TimeDis  (None, 20, 7, 7, 14)     0         
 tributed)                                                       
                                                                 
 conv_lstm2d_3 (ConvLSTM2D)  (None, 20, 5, 5, 16)      17344     
                                                                 
 max_pooling3d_3 (MaxPooling  (None, 20, 3, 3, 16)     0         
```

```
 3D)

 flatten (Flatten)           (None, 2880)              0

 dense (Dense)               (None, 4)                 11524

=================================================================
Total params: 44,524
Trainable params: 44,524
Non-trainable params: 0
_________________________________________________________________
Model Created Successfully!
```

Check Model's Structure:

Now we will use the `plot_model()` function, to check the structure of the constructed model, this is helpful while constructing a complex network and making that the network is created correctly.

```
# Plot the structure of the contructed model.
plot_model(convlstm_model, to_file = 'convlstm_model_structure_plot.png', show_shapes = True, show_layer_names = True)
```


conv_lstm2d_input | input: | [(None, 20, 64, 64, 3)]
InputLayer | output: | [(None, 20, 64, 64, 3)]
conv_lstm2d | input: | (None, 20, 64, 64, 3)
ConvLSTM2D | output: | (None, 20, 62, 62, 4)
max_pooling3d | input: | (None, 20, 62, 62, 4)
MaxPooling3D | output: | (None, 20, 31, 31, 4)
time_distributed(dropout) | input: | (None, 20, 31, 31, 4)
TimeDistributed(Dropout) | output: | (None, 20, 31, 31, 4)
conv_lstm2d_1 | input: | (None, 20, 31, 31, 4)
ConvLSTM2D | output: | (None, 20, 29, 29, 8)
max_pooling3d_1 | input: | (None, 20, 29, 29, 8)
MaxPooling3D | output: | (None, 20, 15, 15, 8)
time_distributed_1(dropout_1) | input: | (None, 20, 15, 15, 8)
TimeDistributed(Dropout) | output: | (None, 20, 15, 15, 8)
conv_lstm2d_2 | input: | (None, 20, 15, 15, 8)
ConvLSTM2D | output: | (None, 20, 13, 13, 14)

```
# Create an Instance of Early Stopping Callback
early_stopping_callback = EarlyStopping(monitor = 'val_loss', patience = 10, mode = 'min', restore_best_weights = True)

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

# Start training the model.
convlstm_model_training_history = convlstm_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
73/73 [==============================] - 172s 2s/step - loss: 1.3890 - accuracy: 0.2568 - val_loss: 1.3710 - val_accuracy: 0.4795
Epoch 2/50
73/73 [==============================] - 166s 2s/step - loss: 1.2871 - accuracy: 0.4452 - val_loss: 1.0778 - val_accuracy: 0.6027
Epoch 3/50
73/73 [==============================] - 159s 2s/step - loss: 1.1063 - accuracy: 0.5240 - val_loss: 1.0402 - val_accuracy: 0.6027
Epoch 4/50
73/73 [==============================] - 166s 2s/step - loss: 0.9332 - accuracy: 0.5856 - val_loss: 0.9463 - val_accuracy: 0.6301
Epoch 5/50
73/73 [==============================] - 159s 2s/step - loss: 0.8649 - accuracy: 0.6644 - val_loss: 0.7938 - val_accuracy: 0.7123
Epoch 6/50
73/73 [==============================] - 171s 2s/step - loss: 0.7271 - accuracy: 0.7055 - val_loss: 0.7218 - val_accuracy: 0.6849
Epoch 7/50
73/73 [==============================] - 171s 2s/step - loss: 0.6093 - accuracy: 0.7603 - val_loss: 0.6369 - val_accuracy: 0.7397
Epoch 8/50
73/73 [==============================] - 172s 2s/step - loss: 0.4812 - accuracy: 0.8082 - val_loss: 0.5891 - val_accuracy: 0.7123
Epoch 9/50
73/73 [==============================] - 169s 2s/step - loss: 0.4055 - accuracy: 0.8630 - val_loss: 0.6137 - val_accuracy: 0.7123
Epoch 10/50
73/73 [==============================] - 173s 2s/step - loss: 0.3166 - accuracy: 0.8664 - val_loss: 0.6434 - val_accuracy: 0.7260
Epoch 11/50
73/73 [==============================] - 171s 2s/step - loss: 0.3163 - accuracy: 0.8938 - val_loss: 0.8252 - val_accuracy: 0.6575
Epoch 12/50
73/73 [==============================] - 170s 2s/step - loss: 0.2710 - accuracy: 0.8973 - val_loss: 0.8275 - val_accuracy: 0.7534
Epoch 13/50
73/73 [==============================] - 154s 2s/step - loss: 0.1741 - accuracy: 0.9521 - val_loss: 0.6971 - val_accuracy: 0.7671
Epoch 14/50
73/73 [==============================] - 165s 2s/step - loss: 0.0915 - accuracy: 0.9760 - val_loss: 0.8422 - val_accuracy: 0.6849
Epoch 15/50
73/73 [==============================] - 164s 2s/step - loss: 0.1718 - accuracy: 0.9212 - val_loss: 0.6996 - val_accuracy: 0.7397
Epoch 16/50
73/73 [==============================] - 156s 2s/step - loss: 0.1175 - accuracy: 0.9623 - val_loss: 0.8650 - val_accuracy: 0.7397
Epoch 17/50
73/73 [==============================] - 166s 2s/step - loss: 0.0426 - accuracy: 0.9966 - val_loss: 0.7969 - val_accuracy: 0.7671
```

