Outline

- `Step 1:` **Visualize the Data with its Labels**
- `Step 2:` **Preprocess the Dataset**
- `Step 3:` **Split the Data into Train and Test Set**
- `Step 4:` **Implement the ConvLSTM Approach**
  - `Step 4.1:` **Construct the Model**
  - `Step 4.2:` **Compile & Train the Model**
  - `Step 4.3:` **Plot Model's Loss & Accuracy Curves**
- `Step 5:` **implement the LRCN Approach**
  - `Step 5.1:` **Construct the Model**
  - `Step 5.2:` **Compile & Train the Model**
  - `Step 5.3:` **Plot Model's Loss & Accuracy Curves**
- `Step 6:` **Test the Best Performing Model on YouTube videos**

Alright, so without further ado, let's get started.

Import the Libraries

We will start by installing and importing the required libraries.

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

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

```
Requirement already satisfied: pydot in /usr/local/lib/python3.10/dist-packages (1.4.2)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.3.3)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.54.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: jax>=0.3.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.8)
Requirement already satisfied: keras<2.13,>=2.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.12.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.0)
Requirement already satisfied: numpy<1.24,>=1.22 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.22.4)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/pyth
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: tensorboard<2.13,>=2.12 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.12.2)
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.12.0
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.5.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.3
Requirement already satisfied: decorator<5.0,>=4.0.2 in /usr/local/lib/python3.10/dist-packages (from moviepy) (4.4.2)
Requirement already satisfied: tqdm<5.0,>=4.11.2 in /usr/local/lib/python3.10/dist-packages (from moviepy) (4.65.0)
Requirement already satisfied: requests<3.0,>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from moviepy) (2.27.1)
Requirement already satisfied: proglog<=1.0.0 in /usr/local/lib/python3.10/dist-packages (from moviepy) (0.1.10)
Requirement already satisfied: imageio<3.0,>=2.5 in /usr/local/lib/python3.10/dist-packages (from moviepy) (2.25.1)
Requirement already satisfied: imageio-ffmpeg>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from moviepy) (0.4.8)
Requirement already satisfied: pyparsing>=2.1.4 in /usr/local/lib/python3.10/dist-packages (from pydot) (3.0.9)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0.4
Requirement already satisfied: pillow>=8.3.2 in /usr/local/lib/python3.10/dist-packages (from imageio<3.0,>=2.5->moviepy) (8.4.0)
Requirement already satisfied: ml-dtypes>=0.0.3 in /usr/local/lib/python3.10/dist-packages (from jax>=0.3.15->tensorflow) (0.1.0)
Requirement already satisfied: scipy>=1.7 in /usr/local/lib/python3.10/dist-packages (from jax>=0.3.15->tensorflow) (1.10.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0,>=2.8.1->moviepy)
```

```
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.1
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorbo
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboa
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.13
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<1.1,>=0
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.13,>=
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-aut
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth
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-vkz61fmd/pafy_2f0fb44e9f13415ab363f1c7b214d483
  Running command git clone --filter=blob:none --quiet https://github.com/TahaAnwar/pafy.git /tmp/pip-install-vkz61fmd/pafy_2f0fb44e9
  Resolved https://github.com/TahaAnwar/pafy.git to commit 2f3c473b3df7961721d07e1504675313afd1d2cb
```

```
# 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)
```

```
from google.colab import drive
import pathlib
drive.mount('/content/drive')

#dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"

#data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url, untar=True)

data_dir = pathlib.Path('/content/drive/MyDrive/Explosion')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

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

```
--2023-05-11 12:26:55--  https://www.crcv.ucf.edu/data/UCF50.rar
Resolving www.crcv.ucf.edu (www.crcv.ucf.edu)... 132.170.214.127
Connecting to www.crcv.ucf.edu (www.crcv.ucf.edu)|132.170.214.127|:443... connected.
WARNING: cannot verify www.crcv.ucf.edu's certificate, issued by ‘CN=InCommon RSA Server CA,OU=InCommon,O=Internet2,L=Ann Arbor,ST=MI,C=
  Unable to locally verify the issuer's authority.
HTTP request sent, awaiting response... 200 OK
Length: 3233554570 (3.0G) [application/rar]
Saving to: ‘UCF50.rar.2’

UCF50.rar.2          29%[====>               ] 917.12M  73.7MB/s    eta 30s    ^C

UNRAR 5.61 beta 1 freeware      Copyright (c) 1993-2018 Alexander Roshal
```

```
Extracting from UCF50.rar


Would you like to replace the existing file UCF50/BaseballPitch/v_BaseballPitch_g01_c01.avi
318098 bytes, modified on 2010-10-01 15:49
with a new one
318098 bytes, modified on 2010-10-01 15:49

[Y]es, [N]o, [A]ll, n[E]ver, [R]ename, [Q]uit
User break

User break
```

Step 1: Visualize the Data with its Labels

In the first step, we will 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

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('/content/drive/MyDrive/Explosion')
print(all_classes_names)

# 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)),2 )

# 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'/content/drive/MyDrive/Explosion/{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'/content/drive/MyDrive/Explosion/{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')
```

```
['abnormal', 'normal']
```

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 = 10

# Specify the directory containing the UCF50 dataset.
DATASET_DIR = "/content/drive/MyDrive/Explosion"

# Specify the list containing the names of the classes used for training. Feel free to choose any set of classes.
CLASSES_LIST = all_classes_names
print(CLASSES_LIST)
```

