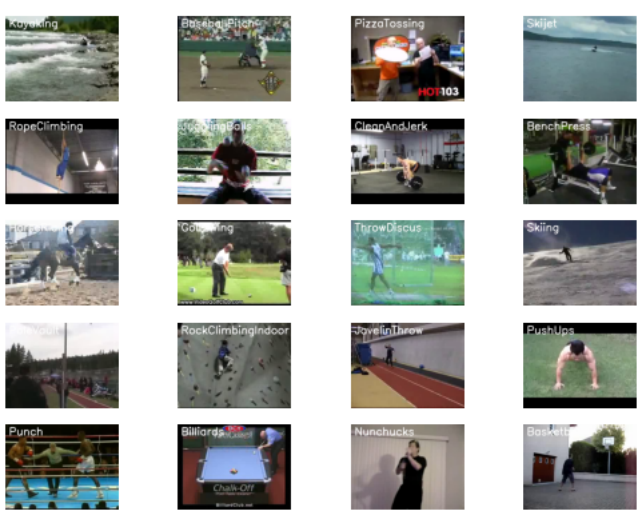
```
%%capture
!pip install tensorflow opencv-contrib-python youtube-dl moviepy pydot
!pip install git+https://github.com/TahaAnwar/pafy.git#egg=pafy
import os
import cv2
import pafy
import math
import random
import numpy as np
import datetime as dt
import tensorflow as tf
from collections import deque
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)       #numpy module
random.seed(seed_constant)          #random module
tf.random.set_seed(seed_constant)   #tensor flow module


%%capture
!wget --no-check-certificate https://www.crcv.ucf.edu/data/UCF50.rar
!unrar x UCF50.rar



plt.figure(figsize=(10,10))
# Check the current directory to find the correct path to UCF50
!ls -l

# Check the extracted contents to ensure the UCF50 directory exists
!ls -l UCF50

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