```
Epoch 18/50
73/73 [==============================] - 155s 2s/step - loss: 0.0934 - accuracy: 0.9692 - val_loss: 1.4612 - val_accuracy: 0.6164
```

Evaluate the Trained Model

After training, we will evaluate the model on the test set.

```
# Evaluate the trained model.
model_evaluation_history = convlstm_model.evaluate(features_test, labels_test)
```

```
4/4 [==============================] - 17s 4s/step - loss: 0.6172 - accuracy: 0.7705
```

Save the Model

Now we will save the model to avoid training it from scratch every time we need the model.

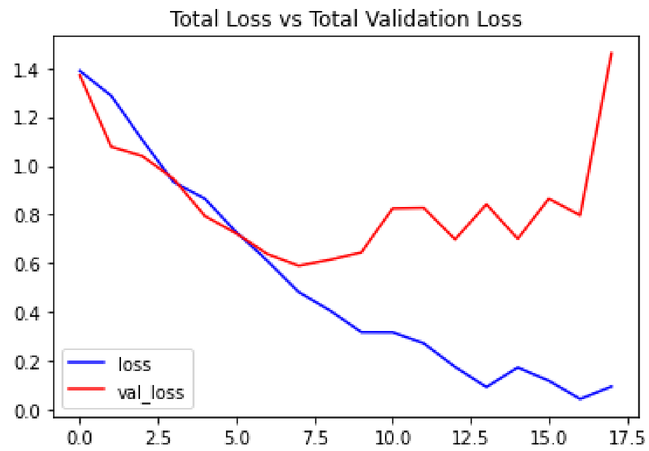
```
# Get the loss and accuracy from model_evaluation_history.
model_evaluation_loss, model_evaluation_accuracy = model_evaluation_history

# Define the string date format.
# Get the current Date and Time in a DateTime Object.
# Convert the DateTime object to string according to the style mentioned in date_time_format string.
date_time_format = '%Y_%m_%d__%H_%M_%S'
current_date_time_dt = dt.datetime.now()
current_date_time_string = dt.datetime.strftime(current_date_time_dt, date_time_format)

# Define a useful name for our model to make it easy for us while navigating through multiple saved models.
model_file_name = f'convlstm_model___Date_Time_{current_date_time_string}___Loss_{model_evaluation_loss}___Accuracy_{model_evaluation_accuracy}

# Save your Model.
convlstm_model.save(model_file_name)
```

```
def plot_metric(model_training_history, metric_name_1, metric_name_2, plot_name):
    '''
    This function will plot the metrics passed to it in a graph.
    Args:
        model_training_history: A history object containing a record of training and validation
                                loss values and metrics values at successive epochs
        metric_name_1:          The name of the first metric that needs to be plotted in the graph.
        metric_name_2:          The name of the second metric that needs to be plotted in the graph.
        plot_name:              The title of the graph.
```

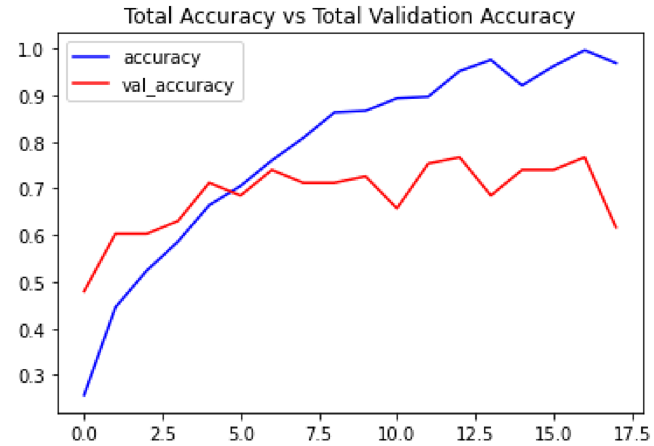
```
    '''