```
['abnormal', 'normal']
```

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.
    #print(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))
        print(files_list)

        # Iterate through all the files present in the files list.
        for file_name in files_list:
                #print(len(file_name))
                #print(vn)

                # Get the complete video path.
                #print(sn)
                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)
                #print(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.
    #print(video_files_paths)
    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: abnormal
['Explosion011_x264_30.mp4', 'Explosion009_x264_7.mp4', 'Explosion009_x264_31.mp4', 'Explosion009_x264_8.mp4', 'Explosion011_x264_28.mp4
Extracting Data of Class: normal
['Explosion006_x264_17.mp4', 'Explosion006_x264_10.mp4', 'Explosion006_x264_16.mp4', 'Explosion006_x264_15.mp4', 'Explosion004_x264_12.m
```

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)
```

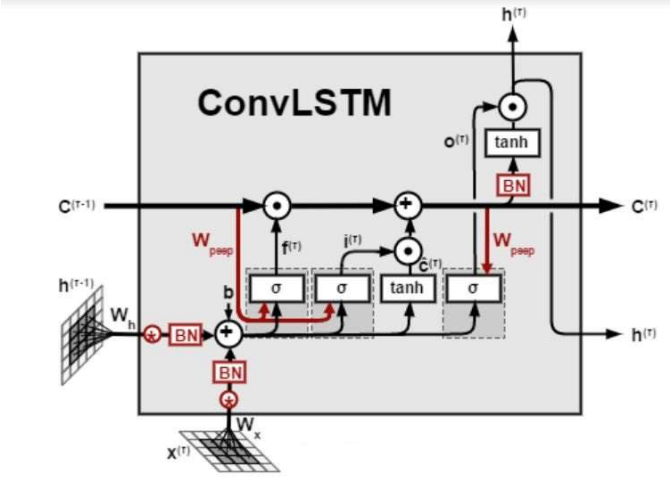
Step 3: Split the Data into Train and Test Set

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)
```

Step 4: Implement the ConvLSTM Approach

In this step, we will implement the first approach by using a combination of ConvLSTM cells. A ConvLSTM cell is a variant of an LSTM network that contains convolutions operations in the network. it is an LSTM with convolution embedded in the architecture, which makes it capable of identifying spatial features of the data while keeping into account the temporal relation.



For video classification, this approach effectively captures the spatial relation in the individual frames and the temporal relation across the different frames. As a result of this convolution structure, the ConvLSTM is capable of taking in 3-dimensional input `(width, height, num_of_channels)` whereas a simple LSTM only takes in 1-dimensional input hence an LSTM is incompatible for modeling Spatio-temporal data on its own.

You can read the paper **Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting** by **Xingjian Shi** (NIPS 2015), to learn more about this architecture.

Step 4.1: Construct the Model

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.

We will also 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.

```
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.

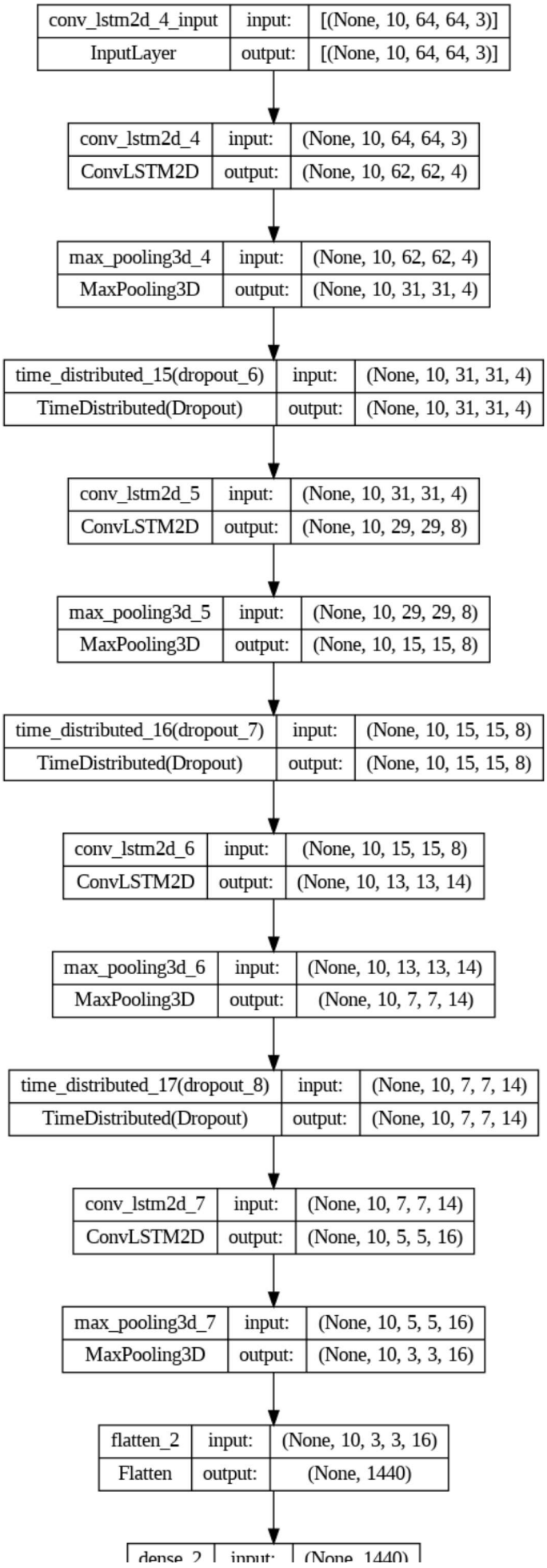
```
# Construct the required convlstm model.
convlstm_model = create_convlstm_model()