```
total 3157784
drwxr-xr-x  1 root root       4096 Jun 21 13:28 sample_data
drwxr-xr-x 52 root root       4096 Oct  4  2010 UCF50
-rw-r--r--  1 root root 3233554570 Dec  1  2011 UCF50.rar
total 456
drwxr-xr-x 2 root root 12288 Oct  1  2010 BaseballPitch
drwxr-xr-x 2 root root 12288 Oct  5  2010 Basketball
drwxr-xr-x 2 root root 12288 Oct  1  2010 BenchPress
drwxr-xr-x 2 root root  4096 Oct  1  2010 Biking
drwxr-xr-x 2 root root 12288 Oct  1  2010 Billiards
drwxr-xr-x 2 root root  4096 Oct  5  2010 BreastStroke
drwxr-xr-x 2 root root  4096 Oct  1  2010 CleanAndJerk
drwxr-xr-x 2 root root 12288 Oct  5  2010 Diving
drwxr-xr-x 2 root root 12288 Oct 13  2010 Drumming
drwxr-xr-x 2 root root  4096 Oct  1  2010 Fencing
drwxr-xr-x 2 root root 12288 Oct  1  2010 GolfSwing
drwxr-xr-x 2 root root  4096 Oct 13  2010 HighJump
drwxr-xr-x 2 root root 12288 Oct 13  2010 HorseRace
drwxr-xr-x 2 root root 12288 Oct  5  2010 HorseRiding
drwxr-xr-x 2 root root  4096 Oct  1  2010 HulaHoop
drwxr-xr-x 2 root root 12288 Oct  1  2010 JavelinThrow
drwxr-xr-x 2 root root 12288 Oct  1  2010 JugglingBalls
drwxr-xr-x 2 root root 12288 Oct 13  2010 JumpingJack
drwxr-xr-x 2 root root 12288 Oct  1  2010 JumpRope
drwxr-xr-x 2 root root 12288 Oct  1  2010 Kayaking
drwxr-xr-x 2 root root  4096 Oct  1  2010 Lunges
drwxr-xr-x 2 root root 12288 Oct  1  2010 MilitaryParade
drwxr-xr-x 2 root root  4096 Oct  1  2010 Mixing
drwxr-xr-x 2 root root 12288 Oct  1  2010 Nunchucks
drwxr-xr-x 2 root root 12288 Oct  1  2010 PizzaTossing
drwxr-xr-x 2 root root 12288 Oct  1  2010 PlayingGuitar
drwxr-xr-x 2 root root  4096 Oct  1  2010 PlayingPiano
drwxr-xr-x 2 root root 12288 Oct 13  2010 PlayingTabla
drwxr-xr-x 2 root root  4096 Oct  1  2010 PlayingViolin
drwxr-xr-x 2 root root 12288 Oct 13  2010 PoleVault
drwxr-xr-x 2 root root 12288 Oct 13  2010 PommelHorse
drwxr-xr-x 2 root root  4096 Oct  1  2010 PullUps
drwxr-xr-x 2 root root 12288 Oct  1  2010 Punch
drwxr-xr-x 2 root root  4096 Oct 13  2010 PushUps
drwxr-xr-x 2 root root 12288 Oct  1  2010 RockClimbingIndoor
drwxr-xr-x 2 root root 12288 Oct  1  2010 RopeClimbing
drwxr-xr-x 2 root root  4096 Oct  1  2010 Rowing
drwxr-xr-x 2 root root 12288 Oct  1  2010 SalsaSpin
drwxr-xr-x 2 root root 12288 Oct  1  2010 SkateBoarding
drwxr-xr-x 2 root root  4096 Oct  1  2010 Skiing
drwxr-xr-x 2 root root  4096 Oct  1  2010 Skijet
drwxr-xr-x 2 root root 12288 Oct  1  2010 SoccerJuggling
drwxr-xr-x 2 root root  4096 Oct  1  2010 Swing
drwxr-xr-x 2 root root  4096 Oct  1  2010 TaiChi
drwxr-xr-x 2 root root 12288 Oct  1  2010 TennisSwing
drwxr-xr-x 2 root root 12288 Oct  1  2010 ThrowDiscus
drwxr-xr-x 2 root root 12288 Oct  4  2010 TrampolineJumping
drwxr-xr-x 2 root root 12288 Oct  1  2010 VolleyballSpiking
drwxr-xr-x 2 root root 12288 Oct  1  2010 WalkingWithDog
drwxr-xr-x 2 root root  4096 Oct  1  2010 YoYo
['Rowing', 'TaiChi', 'Swing', 'WalkingWithDog', 'CleanAndJerk', 'YoYo', 'TennisSwing', 'PlayingGuitar', 'HorseRace', 'Pl
<Figure size 1000x1000 with 0 Axes>
```

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


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)
print(random_range)

# Loop through the random values.
```

```
[41, 30, 44, 17, 18, 12, 4, 43, 16, 34, 21, 49, 23, 25, 11, 15, 40, 31, 42, 37]
<Figure size 2000x2000 with 0 Axes>
```

```
# 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.
    ret, bgr_frame = video_reader.read()

    # Release the VideoCapture object. (Since we read only a single frame)
    video_reader.release()

    # Check if frame is read correctly
    if not ret:
        print(f"Error: Failed to read frame from video {selected_video_file_name} in class {selected_class_name}")
        continue

    # 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')

plt.show()
```



DATA PREPROCESSING

```
# 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 #for LSTM network, greater the sequence more time it takes to process ie., we sample 20 frames out of e

# 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"]
```

Function to extract,resize and normalise 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 FUNCTION FOR DATASET CREATION

```
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.
    '''
    DATASET_DIR='UCF50'
    # 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
```

```
#converting labels 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)


# 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)
```

- To construct the model, we will use Keras **ConvLSTM2D** recurrent layers. The ConvLSTM2D layer also takes in the number of filters and kernel size required for applying the convolutional operations
- The output of the layers is flattened in the end and is fed to the Dense layer with softmax activation which outputs the probability of each action category.
- use MaxPooling3D layers to reduce the dimensions of the frames and avoid unnecessary computations and Dropout layers to prevent overfitting the model on the data.
- The architecture is a simple one and has a small number of trainable parameters. This is because we are only dealing with a small subset of the dataset which does not require a large-scale model.