    # 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]

    # Construct a range object which will be used as x-axis (horizontal plane) of the graph.
    epochs = range(len(metric_value_1))

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

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

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

Now we will utilize the function **`plot_metric()`** created above, to visualize and understand the metrics.

```
# Visualize the training and validation loss metrices.
plot_metric(convlstm_model_training_history, 'loss', 'val_loss', 'Total Loss vs Total Validation Loss')
```




```
# Visualize the training and validation accuracy metrices.
plot_metric(convlstm_model_training_history, 'accuracy', 'val_accuracy', 'Total Accuracy vs Total Validation Accuracy')
```



LRCN Approach

```
def create_LRCN_model():
    '''
    This function will construct the required LRCN model.
    Returns:
        model: It is the required constructed LRCN model.
    '''

    # We will use a Sequential model for model construction.
    model = Sequential()

    # Define the Model Architecture.
    ########################################################################################################################

    model.add(TimeDistributed(Conv2D(16, (3, 3), padding='same',activation = 'relu'),
                              input_shape = (SEQUENCE_LENGTH, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.25)))

    model.add(TimeDistributed(Conv2D(32, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.25)))

    model.add(TimeDistributed(Conv2D(64, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.25)))
```

```
    model.add(TimeDistributed(Conv2D(64, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    #model.add(TimeDistributed(Dropout(0.25)))

    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(32))

    model.add(Dense(len(CLASSES_LIST), activation = 'softmax'))

    ########################################################################################################################

    # Display the models summary.
    model.summary()

    # Return the constructed LRCN model.
    return model
```

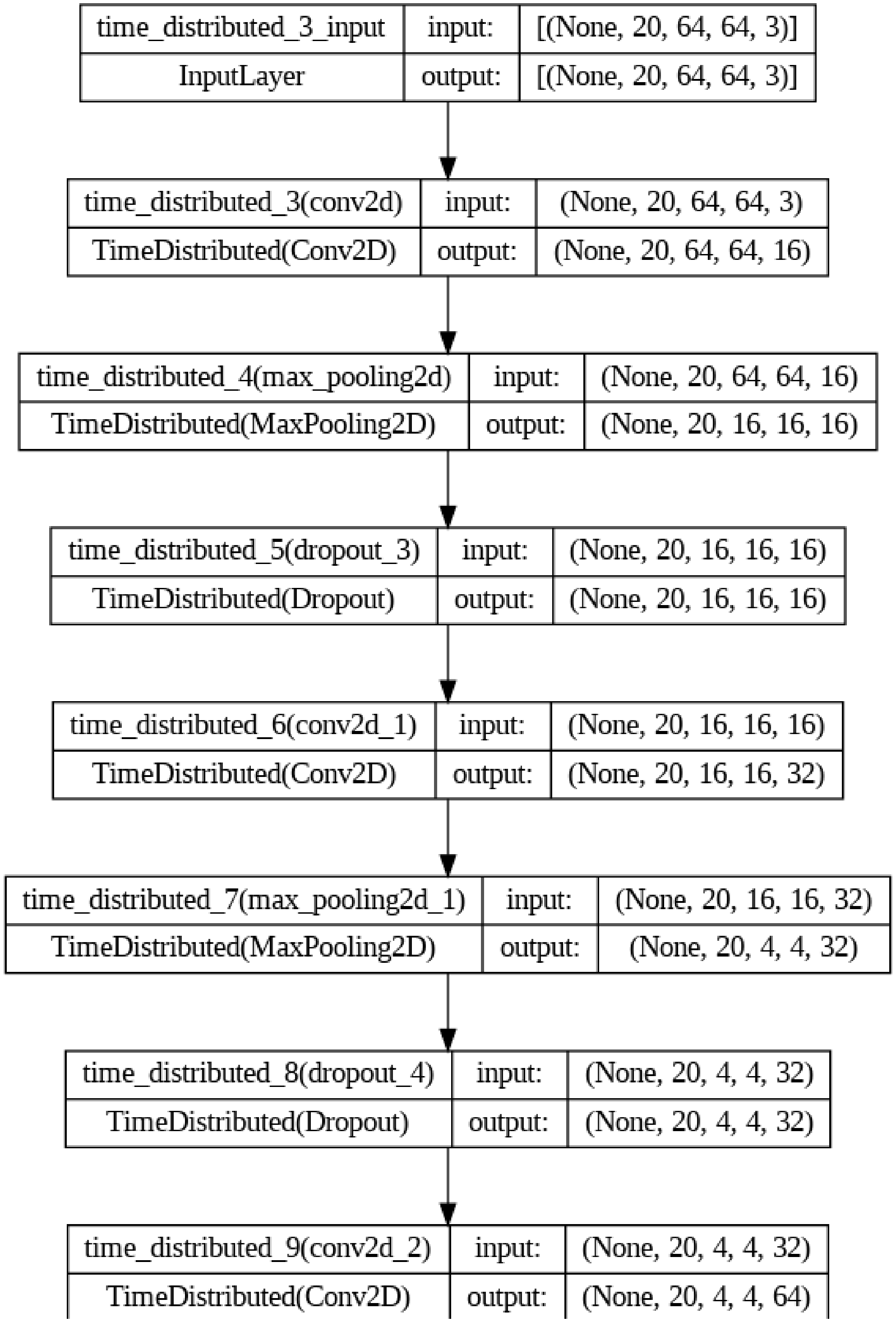
Now we will utilize the function **`create_LRCN_model()`** created above to construct the required `LRCN` model.