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

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm2d_4 (ConvLSTM2D)  (None, 10, 62, 62, 4)     1024

 max_pooling3d_4 (MaxPooling  (None, 10, 31, 31, 4)    0
 3D)

 time_distributed_15 (TimeDi  (None, 10, 31, 31, 4)    0
 stributed)

 conv_lstm2d_5 (ConvLSTM2D)  (None, 10, 29, 29, 8)     3488

 max_pooling3d_5 (MaxPooling  (None, 10, 15, 15, 8)    0
 3D)

 time_distributed_16 (TimeDi  (None, 10, 15, 15, 8)    0
 stributed)

 conv_lstm2d_6 (ConvLSTM2D)  (None, 10, 13, 13, 14)    11144

 max_pooling3d_6 (MaxPooling  (None, 10, 7, 7, 14)     0
 3D)

 time_distributed_17 (TimeDi  (None, 10, 7, 7, 14)     0
 stributed)

 conv_lstm2d_7 (ConvLSTM2D)  (None, 10, 5, 5, 16)      17344

 max_pooling3d_7 (MaxPooling  (None, 10, 3, 3, 16)     0
 3D)

 flatten_2 (Flatten)         (None, 1440)              0

 dense_2 (Dense)             (None, 2)                 2882

=================================================================
Total params: 35,882
Trainable params: 35,882
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_4_input | input: | [(None, 10, 64, 64, 3)]
InputLayer | output: | [(None, 10, 64, 64, 3)]
conv_lstm2d_4 | input: | (None, 10, 64, 64, 3)
ConvLSTM2D | output: | (None, 10, 62, 62, 4)
max_pooling3d_4 | input: | (None, 10, 62, 62, 4)
MaxPooling3D | output: | (None, 10, 31, 31, 4)
time_distributed_15(dropout_6) | input: | (None, 10, 31, 31, 4)
TimeDistributed(Dropout) | output: | (None, 10, 31, 31, 4)
conv_lstm2d_5 | input: | (None, 10, 31, 31, 4)
ConvLSTM2D | output: | (None, 10, 29, 29, 8)
max_pooling3d_5 | input: | (None, 10, 29, 29, 8)
MaxPooling3D | output: | (None, 10, 15, 15, 8)
time_distributed_16(dropout_7) | input: | (None, 10, 15, 15, 8)
TimeDistributed(Dropout) | output: | (None, 10, 15, 15, 8)
conv_lstm2d_6 | input: | (None, 10, 15, 15, 8)
ConvLSTM2D | output: | (None, 10, 13, 13, 14)
max_pooling3d_6 | input: | (None, 10, 13, 13, 14)
MaxPooling3D | output: | (None, 10, 7, 7, 14)
time_distributed_17(dropout_8) | input: | (None, 10, 7, 7, 14)
TimeDistributed(Dropout) | output: | (None, 10, 7, 7, 14)
conv_lstm2d_7 | input: | (None, 10, 7, 7, 14)
ConvLSTM2D | output: | (None, 10, 5, 5, 16)
max_pooling3d_7 | input: | (None, 10, 5, 5, 16)
MaxPooling3D | output: | (None, 10, 3, 3, 16)
flatten_2 | input: | (None, 10, 3, 3, 16)
Flatten | output: | (None, 1440)
dense_2 | input: | (None, 1440)

