```
# Discards the output of the cell
%%capture

# To work with youtube videos
!pip install pafy youtube-dl moviepy
```

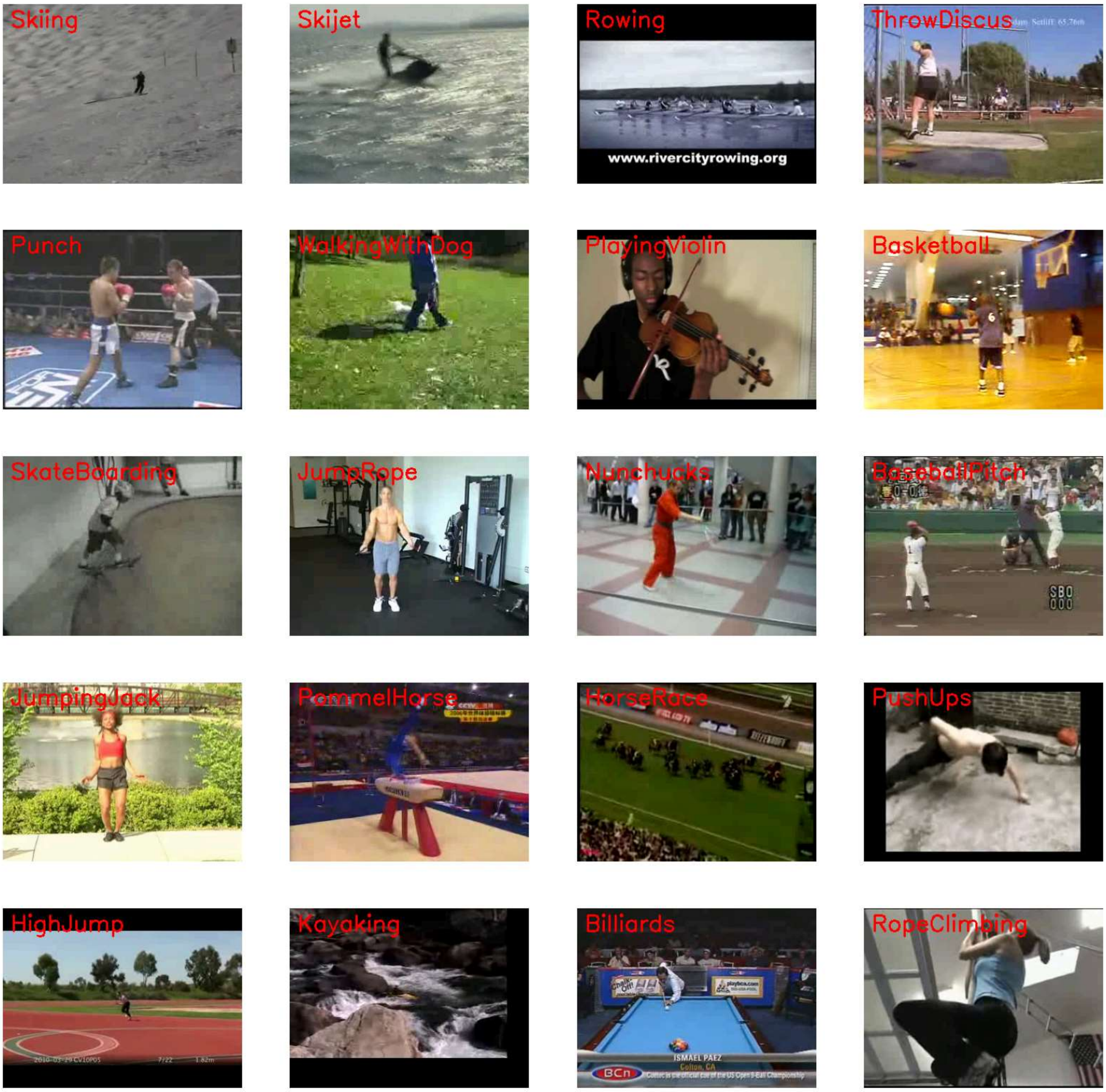
```
# libraries
import os
import cv2
import pafy
import math
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import deque

from moviepy.editor import *
%matplotlib inline

from sklearn.model_selection import train_test_split

import tensorflow as tf
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
```

```
# set Numpy, Python and Tensorflow seeds to get consistent results on every execution
# restricting the randomness
seed_constant = 27
np.random.seed(seed_constant)
random.seed(seed_constant)
tf.random.set_seed(seed_constant)
```

```
# Discards the output of the cell
%%capture

# Downloading the UCF50 Action dataset from the web
!wget --no-check-certificate https://www.crcv.ucf.edu/data/UCF50.rar

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

Visualizing the dataset