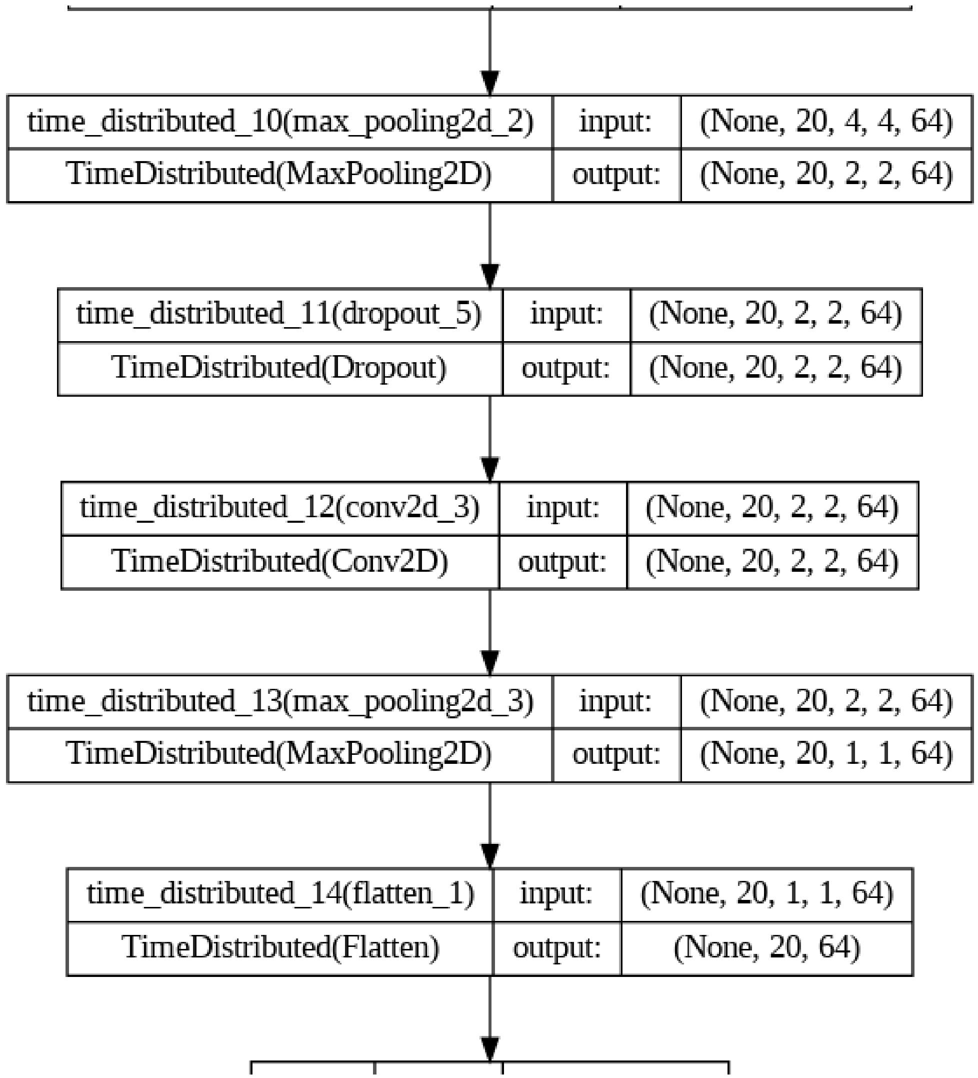
```
# Construct the required LRCN model.
LRCN_model = create_LRCN_model()

# Display the success message.
print("Model Created Successfully!")
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 time_distributed_3 (TimeDis  (None, 20, 64, 64, 16)   448       
 tributed)                                                       
                                                                 
 time_distributed_4 (TimeDis  (None, 20, 16, 16, 16)   0         
 tributed)                                                       
                                                                 
 time_distributed_5 (TimeDis  (None, 20, 16, 16, 16)   0         
 tributed)                                                       
                                                                 
 time_distributed_6 (TimeDis  (None, 20, 16, 16, 32)   4640      
 tributed)                                                       
                                                                 
 time_distributed_7 (TimeDis  (None, 20, 4, 4, 32)     0         
```

```
 tributed)

 time_distributed_8 (TimeDis  (None, 20, 4, 4, 32)     0
 tributed)

 time_distributed_9 (TimeDis  (None, 20, 4, 4, 64)     18496
 tributed)

 time_distributed_10 (TimeDi  (None, 20, 2, 2, 64)     0
 stributed)

 time_distributed_11 (TimeDi  (None, 20, 2, 2, 64)     0
 stributed)

 time_distributed_12 (TimeDi  (None, 20, 2, 2, 64)     36928
 stributed)

 time_distributed_13 (TimeDi  (None, 20, 1, 1, 64)     0
 stributed)

 time_distributed_14 (TimeDi  (None, 20, 64)           0
 stributed)

 lstm (LSTM)                 (None, 32)                12416

 dense_1 (Dense)             (None, 4)                 132

=================================================================
Total params: 73,060
Trainable params: 73,060
Non-trainable params: 0
_________________________________________________________________
Model Created Successfully!
```

```
# Plot the structure of the contructed LRCN model.
plot_model(LRCN_model, to_file = 'LRCN_model_structure_plot.png', show_shapes = True, show_layer_names = True)
```