Step 4.2: Compile & Train the Model

Next, we will add an early stopping callback to prevent overfitting and start the training after compiling the model.

```
# 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 4/50
111/111 [==============================] - 33s 294ms/step - loss: 0.4935 - accuracy: 0.7846 - val_loss: 0.8199 - val_accuracy: 0.7477
Epoch 5/50
111/111 [==============================] - 31s 276ms/step - loss: 0.4954 - accuracy: 0.7959 - val_loss: 0.4834 - val_accuracy: 0.8018
Epoch 6/50
111/111 [==============================] - 31s 276ms/step - loss: 0.4723 - accuracy: 0.7937 - val_loss: 0.4868 - val_accuracy: 0.8018
Epoch 7/50
111/111 [==============================] - 32s 292ms/step - loss: 0.4480 - accuracy: 0.8186 - val_loss: 0.4297 - val_accuracy: 0.8649
Epoch 8/50
111/111 [==============================] - 32s 283ms/step - loss: 0.4207 - accuracy: 0.8503 - val_loss: 0.4277 - val_accuracy: 0.8198
Epoch 9/50
111/111 [==============================] - 31s 279ms/step - loss: 0.3728 - accuracy: 0.8617 - val_loss: 0.3956 - val_accuracy: 0.8739
Epoch 10/50
111/111 [==============================] - 31s 284ms/step - loss: 0.3379 - accuracy: 0.8730 - val_loss: 0.4248 - val_accuracy: 0.8288
Epoch 11/50
111/111 [==============================] - 32s 286ms/step - loss: 0.3497 - accuracy: 0.8571 - val_loss: 0.3645 - val_accuracy: 0.8829
Epoch 12/50
111/111 [==============================] - 31s 277ms/step - loss: 0.3456 - accuracy: 0.8594 - val_loss: 0.3576 - val_accuracy: 0.8739
Epoch 13/50
111/111 [==============================] - 31s 280ms/step - loss: 0.2755 - accuracy: 0.8980 - val_loss: 0.4725 - val_accuracy: 0.8108
Epoch 14/50
111/111 [==============================] - 33s 296ms/step - loss: 0.2769 - accuracy: 0.8844 - val_loss: 0.3877 - val_accuracy: 0.8739
Epoch 15/50
111/111 [==============================] - 31s 279ms/step - loss: 0.2521 - accuracy: 0.9070 - val_loss: 0.3759 - val_accuracy: 0.8559
Epoch 16/50
111/111 [==============================] - 31s 276ms/step - loss: 0.2459 - accuracy: 0.9002 - val_loss: 0.3608 - val_accuracy: 0.8559
Epoch 17/50
111/111 [==============================] - 31s 282ms/step - loss: 0.2373 - accuracy: 0.9025 - val_loss: 0.3860 - val_accuracy: 0.8649
Epoch 18/50
111/111 [==============================] - 33s 292ms/step - loss: 0.2278 - accuracy: 0.9138 - val_loss: 0.4265 - val_accuracy: 0.8378
Epoch 19/50
111/111 [==============================] - 31s 276ms/step - loss: 0.2214 - accuracy: 0.9138 - val_loss: 0.2912 - val_accuracy: 0.8919
Epoch 20/50
111/111 [==============================] - 31s 279ms/step - loss: 0.2045 - accuracy: 0.9206 - val_loss: 0.3490 - val_accuracy: 0.8559
Epoch 21/50
111/111 [==============================] - 33s 294ms/step - loss: 0.2564 - accuracy: 0.9070 - val_loss: 0.3581 - val_accuracy: 0.8649
Epoch 22/50
111/111 [==============================] - 31s 278ms/step - loss: 0.1904 - accuracy: 0.9252 - val_loss: 0.2822 - val_accuracy: 0.8919
Epoch 23/50
111/111 [==============================] - 31s 278ms/step - loss: 0.1767 - accuracy: 0.9252 - val_loss: 0.3170 - val_accuracy: 0.9099
Epoch 24/50
111/111 [==============================] - 31s 283ms/step - loss: 0.1865 - accuracy: 0.9410 - val_loss: 0.3122 - val_accuracy: 0.8919
Epoch 25/50
111/111 [==============================] - 32s 285ms/step - loss: 0.1599 - accuracy: 0.9320 - val_loss: 0.3583 - val_accuracy: 0.8739
Epoch 26/50
111/111 [==============================] - 31s 278ms/step - loss: 0.2312 - accuracy: 0.9138 - val_loss: 0.3253 - val_accuracy: 0.8649
Epoch 27/50
111/111 [==============================] - 31s 280ms/step - loss: 0.1898 - accuracy: 0.9320 - val_loss: 0.3301 - val_accuracy: 0.8919
Epoch 28/50
111/111 [==============================] - 33s 295ms/step - loss: 0.1924 - accuracy: 0.9161 - val_loss: 0.3349 - val_accuracy: 0.8739
Epoch 29/50
111/111 [==============================] - 31s 276ms/step - loss: 0.1595 - accuracy: 0.9456 - val_loss: 0.3466 - val_accuracy: 0.9009
Epoch 30/50
111/111 [==============================] - 31s 279ms/step - loss: 0.1380 - accuracy: 0.9433 - val_loss: 0.3562 - val_accuracy: 0.8829
Epoch 31/50
111/111 [==============================] - 31s 282ms/step - loss: 0.1325 - accuracy: 0.9592 - val_loss: 0.4121 - val_accuracy: 0.8919
Epoch 32/50
111/111 [==============================] - 32s 289ms/step - loss: 0.1589 - accuracy: 0.9433 - val_loss: 0.3046 - val_accuracy: 0.9099
```

