A Deep Learning Framework for Human Action Recognition on YouTube Videos

Rahul Darelli [a], Mahathi L P V S [b], Koushik M V [c] and Praveen Kumar Dr. Kollu [d]

[a] *V R Siddhartha Engineering College, Kanuru, Vijayawada, 520007, India*

Abstract

Human Action Recognition or HAR is still a challenging problem that has been surrounded by various studies and experiments in the last decade. Due to the approach of deep learning techniques such as CNNs, it has become possible to improve the performance of HAR systems over traditional method. CNNs have been widely used in the analysis of image and when comes to LSTM networks, these networks work as a better version when prediction and analysis of sequence of data is involved but when we combine both of them, we get the best versions of both CNNs [7], LSTM [8]. So that difficult computer vision problems like video classification can be solved. Here we train and test the deep learning models ConvLSTM and LRCN using the dataset UCF50 - Action Recognition Dataset. In which UCF50 dataset consists of 50 action categories and each category grouped of 25 videos per action. Where ConvLSTM uses the concept similar to the LSTM [9] approach that which uses result processing and its computation simultaneously. Whereas LRCN is combination of convolution and LSTM layers that were mounted in single model. We find the accuracy of both ConvLSTM and LRCN models. The best model out of above two mentioned models that which is considered based on highest accuracy among both and is taken to test the accuracy model on YouTube videos to predict the human action that which is performed.

Keywords

HAR, CNNs, LSTM, ConvLSTM, UCF50 dataset, 50 actions, 25 videos per action, LRCN, accuracy, YouTube videos.

1. Introduction

Recognition of human activities from various sources like recorded videos, real time cameras start from various image processing technics. If we need to understand what exactly an image consists, we just input this image to an image classifier or to a pre-trained deep neural network. Just like that, videos are also a collection of frames or image. So, recognition of an activity on a video or real time camera is just analyzing all the collected frames, using an image classifier at each frame. And labeling the output. One on each frame and finally deciding most frequently occurred label among the frames as output. So, this is one of the tradition approaches for recognizing human activity from a sequence of data that which involved in videos. But this approach may not be correct all the time because this traditional approach is not effective at the same time it doesn't considers all the aspects that involved in the video. For example, if in the sequence of collected frames has jumping but most probably involves standing, Since by Using above mention traditional approach provides standing action as output. But it actually has jumping activity. So, in such cases this approach fails to give accurate results.

Another approach for recognition of human activity is using CNN which comes under deep neural network. Generally, CNN works by taking an image and generating feature maps, which are

WINS-2022: Workshop on Intelligent Systems, April 22 – 24, 2022, Chennai, India.
EMAIL: 188w1a0573@vrsiddhartha.ac.in (Rahul Darelli)
ORCID: 0000-0001-9697-1220 (Rahul Darelli)

representations of a certain feature or location in the image. As the network gets deeper, the number of feature maps gets increased. However, the disadvantage of this approach expensive at the same time its computation is very slow. Since working on videos involves sequenced data, the best method when comes to this type of data LSTM is a best approach. Here the limitation is that we ignore all the data other than land markings.

So, when we combine both CNN and LSTM results in better accuracy for recognizing activities of human. Using this approach, we will be able extracting spatial features from a video sequence and then identifying temporal relations between frames. And this way of combining CNN and LSTM is called LRCN approach. The proposed model LRCM achieved an accuracy of 92.62%. Whereas ConvLSTM model achieved an accuracy of 80.33%. On comparing both LRCN achieved more accuracy, so we use LRCM model on YouTube videos to recognize involved activities of humans. We use pafy [15] library to download videos on colab that which only requires URL of YouTube videos.

