```
!pip install pafy youtube-dl moviepy
!pip install imageio-ffmpeg
!pip3 install imageio==2.4.1
```

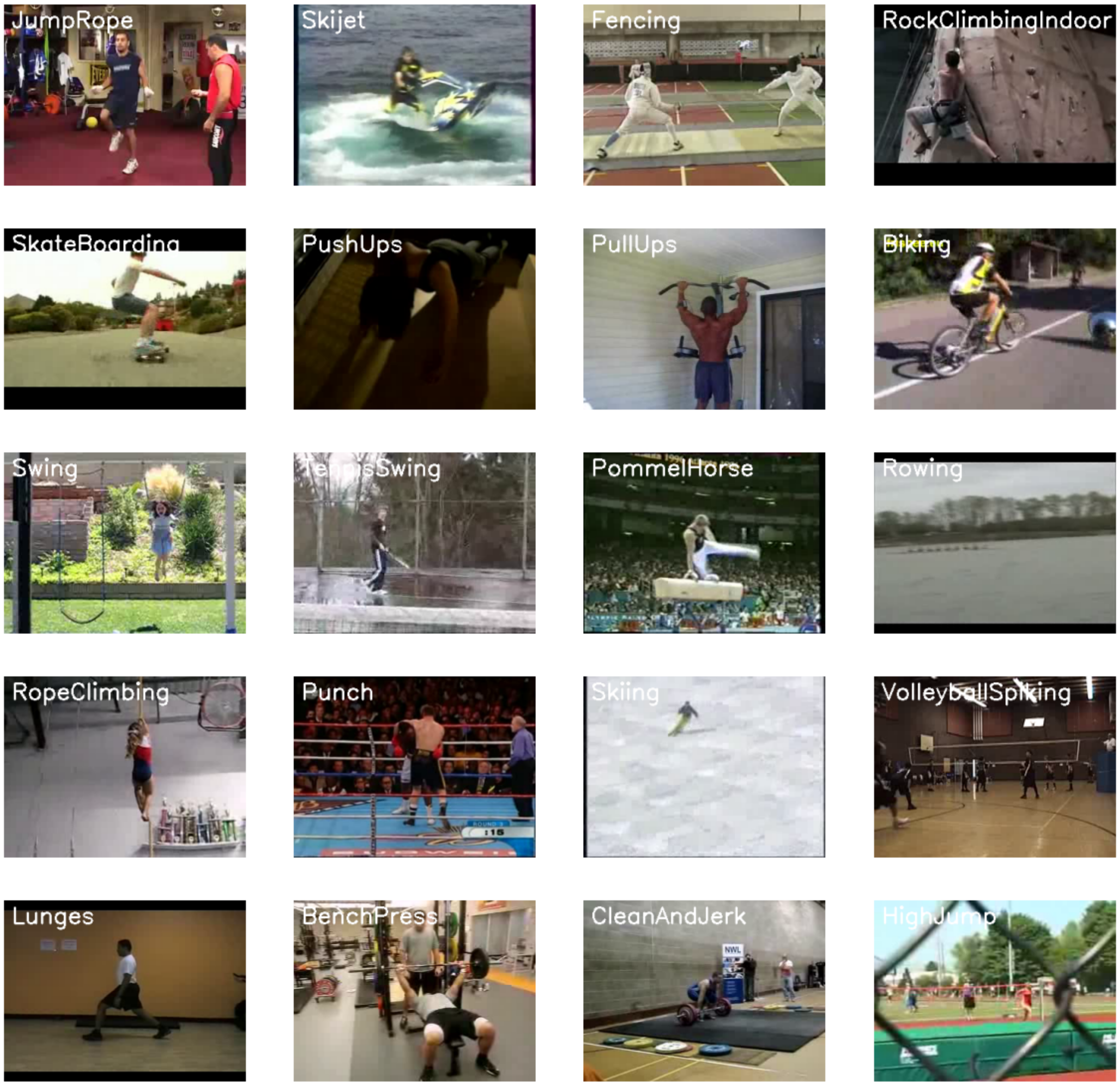
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publ
Requirement already satisfied: pafy in /usr/local/lib/python3.7/dist-packages (0.5.5)
Requirement already satisfied: youtube-dl in /usr/local/lib/python3.7/dist-packages (202
Requirement already satisfied: moviepy in /usr/local/lib/python3.7/dist-packages (0.2.3
Requirement already satisfied: decorator<5.0,>=4.0.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from mov
Requirement already satisfied: tqdm<5.0,>=4.11.2 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: imageio<3.0,>=2.1.2 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: pillow in /usr/local/lib/python3.7/dist-packages (from in
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publ
Requirement already satisfied: imageio-ffmpeg in /usr/local/lib/python3.7/dist-packages
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publ
Requirement already satisfied: imageio==2.4.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from ima
Requirement already satisfied: pillow in /usr/local/lib/python3.7/dist-packages (from in
```