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)
```

```
6/6 [==============================] - 0s 66ms/step - loss: 0.3837 - accuracy: 0.8859
```

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_accurac

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

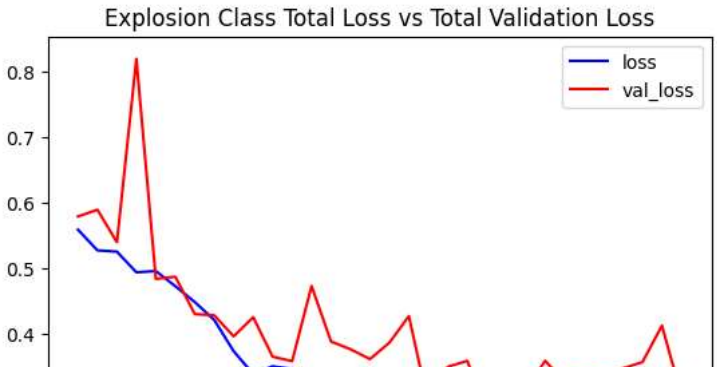
Step 4.3: Plot Model's Loss & Accuracy Curves

Now we will create a function **`plot_metric()`** to visualize the training and validation metrics. We already have separate metrics from our training and validation steps so now we just have to visualize them.

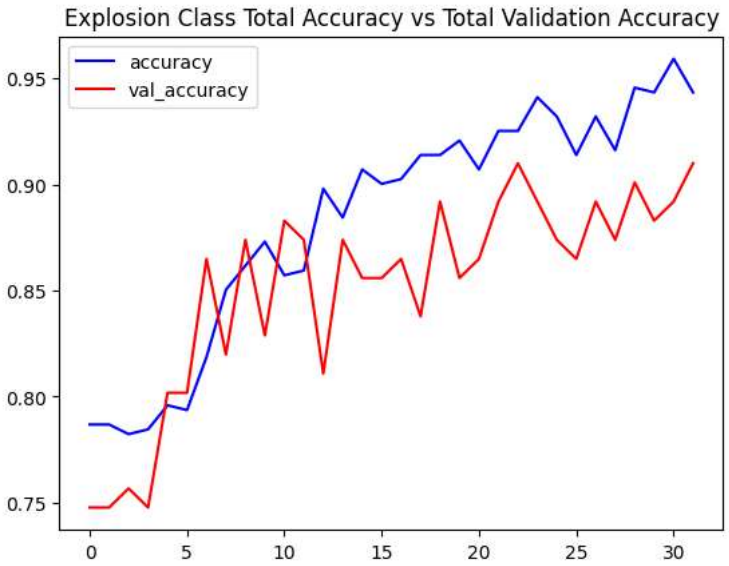
```
def plot_metric(model_training_history, metric_name_1, metric_name_2, plot_name):
    '''
    This function will plot the metrics passed to it in a graph.
    Args:
        model_training_history: A history object containing a record of training and validation
                                loss 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', 'Explosion Class Total Loss vs Total Validation Loss')
```



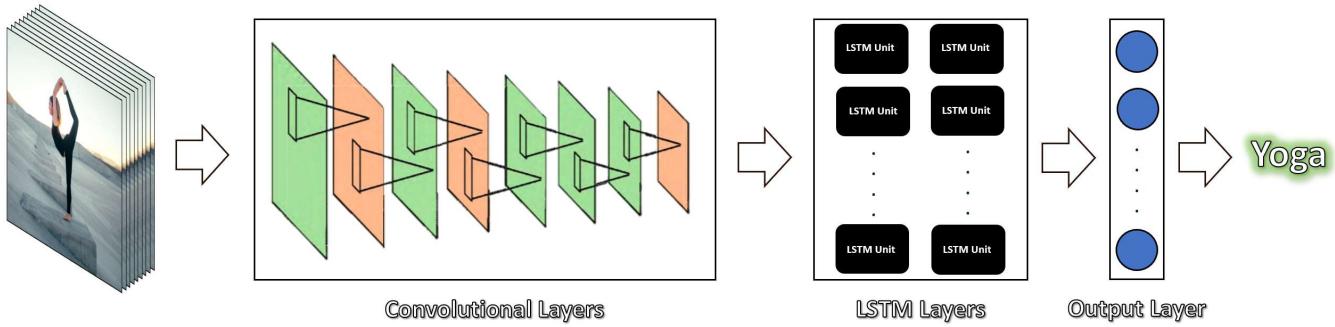

```
# Visualize the training and validation accuracy metrices.
plot_metric(convlstm_model_training_history, 'accuracy', 'val_accuracy', 'Explosion Class Total Accuracy vs Total Validation Accuracy')
```



Step 5: Implement the LRCN Approach

In this step, we will implement the LRCN Approach by combining Convolution and LSTM layers in a single model. Another similar approach can be to use a CNN model and LSTM model trained separately. The CNN model can be used to extract spatial features from the frames in the video, and for this purpose, a pre-trained model can be used, that can be fine-tuned for the problem. And the LSTM model can then use the features extracted by CNN, to predict the action being performed in the video.

But here, we will implement another approach known as the Long-term Recurrent Convolutional Network (LRCN), which combines CNN and LSTM layers in a single model. The Convolutional layers are used for spatial feature extraction from the frames, and the extracted spatial features are fed to LSTM layer(s) at each time-steps for temporal sequence modeling. This way the network learns spatiotemporal features directly in an end-to-end training, resulting in a robust model.



You can read the paper Long-term Recurrent Convolutional Networks for Visual Recognition and Description by Jeff Donahue (CVPR 2015), to learn more about this architecture.

We will also use `TimeDistributed` wrapper layer, which allows applying the same layer to every frame of the video independently. So it makes a layer (around which it is wrapped) capable of taking input of shape `(no_of_frames, width, height, num_of_channels)` if originally the layer's input shape was `(width, height, num_of_channels)` which is very beneficial as it allows to input the whole video into the model in a single shot.

Step 5.1: Construct the Model

To implement our LRCN architecture, we will use time-distributed **`Conv2D`** layers which will be followed by **`MaxPooling2D`** and **`Dropout`** layers. The feature extracted from the **`Conv2D`** layers will be then flattened using the **`Flatten`** layer and will be fed to a **`LSTM`** layer. The **`Dense`** layer with softmax activation will then use the output from the **`LSTM`** layer to predict the action being performed.

```
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.