```python
# create and specify size of matplot figure
plt.figure(figsize = (20, 20))

# get names of all action categories
all_classes_names = os.listdir('UCF50')

# get 20 random categories
random_range = random.sample(range(len(all_classes_names)), 20)

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

    # get the name of the random category
    selected_class_Name = all_classes_names[random_index]

    # get the list of all the video files present in selected_class_Name
    video_files_names_list = os.listdir(f'UCF50/{selected_class_Name}')

    # randomly select a video file from the list
    selected_video_file_name = random.choice(video_files_names_list)

    # video capture object to read 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 video capture 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 (USED TO WRITE ON THE VIDEO)
    cv2.putText(rgb_frame, selected_class_Name, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)

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



Preprocess the data

```
from re import I
# resizing the video frames in our dataset
img_height, img_width = 64, 64

# specify the number of frames of a video that will be fed to the model as one sequence
sequence_length = 20

# specifying the dataset directory
dataset_dir = "UCF50"

# Specifying the list of classes used for training the model
classes_list = ["PullUps", "BenchPress", "Punch", "PlayingGuitar", "PushUps"]
```

Function to Extract , Resize and Normalize the frames

```
# function extracts the required frame from the video after resizing and normalizing it and then returns the a list of resized and norm
def frame_extraction(video_path):
  frames_list = []

  # reading the video file using the VideoCapture object
  video_reader = cv2.VideoCapture(video_path)

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

  # calculating the interval after which frames will be added
  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)

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

    # check if the frame is not successfully read
    if not success:
      break

    # resize the frame
    resized_frame = cv2.resize(frame, (img_height, img_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 video capture object
  video_reader.release()

  # return the frames list
  return frames_list
```

```
# Function for dataset creation
'''
this function returns:
  features: a list containing extracted frames of videos
  labels: a list containing the corresponding labels of the extracted frames
  video_files_paths: a list containing the paths of the videos from which the frames were extracted
'''


def create_dataset():
  features = []
  labels = []
  video_files_paths = []

  # iterating through all the classes mentioned in the class list
  for class_index, class_name in enumerate(classes_list):

    # Display name of the class
    print(f'Extracting data from class: {class_name}')

    # get the list of video files present in the class
    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 video file path
      video_file_path = os.path.join(dataset_dir, class_name, file_name)

      # extract the frames from the video file
      frames = frame_extraction(video_file_path)

      # check if the extracted frames is equal to the sequence length i.e. 20
      # ignore videos having frames less than 20
      if len(frames) == sequence_length:

        # append the data
        features.append(frames)
        labels.append(class_index)
        video_files_paths.append(video_file_path)

  # convert the features and labels lists to numpy arrays
  features = np.asarray(features)
  labels = np.array(labels)

  # return the features, labels and video files paths
  return features, labels, video_files_paths
```

creating the required dataset

```
features, labels, video_files_paths = create_dataset()
```

```
Extracting data from class: PullUps
Extracting data from class: BenchPress
Extracting data from class: Punch
Extracting data from class: PlayingGuitar
Extracting data from class: PushUps
```

```
# converting class indexes (labels) into one-hot encoded vectors
one_hot_encoded_labels = to_categorical(labels)

# it helps in processing the data faster, it converts the interger data to binary 0, 1
```

```
# Splitting data into train and test set
# data for training 75%
# data for testing 25%

features_train, features_test, labels_train, labels_test = train_test_split(features, one_hot_encoded_labels, test_size = 0.25, shuffle
```