time_distributed_3_input | input: | [(None, 20, 64, 64, 3)]
InputLayer | output: | [(None, 20, 64, 64, 3)]
time_distributed_3(conv2d) | input: | (None, 20, 64, 64, 3)
TimeDistributed(Conv2D) | output: | (None, 20, 64, 64, 16)
time_distributed_4(max_pooling2d) | input: | (None, 20, 64, 64, 16)
TimeDistributed(MaxPooling2D) | output: | (None, 20, 16, 16, 16)
time_distributed_5(dropout_3) | input: | (None, 20, 16, 16, 16)
TimeDistributed(Dropout) | output: | (None, 20, 16, 16, 16)
time_distributed_6(conv2d_1) | input: | (None, 20, 16, 16, 16)
TimeDistributed(Conv2D) | output: | (None, 20, 16, 16, 32)
time_distributed_7(max_pooling2d_1) | input: | (None, 20, 16, 16, 32)
TimeDistributed(MaxPooling2D) | output: | (None, 20, 4, 4, 32)
time_distributed_8(dropout_4) | input: | (None, 20, 4, 4, 32)
TimeDistributed(Dropout) | output: | (None, 20, 4, 4, 32)
time_distributed_9(conv2d_2) | input: | (None, 20, 4, 4, 32)
TimeDistributed(Conv2D) | output: | (None, 20, 4, 4, 64)




```
# Create an Instance of Early Stopping Callback.
early_stopping_callback = EarlyStopping(monitor = 'val_loss', patience = 15, mode = 'min', restore_best_weights = True)

# Compile the model and specify loss function, optimizer and metrics to the model.
LRCN_model.compile(loss = 'categorical_crossentropy', optimizer = 'Adam', metrics = ["accuracy"])

# Start training the model.
LRCN_model_training_history = LRCN_model.fit(x = features_train, y = labels_train, epochs = 70, batch_size = 4 ,
                                             shuffle = True, validation_split = 0.2, callbacks = [early_stopping_callback])
```

```
Epoch 10/70
73/73 [==============================] - 16s 222ms/step - loss: 0.5411 - accuracy: 0.8082 - val_loss: 0.7004 - val_accuracy: 0.7260
Epoch 11/70
73/73 [==============================] - 16s 226ms/step - loss: 0.4237 - accuracy: 0.8425 - val_loss: 0.6460 - val_accuracy: 0.7808
Epoch 12/70
73/73 [==============================] - 16s 226ms/step - loss: 0.3505 - accuracy: 0.8596 - val_loss: 0.4957 - val_accuracy: 0.8082
Epoch 13/70
73/73 [==============================] - 16s 227ms/step - loss: 0.3619 - accuracy: 0.8630 - val_loss: 0.4433 - val_accuracy: 0.8356
Epoch 14/70
73/73 [==============================] - 16s 225ms/step - loss: 0.2602 - accuracy: 0.9315 - val_loss: 0.5744 - val_accuracy: 0.7945
Epoch 15/70
73/73 [==============================] - 17s 227ms/step - loss: 0.2199 - accuracy: 0.9212 - val_loss: 0.5951 - val_accuracy: 0.8082
Epoch 16/70
73/73 [==============================] - 17s 238ms/step - loss: 0.2526 - accuracy: 0.8938 - val_loss: 0.3919 - val_accuracy: 0.8630
Epoch 17/70
73/73 [==============================] - 17s 235ms/step - loss: 0.2083 - accuracy: 0.9212 - val_loss: 0.3788 - val_accuracy: 0.8630
Epoch 18/70
73/73 [==============================] - 16s 225ms/step - loss: 0.2946 - accuracy: 0.9110 - val_loss: 0.4639 - val_accuracy: 0.8356
Epoch 19/70
73/73 [==============================] - 16s 224ms/step - loss: 0.1788 - accuracy: 0.9384 - val_loss: 0.5367 - val_accuracy: 0.8082
Epoch 20/70
73/73 [==============================] - 17s 229ms/step - loss: 0.1560 - accuracy: 0.9349 - val_loss: 0.5915 - val_accuracy: 0.7945
Epoch 21/70
73/73 [==============================] - 17s 228ms/step - loss: 0.1060 - accuracy: 0.9726 - val_loss: 0.3839 - val_accuracy: 0.8904
Epoch 22/70
73/73 [==============================] - 16s 227ms/step - loss: 0.0588 - accuracy: 0.9726 - val_loss: 0.2753 - val_accuracy: 0.8904
Epoch 23/70
73/73 [==============================] - 16s 226ms/step - loss: 0.0530 - accuracy: 0.9829 - val_loss: 0.3791 - val_accuracy: 0.8630
Epoch 24/70
73/73 [==============================] - 17s 228ms/step - loss: 0.0535 - accuracy: 0.9863 - val_loss: 0.4232 - val_accuracy: 0.8767
Epoch 25/70
73/73 [==============================] - 18s 249ms/step - loss: 0.5152 - accuracy: 0.8630 - val_loss: 0.6887 - val_accuracy: 0.7260
```

```
Epoch 34/70
73/73 [==============================] - 17s 236ms/step - loss: 0.0077 - accuracy: 1.0000 - val_loss: 0.4265 - val_accuracy: 0.9041
Epoch 35/70
73/73 [==============================] - 17s 227ms/step - loss: 0.0046 - accuracy: 1.0000 - val_loss: 0.4512 - val_accuracy: 0.9041
Epoch 36/70
73/73 [==============================] - 17s 227ms/step - loss: 0.0037 - accuracy: 1.0000 - val_loss: 0.4613 - val_accuracy: 0.9041
Epoch 37/70
73/73 [==============================] - 16s 226ms/step - loss: 0.0062 - accuracy: 0.9966 - val_loss: 0.4615 - val_accuracy: 0.8904
```