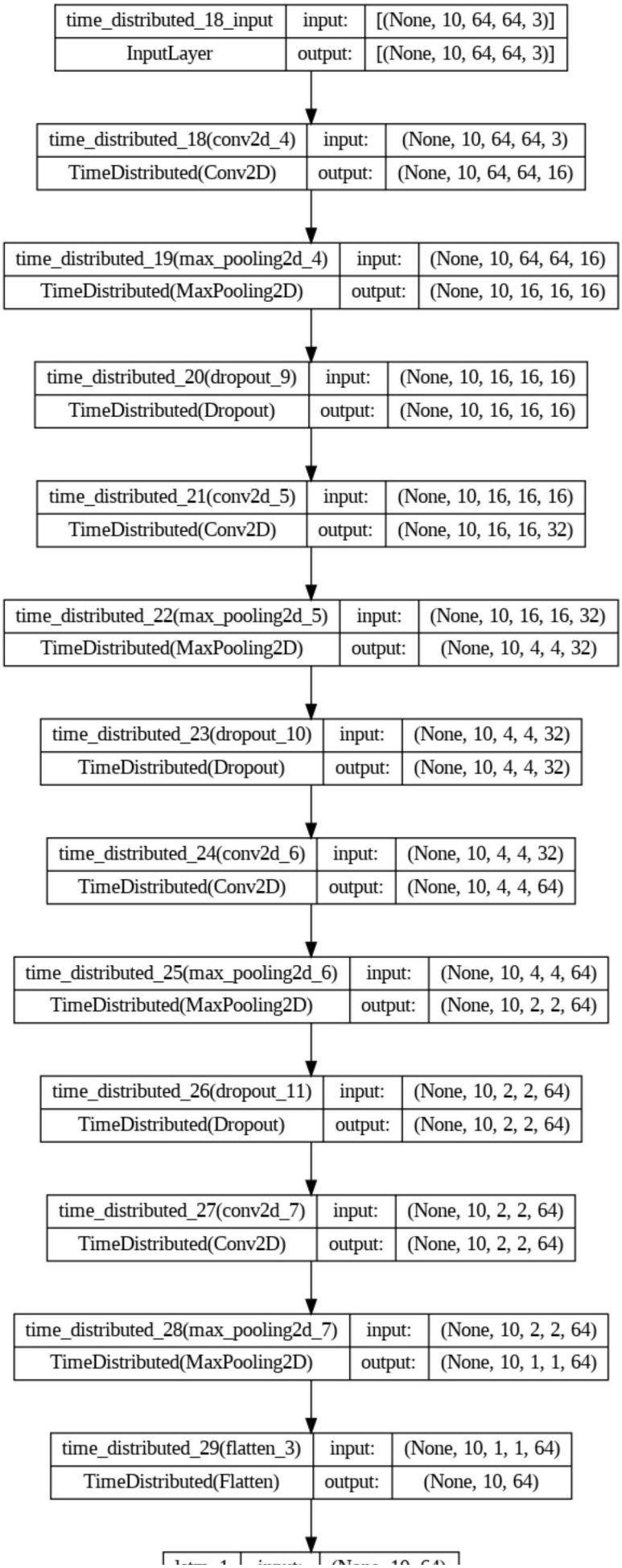
```
# Construct the required LRCN model.
LRCN_model = create_LRCN_model()

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

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed_18 (TimeDi  (None, 10, 64, 64, 16)   448
 stributed)

 time_distributed_19 (TimeDi  (None, 10, 16, 16, 16)   0
 stributed)

 time_distributed_20 (TimeDi  (None, 10, 16, 16, 16)   0
 stributed)

 time_distributed_21 (TimeDi  (None, 10, 16, 16, 32)   4640
 stributed)

 time_distributed_22 (TimeDi  (None, 10, 4, 4, 32)     0
 stributed)

 time_distributed_23 (TimeDi  (None, 10, 4, 4, 32)     0
 stributed)

 time_distributed_24 (TimeDi  (None, 10, 4, 4, 64)     18496
 stributed)

 time_distributed_25 (TimeDi  (None, 10, 2, 2, 64)     0
 stributed)

 time_distributed_26 (TimeDi  (None, 10, 2, 2, 64)     0
 stributed)

 time_distributed_27 (TimeDi  (None, 10, 2, 2, 64)     36928
 stributed)

 time_distributed_28 (TimeDi  (None, 10, 1, 1, 64)     0
 stributed)

 time_distributed_29 (TimeDi  (None, 10, 64)           0
 stributed)

 lstm_1 (LSTM)               (None, 32)                12416

 dense_3 (Dense)             (None, 2)                 66

=================================================================
Total params: 72,994
Trainable params: 72,994
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 `LRCN` model. As we had checked for the previous model.