Long-term Reccurence Convolutional Network Model

```
def create_LRCN_model():
  model = Sequential()

  model.add(TimeDistributed(Conv2D(16, (3, 3), padding = 'same', activation = 'relu'), input_shape = (sequence_length, img_height, img_
  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'))

  model.summary()

  return model
```

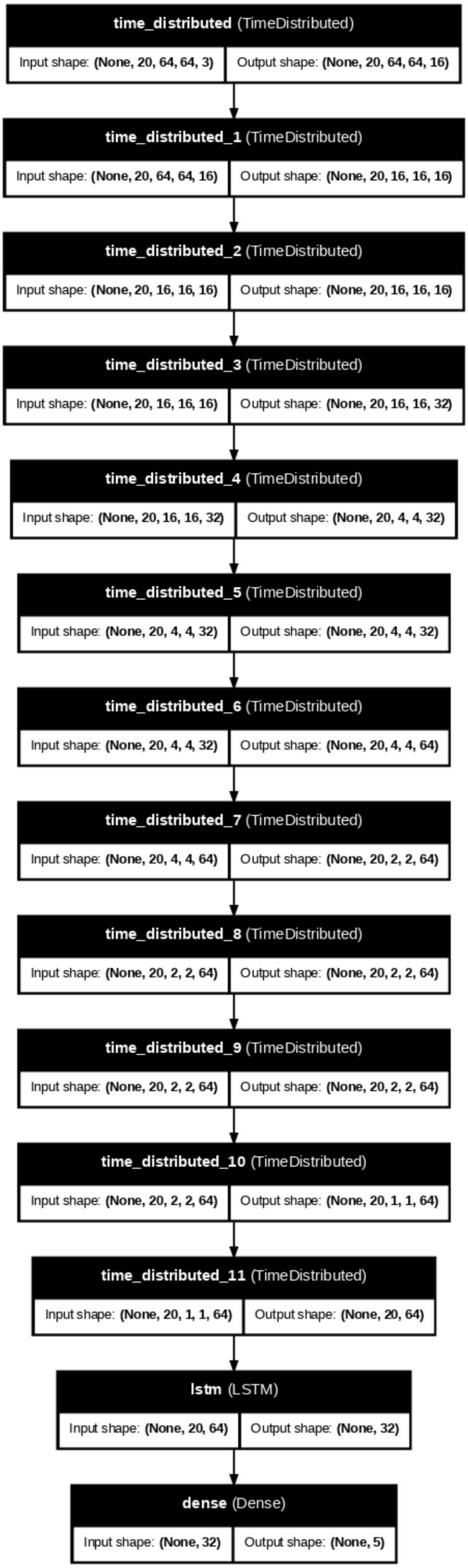
```
# Constructing model
LRCN_model = create_LRCN_model()
print("Model Created Successfully")
```

```
on3.10/dist-packages/keras/src/layers/core/wrapper.py:27: UserWarning: Do not pass an `input_shape`/`input_dim` a
**kwargs)
```

	Output Shape	Param #
(TimeDistributed)	(None, 20, 64, 64, 16)	448
1 (TimeDistributed)	(None, 20, 16, 16, 16)	0
2 (TimeDistributed)	(None, 20, 16, 16, 16)	0
3 (TimeDistributed)	(None, 20, 16, 16, 32)	4,640
4 (TimeDistributed)	(None, 20, 4, 4, 32)	0
5 (TimeDistributed)	(None, 20, 4, 4, 32)	0
6 (TimeDistributed)	(None, 20, 4, 4, 64)	18,496
7 (TimeDistributed)	(None, 20, 2, 2, 64)	0
8 (TimeDistributed)	(None, 20, 2, 2, 64)	0
9 (TimeDistributed)	(None, 20, 2, 2, 64)	36,928
10	(None, 20, 1, 1, 64)	0
11	(None, 20, 64)	0
	(None, 32)	12,416
	(None, 5)	165

```
93 (285.52 KB)
73,093 (285.52 KB)
ms: 0 (0.00 B)
ssfully
```

```
# Plotting the structure of the model
plot_model(LRCN_model, to_file = 'LRCN_model.png', show_shapes = True, show_layer_names = True, dpi = 66)
```