```
# 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
```

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

```
# Discard the output of this cell.
%%capture

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

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

```
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 Directo
    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 D
    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,

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


JumpRope
Skijet
Fencing
RockClimbingIndoor
SkateBoarding
PushUps
PullUps
Biking
Swing
TennisSwing
PommelHorse
Rowing
RopeClimbing
Punch
Skiing
VolleyballSpiking
Lunges
BenchPress
CleanAndJerk
HighJump

```
# 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
CLASSES_LIST = ["WalkingWithDog", "TaiChi", "Swing", "HorseRace"]


def frames_extraction(video_path):
    '''
    This function will extract the required frames from a video after resizing and normalizin
    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 l
```

```
        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


def create_dataset():
    '''
    This function will extract the data of the selected classes and create the required datas
    Returns:
        features:          A list containing the extracted frames of the videos.
        labels:            A list containing the indexes of the classes associated with the v
        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


# 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
```

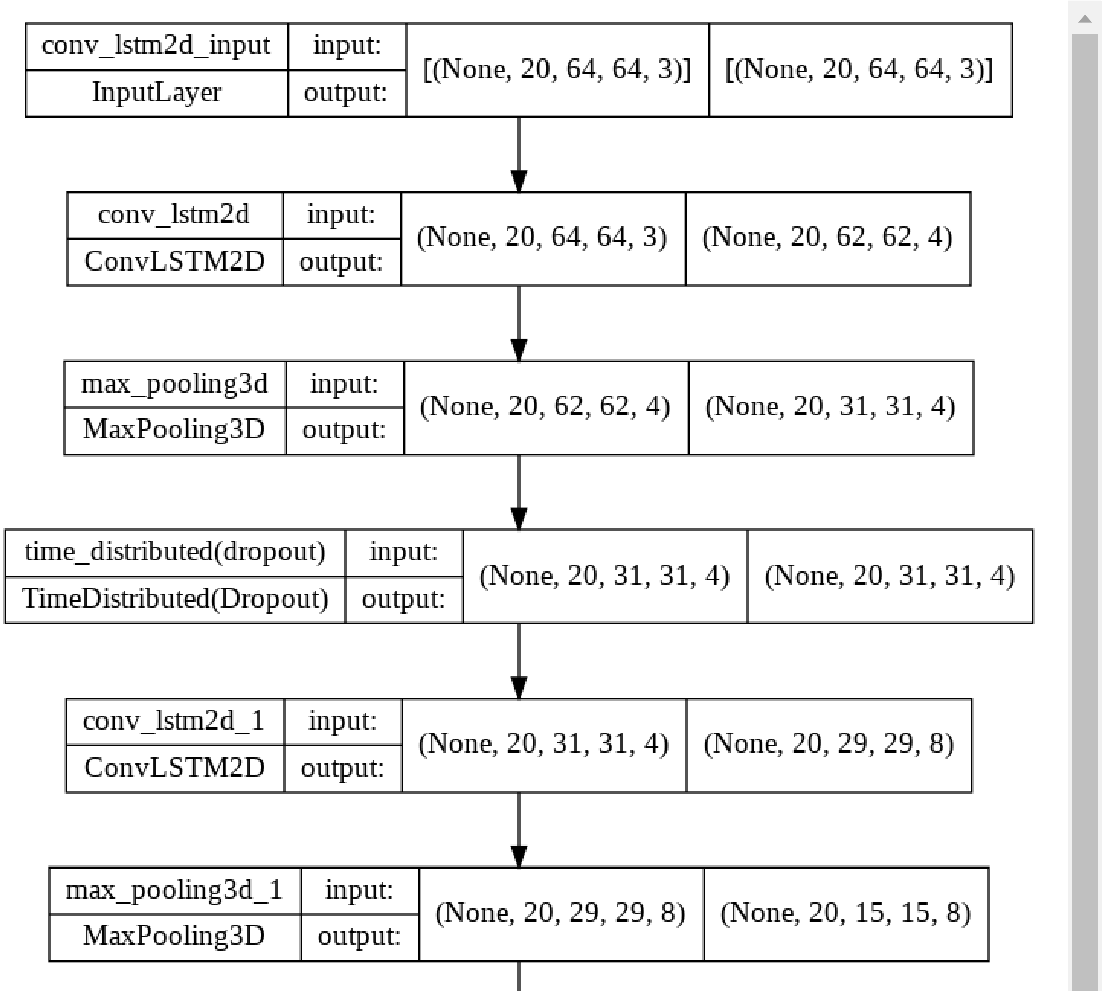
```
# Using Keras's to_categorical method to convert labels into one-hot-encoded vectors
one_hot_encoded_labels = to_categorical(labels)