```
# 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_18_input | input: | [(None, 10, 64, 64, 3)]
InputLayer | output: | [(None, 10, 64, 64, 3)]
time_distributed_18(conv2d_4) | input: | (None, 10, 64, 64, 3)
TimeDistributed(Conv2D) | output: | (None, 10, 64, 64, 16)
time_distributed_19(max_pooling2d_4) | input: | (None, 10, 64, 64, 16)
TimeDistributed(MaxPooling2D) | output: | (None, 10, 16, 16, 16)
time_distributed_20(dropout_9) | input: | (None, 10, 16, 16, 16)
TimeDistributed(Dropout) | output: | (None, 10, 16, 16, 16)
time_distributed_21(conv2d_5) | input: | (None, 10, 16, 16, 16)
TimeDistributed(Conv2D) | output: | (None, 10, 16, 16, 32)
time_distributed_22(max_pooling2d_5) | input: | (None, 10, 16, 16, 32)
TimeDistributed(MaxPooling2D) | output: | (None, 10, 4, 4, 32)
time_distributed_23(dropout_10) | input: | (None, 10, 4, 4, 32)
TimeDistributed(Dropout) | output: | (None, 10, 4, 4, 32)
time_distributed_24(conv2d_6) | input: | (None, 10, 4, 4, 32)
TimeDistributed(Conv2D) | output: | (None, 10, 4, 4, 64)
time_distributed_25(max_pooling2d_6) | input: | (None, 10, 4, 4, 64)
TimeDistributed(MaxPooling2D) | output: | (None, 10, 2, 2, 64)
time_distributed_26(dropout_11) | input: | (None, 10, 2, 2, 64)
TimeDistributed(Dropout) | output: | (None, 10, 2, 2, 64)
time_distributed_27(conv2d_7) | input: | (None, 10, 2, 2, 64)
TimeDistributed(Conv2D) | output: | (None, 10, 2, 2, 64)
time_distributed_28(max_pooling2d_7) | input: | (None, 10, 2, 2, 64)
TimeDistributed(MaxPooling2D) | output: | (None, 10, 1, 1, 64)
time_distributed_29(flatten_3) | input: | (None, 10, 1, 1, 64)
TimeDistributed(Flatten) | output: | (None, 10, 64)

Step 5.2: Compile & Train the Model

After checking the structure, we will compile and start training the model.

```
# 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 22/70
111/111 [==============================] - 1s 8ms/step - loss: 0.2663 - accuracy: 0.8844 - val_loss: 0.2584 - val_accuracy: 0.9189
Epoch 23/70
111/111 [==============================] - 1s 9ms/step - loss: 0.2275 - accuracy: 0.9093 - val_loss: 0.6378 - val_accuracy: 0.8468
Epoch 24/70
111/111 [==============================] - 1s 8ms/step - loss: 0.2254 - accuracy: 0.9070 - val_loss: 0.2635 - val_accuracy: 0.9009
Epoch 25/70
111/111 [==============================] - 1s 9ms/step - loss: 0.2131 - accuracy: 0.9252 - val_loss: 0.2353 - val_accuracy: 0.9189
Epoch 26/70
111/111 [==============================] - 1s 9ms/step - loss: 0.2379 - accuracy: 0.8889 - val_loss: 0.2242 - val_accuracy: 0.9189
Epoch 27/70
111/111 [==============================] - 1s 9ms/step - loss: 0.1924 - accuracy: 0.9274 - val_loss: 0.2462 - val_accuracy: 0.9279
Epoch 28/70
111/111 [==============================] - 1s 10ms/step - loss: 0.2222 - accuracy: 0.9161 - val_loss: 0.2283 - val_accuracy: 0.9099
Epoch 29/70
111/111 [==============================] - 1s 12ms/step - loss: 0.2134 - accuracy: 0.9093 - val_loss: 0.2067 - val_accuracy: 0.9189
Epoch 30/70
111/111 [==============================] - 1s 12ms/step - loss: 0.1840 - accuracy: 0.9161 - val_loss: 0.2007 - val_accuracy: 0.9369
Epoch 31/70
111/111 [==============================] - 1s 9ms/step - loss: 0.1660 - accuracy: 0.9274 - val_loss: 0.2015 - val_accuracy: 0.9279
Epoch 32/70
111/111 [==============================] - 1s 9ms/step - loss: 0.1730 - accuracy: 0.9320 - val_loss: 0.2509 - val_accuracy: 0.9189
Epoch 33/70
111/111 [==============================] - 1s 8ms/step - loss: 0.1637 - accuracy: 0.9252 - val_loss: 0.2614 - val_accuracy: 0.9369
Epoch 34/70
111/111 [==============================] - 1s 9ms/step - loss: 0.1621 - accuracy: 0.9365 - val_loss: 0.1884 - val_accuracy: 0.9459
Epoch 35/70
111/111 [==============================] - 1s 8ms/step - loss: 0.1673 - accuracy: 0.9365 - val_loss: 0.1803 - val_accuracy: 0.9369
Epoch 36/70
111/111 [==============================] - 1s 8ms/step - loss: 0.1632 - accuracy: 0.9388 - val_loss: 0.2153 - val_accuracy: 0.9369
Epoch 37/70
111/111 [==============================] - 1s 8ms/step - loss: 0.1819 - accuracy: 0.9161 - val_loss: 0.1863 - val_accuracy: 0.9459
Epoch 38/70
111/111 [==============================] - 1s 8ms/step - loss: 0.1974 - accuracy: 0.9297 - val_loss: 0.2332 - val_accuracy: 0.9279
Epoch 39/70
111/111 [==============================] - 1s 9ms/step - loss: 0.1796 - accuracy: 0.9365 - val_loss: 0.2016 - val_accuracy: 0.9459
Epoch 40/70
111/111 [==============================] - 1s 9ms/step - loss: 0.1308 - accuracy: 0.9433 - val_loss: 0.2661 - val_accuracy: 0.9279
Epoch 41/70
111/111 [==============================] - 1s 10ms/step - loss: 0.1350 - accuracy: 0.9501 - val_loss: 0.1960 - val_accuracy: 0.9459
Epoch 42/70
111/111 [==============================] - 1s 13ms/step - loss: 0.1197 - accuracy: 0.9501 - val_loss: 0.2296 - val_accuracy: 0.9369
Epoch 43/70
111/111 [==============================] - 1s 12ms/step - loss: 0.1484 - accuracy: 0.9456 - val_loss: 0.2629 - val_accuracy: 0.9369
Epoch 44/70
111/111 [==============================] - 1s 10ms/step - loss: 0.1479 - accuracy: 0.9478 - val_loss: 0.2626 - val_accuracy: 0.9550
Epoch 45/70
111/111 [==============================] - 1s 9ms/step - loss: 0.1134 - accuracy: 0.9546 - val_loss: 0.3459 - val_accuracy: 0.9189
Epoch 46/70
111/111 [==============================] - 1s 9ms/step - loss: 0.1526 - accuracy: 0.9388 - val_loss: 0.2539 - val_accuracy: 0.9459
Epoch 47/70
111/111 [==============================] - 1s 9ms/step - loss: 0.1421 - accuracy: 0.9456 - val_loss: 0.2411 - val_accuracy: 0.9369
Epoch 48/70
111/111 [==============================] - 1s 8ms/step - loss: 0.1300 - accuracy: 0.9524 - val_loss: 0.1946 - val_accuracy: 0.9369
Epoch 49/70
111/111 [==============================] - 1s 8ms/step - loss: 0.1386 - accuracy: 0.9524 - val_loss: 0.2670 - val_accuracy: 0.9459
Epoch 50/70
111/111 [==============================] - 1s 9ms/step - loss: 0.1524 - accuracy: 0.9456 - val_loss: 0.2503 - val_accuracy: 0.9459
```