time_distributed (TimeDistributed)
Input shape: (None, 20, 64, 64, 3)
Output shape: (None, 20, 64, 64, 16)
time_distributed_1 (TimeDistributed)
Input shape: (None, 20, 64, 64, 16)
Output shape: (None, 20, 16, 16, 16)
time_distributed_2 (TimeDistributed)
Input shape: (None, 20, 16, 16, 16)
Output shape: (None, 20, 16, 16, 16)
time_distributed_3 (TimeDistributed)
Input shape: (None, 20, 16, 16, 16)
Output shape: (None, 20, 16, 16, 32)
time_distributed_4 (TimeDistributed)
Input shape: (None, 20, 16, 16, 32)
Output shape: (None, 20, 4, 4, 32)
time_distributed_5 (TimeDistributed)
Input shape: (None, 20, 4, 4, 32)
Output shape: (None, 20, 4, 4, 32)
time_distributed_6 (TimeDistributed)
Input shape: (None, 20, 4, 4, 32)
Output shape: (None, 20, 4, 4, 64)
time_distributed_7 (TimeDistributed)
Input shape: (None, 20, 4, 4, 64)
Output shape: (None, 20, 2, 2, 64)
time_distributed_8 (TimeDistributed)
Input shape: (None, 20, 2, 2, 64)
Output shape: (None, 20, 2, 2, 64)
time_distributed_9 (TimeDistributed)
Input shape: (None, 20, 2, 2, 64)
Output shape: (None, 20, 2, 2, 64)
time_distributed_10 (TimeDistributed)
Input shape: (None, 20, 2, 2, 64)
Output shape: (None, 20, 1, 1, 64)
time_distributed_11 (TimeDistributed)
Input shape: (None, 20, 1, 1, 64)
Output shape: (None, 20, 64)
lstm (LSTM)
Input shape: (None, 20, 64)
Output shape: (None, 32)
dense (Dense)
Input shape: (None, 32)
Output shape: (None, 5)