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 =
                         recurrent_dropout=0.2, return_sequences=True, input_shape = (SEQUENC
                                                                                     IMAGE_H

    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
                         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
                         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
                         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


# 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!
```

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




```
1
2
# Split the Data into Train ( 75% ) and Test Set ( 25% ).
features_train, features_test, labels_train, labels_test = train_test_split(features, one_hot
```

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

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

# Start training the model.
convlstm_model_training_history = convlstm_model.fit(x = features_train, y = labels_train, ep
```

```
Epoch 1/50
73/73 [==============================] - 148s 2s/step - loss: 1.3874 - accuracy: 0.2979
Epoch 2/50
73/73 [==============================] - 138s 2s/step - loss: 1.3407 - accuracy: 0.3322
Epoch 3/50
73/73 [==============================] - 139s 2s/step - loss: 1.2268 - accuracy: 0.4760
Epoch 4/50
73/73 [==============================] - 139s 2s/step - loss: 0.9917 - accuracy: 0.6096
Epoch 5/50
```

```
73/73 [==============================] - 139s 2s/step - loss: 0.7571 - accuracy: 0.6918
Epoch 6/50
73/73 [==============================] - 139s 2s/step - loss: 0.6136 - accuracy: 0.7534
Epoch 7/50
73/73 [==============================] - 139s 2s/step - loss: 0.5121 - accuracy: 0.7979
Epoch 8/50
73/73 [==============================] - 139s 2s/step - loss: 0.3390 - accuracy: 0.8699
Epoch 9/50
73/73 [==============================] - 140s 2s/step - loss: 0.2729 - accuracy: 0.9007
Epoch 10/50
73/73 [==============================] - 138s 2s/step - loss: 0.2915 - accuracy: 0.8801
Epoch 11/50
73/73 [==============================] - 138s 2s/step - loss: 0.2309 - accuracy: 0.9041
Epoch 12/50
73/73 [==============================] - 139s 2s/step - loss: 0.0646 - accuracy: 0.9863
Epoch 13/50
73/73 [==============================] - 139s 2s/step - loss: 0.0747 - accuracy: 0.9760
Epoch 14/50
73/73 [==============================] - 139s 2s/step - loss: 0.0812 - accuracy: 0.9692
Epoch 15/50
73/73 [==============================] - 139s 2s/step - loss: 0.0682 - accuracy: 0.9863
Epoch 16/50
73/73 [==============================] - 139s 2s/step - loss: 0.0914 - accuracy: 0.9658
Epoch 17/50
73/73 [==============================] - 139s 2s/step - loss: 0.0479 - accuracy: 0.9897
Epoch 18/50
73/73 [==============================] - 138s 2s/step - loss: 0.0147 - accuracy: 1.0000
Epoch 19/50
73/73 [==============================] - 139s 2s/step - loss: 0.0228 - accuracy: 0.9966
Epoch 20/50
73/73 [==============================] - 138s 2s/step - loss: 0.1182 - accuracy: 0.9623
Epoch 21/50
73/73 [==============================] - 138s 2s/step - loss: 0.0846 - accuracy: 0.9795
Epoch 22/50
73/73 [==============================] - 138s 2s/step - loss: 0.0529 - accuracy: 0.9760
Epoch 23/50
73/73 [==============================] - 138s 2s/step - loss: 0.0284 - accuracy: 0.9897
```