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)
```

```
6/6 [==============================] - 0s 56ms/step - loss: 0.2542 - accuracy: 0.9076
```

Save the Model

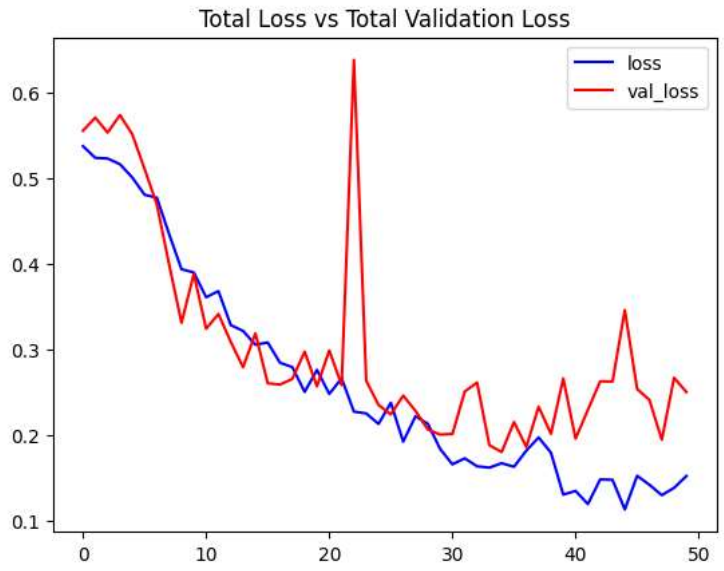
After that, we will save the model for future uses using the same technique we had used for the previous model.

```
# 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}.h

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

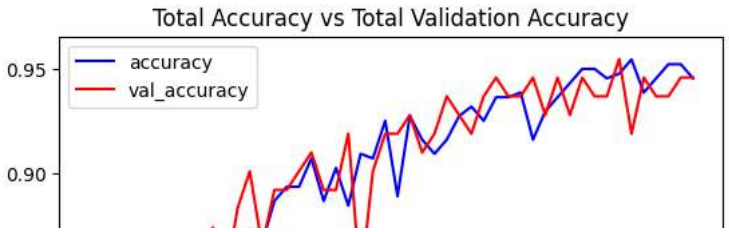
Step 5.3: Plot Model's Loss & Accuracy Curves

Now we will utilize the function `plot_metric()` we had created above to visualize the training and validation metrics of this model.

```
# 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(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 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(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, original_video_height))

    # 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 lies between 0 and 1.
        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 = convlstm_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]
            print(predicted_class_name)
        # 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)
```

```
        # 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'/content/test_videos/gh.mp4'

# Perform Action Recognition on the Test Video.
predict_on_video('/content/900-1_900-2873-PD2_preview.mp4', output_video_file_path, SEQUENCE_LENGTH)

# Display the output video.
VideoFileClip('/content/900-1_900-2873-PD2_preview.mp4', audio=False, target_resolution=(300,None)).ipython_display()
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
<ipython-input-62-82f0fb18e145> in <cell line: 8>()
      6
      7 # Display the output video.
----> 8 VideoFileClip('/content/900-1_900-2873-PD2_preview.mp4', audio=False, target_resolution=(300,None)).ipython_display()

                                  2 frames
/usr/local/lib/python3.10/dist-packages/moviepy/video/io/ffmpeg_reader.py in ffmpeg_parse_infos(filename, print_infos, check_duration,
fps_source)
    268     lines = infos.splitlines()
    269     if "No such file or directory" in lines[-1]:
--> 270         raise IOError(("MoviePy error: the file %s could not be found!\n"
    271                        "Please check that you entered the correct "
    272                        "path.")%filename)

OSError: MoviePy error: the file /content/900-1_900-2873-PD2_preview.mp4 could not be found!
Please check that you entered the correct path.
```

✓ 0s completed at 5:49 PM