Evaluating the trained Model

As done for the previous one, we will evaluate the `LRCN` model on the test set.

```
# Evaluate the trained model.
model_evaluation_history = LRCN_model.evaluate(features_test, labels_test)
```

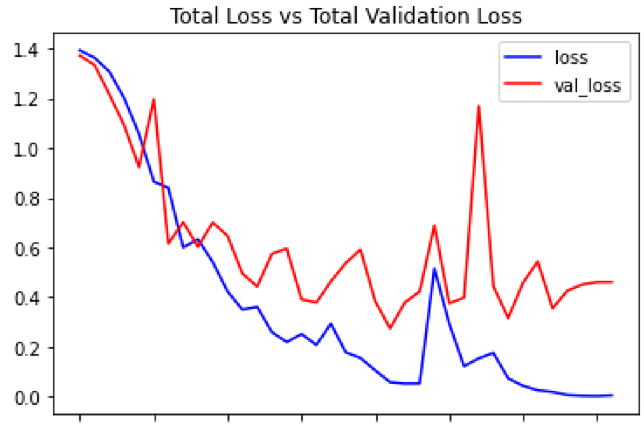
```
4/4 [==============================] - 3s 596ms/step - loss: 0.4431 - accuracy: 0.8607
```

Save the Model

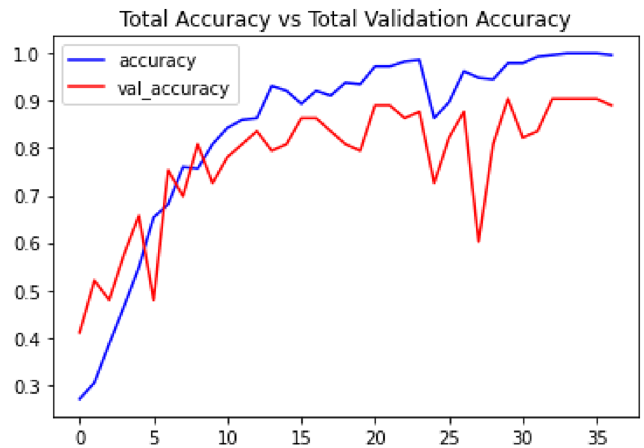
After that, we will save the model for future uses using the same technique we had used for the previous model.

```
# Get the loss and accuracy from model_evaluation_history.
model_evaluation_loss, model_evaluation_accuracy = model_evaluation_history

# Define the string date format.
# Get the current Date and Time in a DateTime Object.
# Convert the DateTime object to string according to the style mentioned in date_time_format string.
date_time_format = '%Y_%m_%d__%H_%M_%S'
current_date_time_dt = dt.datetime.now()
current_date_time_string = dt.datetime.strftime(current_date_time_dt, date_time_format)

# Define a useful name for our model to make it easy for us while navigating through multiple saved models.
model_file_name = f'LRCN_model___Date_Time_{current_date_time_string}___Loss_{model_evaluation_loss}___Accuracy_{model_evaluation_accuracy}.h5'

# Save the Model.
LRCN_model.save(model_file_name)
```

```
# Visualize the training and validation loss metrices.
plot_metric(LRCN_model_training_history, 'loss', 'val_loss', 'Total Loss vs Total Validation Loss')
```




```
# Visualize the training and validation accuracy metrices.
plot_metric(LRCN_model_training_history, 'accuracy', 'val_accuracy', 'Total Accuracy vs Total Validation Accuracy')
```



Create a Function To Perform Action Recognition on Videos

Next, we will create a function **`predict_on_video()`** that will simply read a video frame by frame from the path passed in as an argument and will perform action recognition on video and save the results.

```
def predict_on_video(SEQUENCE_LENGTH):
    '''
    This function will perform action recognition on a video using the LRCN model.
    Args:
    video_file_path:  The path of the video stored in the disk on which the action recognition is to be performed.
```

```
    output_file_path: The path where the ouput video with the predicted action being performed overlayed will be stored.
    SEQUENCE_LENGTH:  The fixed number of frames of a video that can be passed to the model as one sequence.
    '''

    # Initialize the VideoCapture object to read from the video file.
    video_reader = cv2.VideoCapture('videoplayback.mp4')

    # Get the width and height of the video.
    original_video_width = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))
    original_video_height = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # Initialize the VideoWriter Object to store the output video in the disk.
    #video_writer = cv2.VideoWriter('')

    # Declare a queue to store video frames.