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

```
4/4 [==============================] - 14s 3s/step - loss: 0.7390 - accuracy: 0.8033
```

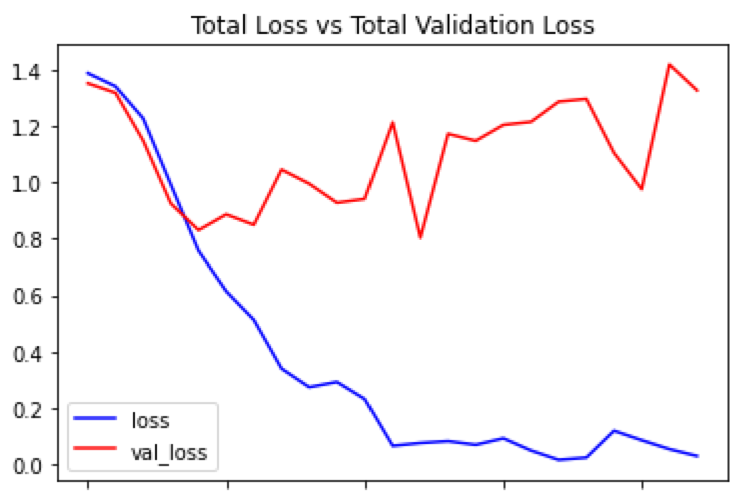
```
# Get the loss and accuracy from model_evaluation_history.
model_evaluation_loss, model_evaluation_accuracy = model_evaluation_history

# 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
```

```
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
model_file_name = f'convlstm_model___Date_Time_{current_date_time_string}___Loss_{model_evalu

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

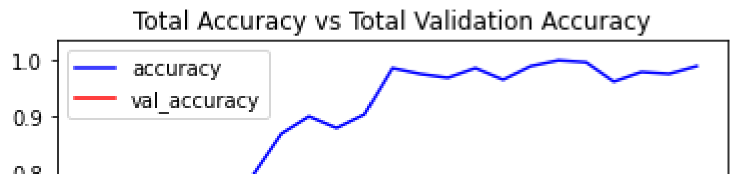
```
def plot_metric(model_training_history, metric_name_1, metric_name_2, plot_name):
    '''
    This function will plot the metrics passed to it in a graph.
    Args:
        model_training_history: A history object containing a record of training and validati
                                loss values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

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



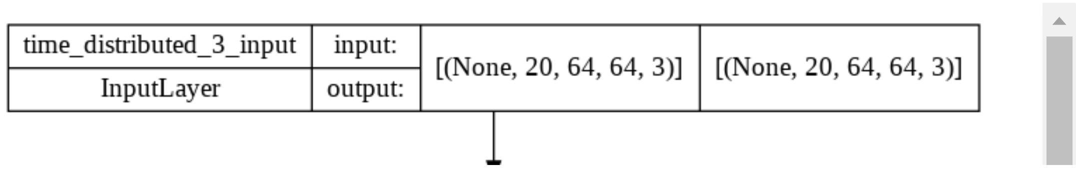

```
# Visualize the training and validation accuracy metrices.
plot_metric(convlstm_model_training_history, 'accuracy', 'val_accuracy', 'Total Accuracy vs T
```


Total Accuracy vs Total Validation Accuracy
accuracy
val_accuracy
1.0
0.9

```python
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


# 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_la
```




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

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

# Start training the model.
LRCN_model_training_history = LRCN_model.fit(x = features_train, y = labels_train, epochs = 5
```