1. Works with 3D data and is a special type of LSTM network

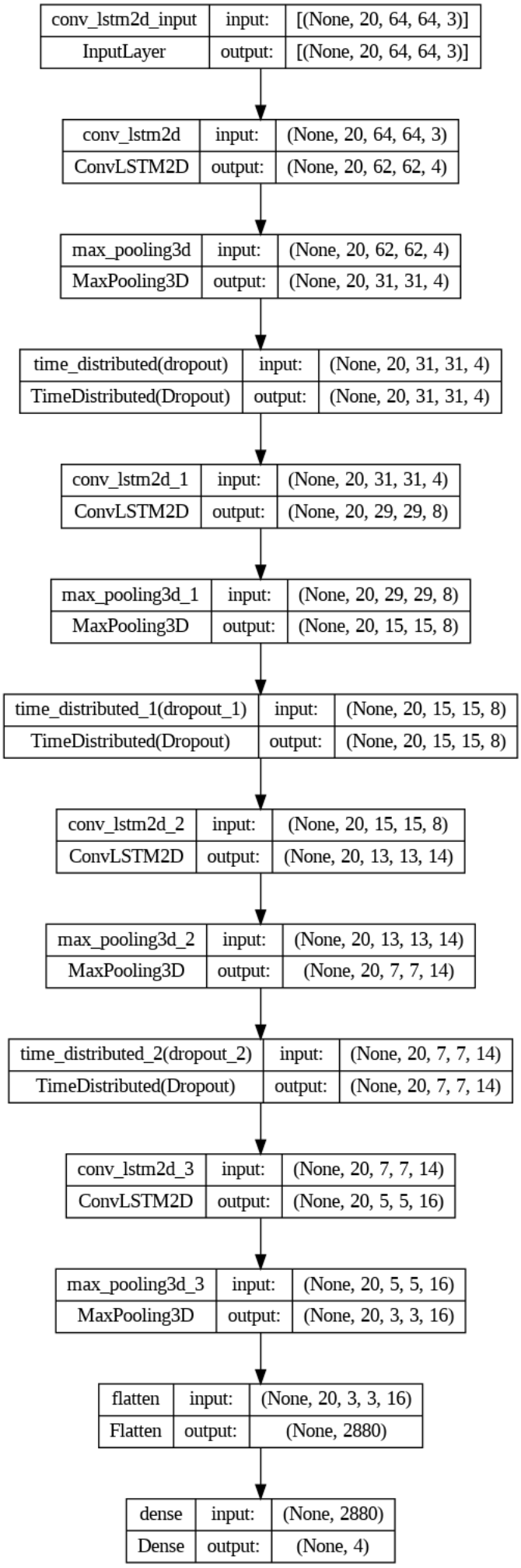
2. Has convolutional layers inside it.

```
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))) #regularization technique->randomly ignore 20% and needs to be applied to all t

    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()) #combine the featuremaps and flatten them

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

    #We use pooling layers after every convolutional layer and help us to reduce the size and learn more about the network
    ########################################################################################################################

    # Display the models summary.
    model.summary()

    # Return the constructed convlstm model.
    return model


#model creation
# 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 (MaxPooling3  (None, 20, 31, 31, 4)     0
 D)

 time_distributed (TimeDist  (None, 20, 31, 31, 4)     0
 ributed)

 conv_lstm2d_1 (ConvLSTM2D)  (None, 20, 29, 29, 8)     3488

 max_pooling3d_1 (MaxPoolin  (None, 20, 15, 15, 8)     0
 g3D)

 time_distributed_1 (TimeDi  (None, 20, 15, 15, 8)     0
 stributed)

 conv_lstm2d_2 (ConvLSTM2D)  (None, 20, 13, 13, 14)    11144

 max_pooling3d_2 (MaxPoolin  (None, 20, 7, 7, 14)      0
```

```
 g3D)

 time_distributed_2 (TimeDi  (None, 20, 7, 7, 14)      0
 stributed)

 conv_lstm2d_3 (ConvLSTM2D)  (None, 20, 5, 5, 16)      17344

 max_pooling3d_3 (MaxPoolin  (None, 20, 3, 3, 16)      0
 g3D)

 flatten (Flatten)           (None, 2880)              0

 dense (Dense)               (None, 4)                 11524

=================================================================
Total params: 44524 (173.92 KB)
Trainable params: 44524 (173.92 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Model Created Successfully!
```

```
# 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)
max_pooling3d_2 | input: | (None, 20, 13, 13, 14)
MaxPooling3D | output: | (None, 20, 7, 7, 14)
time_distributed_2(dropout_2) | input: | (None, 20, 7, 7, 14)
TimeDistributed(Dropout) | output: | (None, 20, 7, 7, 14)
conv_lstm2d_3 | input: | (None, 20, 7, 7, 14)
ConvLSTM2D | output: | (None, 20, 5, 5, 16)
max_pooling3d_3 | input: | (None, 20, 5, 5, 16)
MaxPooling3D | output: | (None, 20, 3, 3, 16)
flatten | input: | (None, 20, 3, 3, 16)
Flatten | output: | (None, 2880)
dense | input: | (None, 2880)
Dense | output: | (None, 4)

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

# 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]) #20% of training data is used fo
```

```
Epoch 1/50
73/73 [==============================] - 130s 2s/step - loss: 1.3710 - accuracy: 0.3493 - val_loss: 1.3394 - val_accurac
Epoch 2/50
73/73 [==============================] - 124s 2s/step - loss: 1.3289 - accuracy: 0.4041 - val_loss: 1.2545 - val_accurac
Epoch 3/50
73/73 [==============================] - 125s 2s/step - loss: 1.1983 - accuracy: 0.4521 - val_loss: 1.0143 - val_accurac
Epoch 4/50
73/73 [==============================] - 120s 2s/step - loss: 0.8268 - accuracy: 0.6267 - val_loss: 0.8371 - val_accurac
Epoch 5/50
73/73 [==============================] - 124s 2s/step - loss: 0.6638 - accuracy: 0.7192 - val_loss: 0.7450 - val_accurac
Epoch 6/50
73/73 [==============================] - 124s 2s/step - loss: 0.4968 - accuracy: 0.7705 - val_loss: 0.6039 - val_accurac
Epoch 7/50
73/73 [==============================] - 123s 2s/step - loss: 0.4214 - accuracy: 0.8356 - val_loss: 0.4538 - val_accurac
Epoch 8/50
73/73 [==============================] - 126s 2s/step - loss: 0.3782 - accuracy: 0.8253 - val_loss: 0.5341 - val_accurac
Epoch 9/50
73/73 [==============================] - 123s 2s/step - loss: 0.2552 - accuracy: 0.8904 - val_loss: 0.5409 - val_accurac
Epoch 10/50
73/73 [==============================] - 122s 2s/step - loss: 0.2762 - accuracy: 0.8767 - val_loss: 0.4564 - val_accurac
Epoch 11/50
73/73 [==============================] - 120s 2s/step - loss: 0.1475 - accuracy: 0.9486 - val_loss: 0.5230 - val_accurac
Epoch 12/50
73/73 [==============================] - 121s 2s/step - loss: 0.1189 - accuracy: 0.9589 - val_loss: 0.5514 - val_accurac
Epoch 13/50
73/73 [==============================] - 122s 2s/step - loss: 0.1590 - accuracy: 0.9384 - val_loss: 0.9964 - val_accurac
Epoch 14/50
73/73 [==============================] - 115s 2s/step - loss: 0.1347 - accuracy: 0.9418 - val_loss: 0.4301 - val_accurac
Epoch 15/50
73/73 [==============================] - 118s 2s/step - loss: 0.0821 - accuracy: 0.9726 - val_loss: 0.6823 - val_accurac
Epoch 16/50
73/73 [==============================] - 118s 2s/step - loss: 0.0783 - accuracy: 0.9692 - val_loss: 0.7253 - val_accurac
Epoch 17/50
73/73 [==============================] - 118s 2s/step - loss: 0.0808 - accuracy: 0.9692 - val_loss: 0.5205 - val_accurac
Epoch 18/50
73/73 [==============================] - 115s 2s/step - loss: 0.0348 - accuracy: 0.9829 - val_loss: 0.7106 - val_accurac
Epoch 19/50
73/73 [==============================] - 113s 2s/step - loss: 0.0515 - accuracy: 0.9829 - val_loss: 0.5851 - val_accurac
Epoch 20/50
73/73 [==============================] - 118s 2s/step - loss: 0.0243 - accuracy: 0.9932 - val_loss: 0.6206 - val_accurac
Epoch 21/50
73/73 [==============================] - 117s 2s/step - loss: 0.0694 - accuracy: 0.9760 - val_loss: 0.8572 - val_accurac
Epoch 22/50
73/73 [==============================] - 119s 2s/step - loss: 0.0864 - accuracy: 0.9692 - val_loss: 0.5608 - val_accurac
Epoch 23/50
73/73 [==============================] - 116s 2s/step - loss: 0.0109 - accuracy: 1.0000 - val_loss: 0.5352 - val_accurac
Epoch 24/50
73/73 [==============================] - 117s 2s/step - loss: 0.0200 - accuracy: 0.9932 - val_loss: 0.7337 - val_accurac
```