Compiling and Training the model

```
#Compiling and Training
early_stopping_callback = EarlyStopping(monitor = 'val_loss', patience = 15, mode = 'min', restore_best_weights = True)

LRCN_model.compile(loss = 'categorical_crossentropy', optimizer = 'Adam', metrics = ["accuracy"])

LRCN_model_training_history = LRCN_model.fit(x = features_train, y = labels_train, epochs = 50, batch_size = 4, shuffle = True, validat
```

```
Epoch 13/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 52s 493ms/step - accuracy: 0.8636 - loss: 0.4060 - val_accuracy: 0.7642 - val_loss: 0.4879
Epoch 14/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 84s 514ms/step - accuracy: 0.8825 - loss: 0.3471 - val_accuracy: 0.8019 - val_loss: 0.5516
Epoch 15/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 84s 541ms/step - accuracy: 0.8638 - loss: 0.3950 - val_accuracy: 0.8774 - val_loss: 0.3922
Epoch 16/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 80s 525ms/step - accuracy: 0.9446 - loss: 0.1640 - val_accuracy: 0.8585 - val_loss: 0.3572
Epoch 17/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 86s 565ms/step - accuracy: 0.9610 - loss: 0.1479 - val_accuracy: 0.8868 - val_loss: 0.2767
Epoch 18/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 87s 616ms/step - accuracy: 0.9703 - loss: 0.1265 - val_accuracy: 0.9528 - val_loss: 0.1804
Epoch 19/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 71s 514ms/step - accuracy: 0.9606 - loss: 0.1376 - val_accuracy: 0.9434 - val_loss: 0.2132
Epoch 20/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 97s 658ms/step - accuracy: 0.9800 - loss: 0.0973 - val_accuracy: 0.7547 - val_loss: 0.7290
Epoch 21/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 69s 535ms/step - accuracy: 0.9061 - loss: 0.2602 - val_accuracy: 0.8585 - val_loss: 0.3987
Epoch 22/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 78s 496ms/step - accuracy: 0.9617 - loss: 0.1047 - val_accuracy: 0.9245 - val_loss: 0.2539
Epoch 23/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 86s 529ms/step - accuracy: 0.9890 - loss: 0.0582 - val_accuracy: 0.9245 - val_loss: 0.2439
Epoch 24/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 54s 511ms/step - accuracy: 0.9879 - loss: 0.0680 - val_accuracy: 0.8396 - val_loss: 0.5938
Epoch 25/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 86s 551ms/step - accuracy: 0.9347 - loss: 0.1752 - val_accuracy: 0.8868 - val_loss: 0.3083
Epoch 26/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 82s 553ms/step - accuracy: 0.9530 - loss: 0.1842 - val_accuracy: 0.9811 - val_loss: 0.1021
Epoch 27/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 60s 567ms/step - accuracy: 0.9915 - loss: 0.0339 - val_accuracy: 0.9623 - val_loss: 0.1754
Epoch 28/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 80s 546ms/step - accuracy: 0.9424 - loss: 0.1720 - val_accuracy: 0.9340 - val_loss: 0.2635
Epoch 29/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 83s 556ms/step - accuracy: 0.9758 - loss: 0.0897 - val_accuracy: 0.9623 - val_loss: 0.1439
Epoch 30/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 75s 485ms/step - accuracy: 1.0000 - loss: 0.0112 - val_accuracy: 0.9717 - val_loss: 0.1261
Epoch 31/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 82s 489ms/step - accuracy: 1.0000 - loss: 0.0077 - val_accuracy: 0.9717 - val_loss: 0.1321
Epoch 32/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 58s 548ms/step - accuracy: 1.0000 - loss: 0.0064 - val_accuracy: 0.9717 - val_loss: 0.1376
Epoch 33/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 94s 665ms/step - accuracy: 1.0000 - loss: 0.0052 - val_accuracy: 0.9717 - val_loss: 0.1387
Epoch 34/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 64s 603ms/step - accuracy: 1.0000 - loss: 0.0045 - val_accuracy: 0.9717 - val_loss: 0.1410
Epoch 35/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 80s 588ms/step - accuracy: 1.0000 - loss: 0.0042 - val_accuracy: 0.9717 - val_loss: 0.1442
Epoch 36/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 80s 573ms/step - accuracy: 1.0000 - loss: 0.0035 - val_accuracy: 0.9717 - val_loss: 0.1410
Epoch 37/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 58s 552ms/step - accuracy: 1.0000 - loss: 0.0031 - val_accuracy: 0.9717 - val_loss: 0.1429
Epoch 38/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 79s 520ms/step - accuracy: 1.0000 - loss: 0.0028 - val_accuracy: 0.9717 - val_loss: 0.1441
Epoch 39/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 87s 561ms/step - accuracy: 1.0000 - loss: 0.0025 - val_accuracy: 0.9717 - val_loss: 0.1446
Epoch 40/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 81s 560ms/step - accuracy: 1.0000 - loss: 0.0023 - val_accuracy: 0.9717 - val_loss: 0.1471
Epoch 41/50
106/106 ━━━━━━━━━━━━━━━━━━━━ 85s 581ms/step - accuracy: 1.0000 - loss: 0.0020 - val_accuracy: 0.9717 - val_loss: 0.1487
```