    frames_queue = deque(maxlen = SEQUENCE_LENGTH)
    print(frames_queue)

    # Initialize a variable to store the predicted action being performed in the video.
    predicted_class_name = ''

    # Iterate until the video is accessed successfully.
    while video_reader.isOpened():

        # Read the frame.
        ok, frame = video_reader.read()

        # Check if frame is not read properly then break the loop.
        if not ok:
            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 pre-processed frame into the frames list.
        frames_queue.append(normalized_frame)
        print(len(frames_queue))

        # Check if the number of frames in the queue are equal to the fixed sequence length.
        if len(frames_queue) == SEQUENCE_LENGTH:
```

```
            # Pass the normalized frames to the model and get the predicted probabilities.
            predicted_labels_probabilities = LRCN_model.predict(np.expand_dims(frames_queue, axis = 0))[0]

            # Get the index of class with highest probability.
            predicted_label = np.argmax(predicted_labels_probabilities)

            # Get the class name using the retrieved index.
            predicted_class_name = CLASSES_LIST[predicted_label]

        # Write predicted class name on top of the frame.
        cv2.putText(frame, predicted_class_name, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        print(f'Action Predicted: {predicted_class_name}')
        # Write The frame into the disk using the VideoWriter Object.
        #video_writer.write(frame)

    # Release the VideoCapture and VideoWriter objects.
    video_reader.release()
    #video_writer.release()
```

Perform Action Recognition on the Test Video

Now we will utilize the function **`predict_on_video()`** created above to perform action recognition on the test video we had downloaded using the function **`download_youtube_videos()`** and display the output video with the predicted action overlayed on it.

```
# Construct the output video path.
#output_video_file_path = f'{test_videos_directory}/{video_title}-Output-SeqLen{SEQUENCE_LENGTH}.mp4'

# Perform Action Recognition on the Test Video.
predict_on_video(SEQUENCE_LENGTH)

# Display the output video.
VideoFileClip(output_video_file_path, audio=False, target_resolution=(300,None)).ipython_display()
```

```
deque([], maxlen=20)
1
Action Predicted:
2
Action Predicted:
3
Action Predicted:
4
Action Predicted:
5
Action Predicted:
6
Action Predicted:
7
Action Predicted:
8
Action Predicted:
9
Action Predicted:
10
Action Predicted:
11
Action Predicted:
12
Action Predicted:
13
Action Predicted:
14
Action Predicted:
15
Action Predicted:
16
Action Predicted:
17
Action Predicted:
18
Action Predicted:
19
Action Predicted:
20
1/1 [==============================] - 0s 47ms/step
Action Predicted: Swing
20
1/1 [==============================] - 0s 44ms/step
Action Predicted: Swing
20
1/1 [==============================] - 0s 44ms/step
Action Predicted: Swing
```

```
20
```