```
Epoch 1/50
73/73 [==============================] - 16s 188ms/step - loss: 1.4006 - accuracy: 0.274
Epoch 2/50
73/73 [==============================] - 13s 177ms/step - loss: 1.3551 - accuracy: 0.291
Epoch 3/50
73/73 [==============================] - 13s 178ms/step - loss: 1.2244 - accuracy: 0.455
Epoch 4/50
73/73 [==============================] - 13s 178ms/step - loss: 1.1435 - accuracy: 0.516
Epoch 5/50
73/73 [==============================] - 13s 179ms/step - loss: 1.0011 - accuracy: 0.602
Epoch 6/50
73/73 [==============================] - 13s 178ms/step - loss: 0.9180 - accuracy: 0.616
Epoch 7/50
73/73 [==============================] - 13s 178ms/step - loss: 0.9012 - accuracy: 0.616
Epoch 8/50
73/73 [==============================] - 13s 177ms/step - loss: 0.9127 - accuracy: 0.661
Epoch 9/50
73/73 [==============================] - 13s 178ms/step - loss: 0.7579 - accuracy: 0.688
Epoch 10/50
73/73 [==============================] - 13s 178ms/step - loss: 0.6774 - accuracy: 0.732
Epoch 11/50
73/73 [==============================] - 13s 177ms/step - loss: 0.5738 - accuracy: 0.787
Epoch 12/50
73/73 [==============================] - 13s 178ms/step - loss: 0.5762 - accuracy: 0.791
Epoch 13/50
73/73 [==============================] - 13s 176ms/step - loss: 0.5474 - accuracy: 0.791
Epoch 14/50
73/73 [==============================] - 13s 177ms/step - loss: 0.3842 - accuracy: 0.873
Epoch 15/50
73/73 [==============================] - 13s 177ms/step - loss: 0.3665 - accuracy: 0.869
Epoch 16/50
```

```
73/73 [==============================] - 13s 177ms/step - loss: 0.3413 - accuracy: 0.887
Epoch 17/50
73/73 [==============================] - 13s 178ms/step - loss: 0.2634 - accuracy: 0.907
Epoch 18/50
73/73 [==============================] - 13s 177ms/step - loss: 0.2320 - accuracy: 0.928
Epoch 19/50
73/73 [==============================] - 13s 178ms/step - loss: 0.2178 - accuracy: 0.938
Epoch 20/50
73/73 [==============================] - 13s 178ms/step - loss: 0.3693 - accuracy: 0.886
Epoch 21/50
73/73 [==============================] - 13s 178ms/step - loss: 0.1706 - accuracy: 0.948
Epoch 22/50
55/73 [=====================>........] - ETA: 2s - loss: 0.2385 - accuracy: 0.9227
```

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

```
# 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
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
model_file_name = f'LRCN_model___Date_Time_{current_date_time_string}___Loss_{model_evaluatio
```

```
# 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
```

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

```
def download_youtube_videos(youtube_video_url, output_directory):
     '''
    This 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 dow
```

```
    Returns:
        title: The title of the downloaded youtube video.
    '''

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

    # Retrieve the title of the video.
    title = video.title

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

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

    # Download the youtube video at the best available quality and store it to the contructe
    video_best.download(filepath = output_file_path, quiet = True)

    # Return the video title.
    return title
```

```
!pip install git+https://github.com/Cupcakus/pafy
```

```
!pip uninstall -y pafy
!pip install git+https://github.com/Cupcakus/pafy
```

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


# Download a YouTube Video.
video_title = download_youtube_videos('https://www.youtube.com/watch?v=8u0qjmHIOcE', test_vid

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

```
def predict_on_video(video_file_path, output_file_path, 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 recognitio
    output_file_path: The path where the ouput video with the predicted action being performe
    SEQUENCE_LENGTH:  The fixed number of frames of a video that can be passed to the model a
    '''

    # 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))

    # Initialize the VideoWriter Object to store the output video in the disk.
    video_writer = cv2.VideoWriter(output_file_path, cv2.VideoWriter_fourcc('M', 'P', '4', 'V
                                   video_reader.get(cv2.CAP_PROP_FPS), (original_video_width,

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

    # 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 l
        normalized_frame = resized_frame / 255

        # Appending the pre-processed frame into the frames list.
        frames_queue.append(normalized_frame)

        # 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,

            # 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, 2

        # 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()
```

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

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

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

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

    # 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))

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

    # 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 l
        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 = convlstm_model.predict(np.expand_dims(frames_list, axis

    # 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]

    # Display the predicted action along with the prediction confidence.
    print(f'Action Predicted: {predicted_class_name}\nConfidence: {predicted_labels_probabili

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

```
# Download the youtube video.
video_title = download_youtube_videos('https://www.youtube.com/watch?v=XqqpZS0c1K0', test_vid

# Construct tihe nput youtube video path
input_video_file_path = f'{test_videos_directory}/{video_title}.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_displ
```