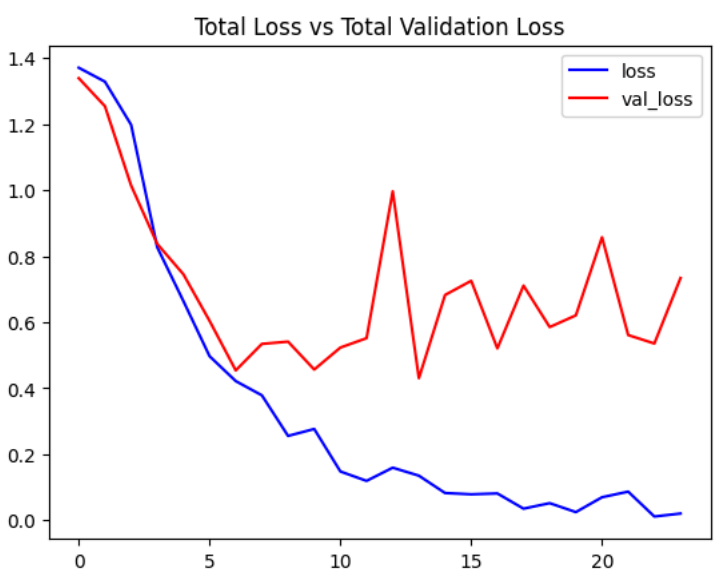
```
# Evaluate the trained model.
model_evaluation_history = convlstm_model.evaluate(features_test, labels_test)
```

```
4/4 [==============================] - 11s 2s/step - loss: 0.5279 - accuracy: 0.8443
```

```
#SAVE 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_e

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

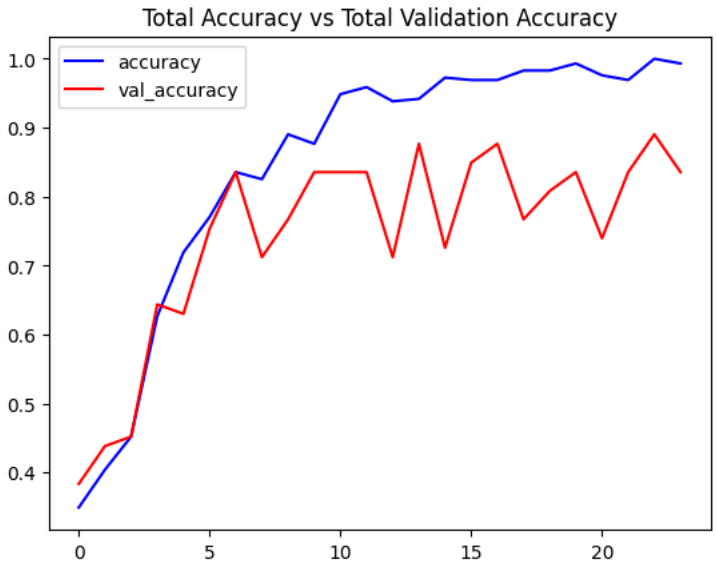
```
WARNING:py.warnings:/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are savi
  saving_api.save_model(
```