Create a Function To Perform a Single Prediction on Videos

Now let's create a function that will perform a single prediction for the complete videos. We will extract evenly distributed **N** **`(SEQUENCE_LENGTH)`** frames from the entire video and pass them to the `LRCN` model. This approach is really useful when you are working with videos containing only one activity as it saves unnecessary computations and time in that scenario.

```
20
```

```
def predict_single_action(video_file_path, SEQUENCE_LENGTH):
    '''
    This function will perform single action recognition prediction on a video using the LRCN model.
    Args:
    video_file_path:  The path of the video stored in the disk on which the action recognition is to be performed.
    SEQUENCE_LENGTH:  The fixed number of frames of a video that can be passed to the model as one sequence.
    '''

    # Initialize the VideoCapture object to read from the video file.
    video_reader = cv2.VideoCapture(video_file_path)

    # Get the width and height of the video.
    original_video_width = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))
    original_video_height = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # Declare a list to store video frames we will extract.
    frames_list = []

    # Initialize a variable to store the predicted action being performed in the video.
    predicted_class_name = ''

    # Get the number of frames in the video.
    video_frames_count = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))
    print(video_frames_count)

    # Calculate the interval after which frames will be added to the list.
    skip_frames_window = max(int(video_frames_count/SEQUENCE_LENGTH),1)
    print(skip_frames_window)

    # Iterating the number of times equal to the fixed length of sequence.
    for frame_counter in range(SEQUENCE_LENGTH):

        # Set the current frame position of the video.
        video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_counter * skip_frames_window)
```

```
        # Read a frame.
        success, frame = video_reader.read()

        # Check if frame is not read properly 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 pre-processed frame into the frames list
        frames_list.append(normalized_frame)

    # Passing the  pre-processed frames to the model and get the predicted probabilities.
    predicted_labels_probabilities = LRCN_model.predict(np.expand_dims(frames_list, axis = 0))[0]

    # Get the index of class with highest probability.
    predicted_label = np.argmax(predicted_labels_probabilities)

    # Get the class name using the retrieved index.
    predicted_class_name = CLASSES_LIST[predicted_label]

    cv2.putText(frame, predicted_class_name, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

    # Display the predicted action along with the prediction confidence.
    print(f'Action Predicted: {predicted_class_name}\nConfidence: {predicted_labels_probabilities[predicted_label]}')

    # Release the VideoCapture object.
    video_reader.release()
```

```
1/1 [==============================] - 0s 58ms/step
```

```
LRCN_model
```

```
<keras.engine.sequential.Sequential at 0x7f06a46a1c40>
```

Perform Single Prediction on a Test Video

Now we will utilize the function `predict_single_action()` created above to perform a single prediction on a complete youtube test video that we will download using the function `download_youtube_videos()`, we had created above.

```
1/1 [==============================] - 0s 43ms/step
```

```
input_video_file_path = 'videoplayback.mp4'

# Perform Single Prediction on the Test Video.
predict_single_action(input_video_file_path, SEQUENCE_LENGTH)

# Display the input video.
VideoFileClip(input_video_file_path, audio=False, target_resolution=(300,None)).ipython_display()
```

```
866
43
1/1 [==============================] - 0s 44ms/step
Action Predicted: HorseRace
Confidence: 0.9097224473953247
Moviepy - Building video __temp__.mp4.
Moviepy - Writing video __temp__.mp4

t:  99%|██████████| 859/869 [00:03<00:00, 294.53it/s, now=None]WARNING:py.warnings:/usr/local/lib/python3.9/dist-packages/moviepy/video/i
  warnings.warn("Warning: in file %s, "%(self.filename)+

WARNING:py.warnings:/usr/local/lib/python3.9/dist-packages/moviepy/video/io/ffmpeg_reader.py:123: UserWarning: Warning: in file videoplay
  warnings.warn("Warning: in file %s, "%(self.filename)+

WARNING:py.warnings:/usr/local/lib/python3.9/dist-packages/moviepy/video/io/ffmpeg_reader.py:123: UserWarning: Warning: in file videoplay
  warnings.warn("Warning: in file %s, "%(self.filename)+

Moviepy - Done !
Moviepy - video ready __temp__.mp4
```

0:00 / 0:31

```
Action Predicted: TaiChi
```