Evaluating the trained model

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

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 3s 470ms/step - accuracy: 0.9381 - loss: 0.2794
```

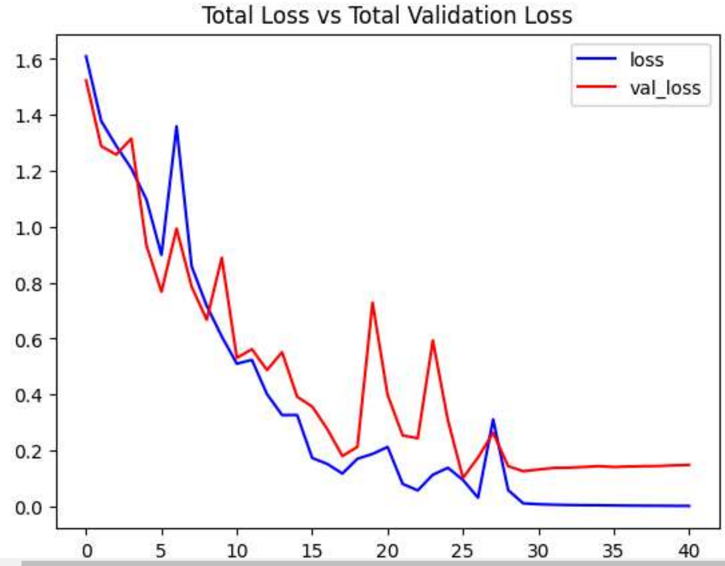
```
# saving the model
LRCN_model.save('LRCN_Human_Activity_Recognition_System.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is
```

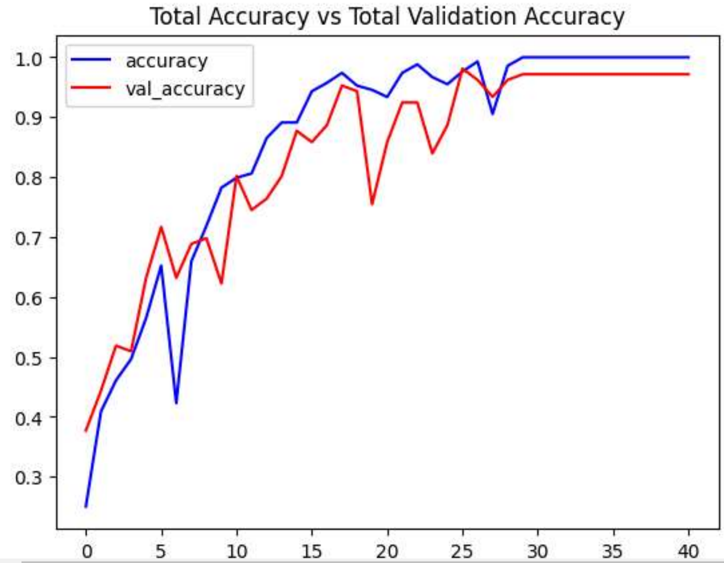
Model Accuracy and Loss Curves

```
def plot_metric(model_training_history, metric_name1, metric_name2, plot_name):
  metric_value1 = model_training_history.history[metric_name1]
  metric_value2 = model_training_history.history[metric_name2]

  epochs = range(len(metric_value1))

  plt.plot(epochs, metric_value1, 'blue', label = metric_name1)
  plt.plot(epochs, metric_value2, 'red', label = metric_name2)

  plt.title(str(plot_name))
  plt.legend()
  plt.show()
```

```
# Graph for Total Loss vs Validation Loss
plot_metric(LRCN_model_training_history, 'loss', 'val_loss', 'Total Loss vs Total Validation Loss')
```




```
# Graph for Total Accuracy vs Validation Accuracy
plot_metric(LRCN_model_training_history, 'accuracy', 'val_accuracy', 'Total Accuracy vs Total Validation Accuracy')
```



Implementing functions to deal with youtube videos

```
'''
    The function downloads the youtube video whose URL is passed to it as an argument.
    Args:
        youtube_video_url: URL of the video that is required to be downloaded.
        output_directory:  The directory path to which the video needs to be stored after downloading.
    It returns the title of the downloaded youtube video.
    '''
def download_youtube_videos(youtube_video_url, output_directory):
  import youtube_dl

  # Create a video object which contains useful information about the video.
  ydl_opts = {'quiet': True, 'verbose': True}  # Add verbose flag
  with youtube_dl.YoutubeDL(ydl_opts) as ydl:
    info_dict = ydl.extract_info(youtube_video_url, download=False)
    video_title = info_dict.get('title', None)

     # Create a video object which contains useful information about the video.
  video = pafy.new(youtube_video_url)

  # Get the best available quality object for the video.
  video_best = video.getbest()

  # Construct the output file path.
  output_file_path = f'{output_directory}/{video_title}.mp4'

# Make the output directory if it does not exist
test_videos_directory = 'test_videos'
os.makedirs(test_videos_directory, exist_ok = True)
  return video_title

# downloading a video
video_title = download_youtube_videos('https://www.youtube.com/watch?v=8u0qjmHIOcE', test_videos_directory)

# Get the YouTube Video's path we just downloaded.
input_video_file_path = f'{test_videos_directory}/{video_title}.mp4'
```