```
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()

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



There is some overfitting in loss but the accuracy is comparable

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



CNN+LSTM APPROACH

Train a Convolution and LSTM layers together

```
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-->we are wrapping up each of the layers in a time distributed layer and the reason for
        we have a number of sequential frames and we need all actions to be applied on the whole sequence'''
    ##################################################################################################################

    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 (TimeDi  (None, 20, 64, 64, 16)    448
 stributed)

 time_distributed_4 (TimeDi  (None, 20, 16, 16, 16)    0
 stributed)

 time_distributed_5 (TimeDi  (None, 20, 16, 16, 16)    0
 stributed)

 time_distributed_6 (TimeDi  (None, 20, 16, 16, 32)    4640
 stributed)

 time_distributed_7 (TimeDi  (None, 20, 4, 4, 32)      0
 stributed)

 time_distributed_8 (TimeDi  (None, 20, 4, 4, 32)      0
 stributed)

 time_distributed_9 (TimeDi  (None, 20, 4, 4, 64)      18496
 stributed)

 time_distributed_10 (TimeD  (None, 20, 2, 2, 64)      0
 istributed)

 time_distributed_11 (TimeD  (None, 20, 2, 2, 64)      0
 istributed)

 time_distributed_12 (TimeD  (None, 20, 2, 2, 64)      36928
 istributed)

 time_distributed_13 (TimeD  (None, 20, 1, 1, 64)      0
 istributed)

 time_distributed_14 (TimeD  (None, 20, 64)            0
 istributed)

 lstm (LSTM)                 (None, 32)                12416

 dense_1 (Dense)             (None, 4)                 132

=================================================================
Total params: 73060 (285.39 KB)
Trainable params: 73060 (285.39 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
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)
```