2. Literature Survey

T. Liu, Y. Song, Y. Gu and A. Li [1] proposed a methodology for activity recognition basing on humans using Microsoft Kinect sensors, Hidden Markov Models (HMMs), and k-means clustering. The method mainly contains two modules training the actions using above mentioned model and identify actions using sensors. In training the action module, the first depth image is taken as input and the skeleton information is derived and features are extracted, as in the frames have frequently having similar coordinates, so to divide them into 50 frames and to convert them into clusters they used k-means clustering algorithm and secondly, they created HMM model with three hidden layers. Next, in the human recognition model again the same process of action learning is repeated up to frames extraction then clusters assignment is done and an observation sequence is produced. And action recognized is given as output. This study is done on seven actions like movementing hands in upwards and downwards directions, pushing forward and circle construction in both clockwise and counter anti clockwise. They did this approach by inspiring from high accuracy in posture and gesture recognition by Kinect as it innovatively separates an action into several clusters for recognition. And this method has achieved an accuracy of 91.4%.

Kamel, B. Sheng, P. Yang, P. Li, R. Shen, and D. D. Feng [2] proposed two types of data sequence that are used as the inputs. And they are Joint posture sequence and Depth map sequence. After they are transformed to the descriptor, the descriptor used for body posture is MJD and for depth map is DMI. Next, preprocessing of the input is done. And three CNN models are trained with three different channels (Ch1, Ch2, and Ch3) and they are tested with different inputs. In three one CNN channel is trained for Depth map images, another one is trained with joint postures and another one is trained with both joint posture and depth map images. Using score fusion operation all the outputs are fused and the final action is classified.

N. Jaouedi, N. Boujnah, O. Htiwich, and M. S. Bouhlel [3] mainly concentrated on behavior analysis on humans from recorded one's that is from camera or any other electronic source and they also focused on actions in the background like fast walking, and sudden movement. This model was mainly designed for predicting behavior on humans through their movement analysis. In this study, they explained human action recognition by using the K Nearest Neighbors approach. In this study, they used GMM model or Gaussian Mixture Model which generally used for data analysis. GMM is a mainly focus on areas which the current state pixel changes from previous state in a sequence of collections of frames. The proposed algorithm run on each frame image converts into binary images for better performance. For this they deleclared 0 for black to to their background and 1 for white background. Kalman Filter method is used for moving human tracking. And these filters are used in two phases frequently, the two phases are prediction and correction. In the prediction phase, it calculates the current state by using the information of previous state. The main agenda of this study is to get an efficient output. At last, classification is done using the KNN method and achieved a rate

of 71.1%.

Q. Xiao and Y. Si [4], they used a deep neural network model which uses autoencoder, PRNN or pattern recognition neural network to predict action performed by human beings. They used two approaches a learning stage system and an action recognition stage. In the learning stage system, they build a binary frame for each frame by drawing the outlines of human body outlines and then join all the frames. And these they used these frames to train their model. In another approach, they used autoencoder to train the model to predict characteristics of actions. After these two approaches they train the PRNN model using unsupervised learning technic. In the end, they merge the autoencoder followed by PRNN model named as APRNN. To calculate the APRNN's performance they used Weizmann motion data which consists of 93 different action recognition recorded clips with 10 motion semantics. For achieving better performance, they used fine-tuning.

Y. Ji, Y. Yang, F. Shen, H. T. Shen and X. Li [5], mainly studied on the analysis of human actions in robotic platforms. They consider the various steps involved in the recognition and prediction of human actions. In this paper divided the human action recognition field into three main categories: hand gesture-based HRI, body action-based HRI, and multi-modal fusion. They discussed the various platforms and datasets that are commonly used in the field of HRI. They also discussed the various challenges and opportunities in the field of action analysis for human recognition. They concluded that in the future, data should be built to address the storage problems related to data.

For 3D action recognition, the authors. L. Wang, D. Q. Huynh and P. Koniusz [6] proposed a total of ten Kinect-based algorithms that are used on six datasets. These algorithms are for cross-view and cross-subject detection. And the algorithms used by them are HON4D, HDG, LARP-SO, HOPC, SCK+DCK, P-LSTM, HPM+TM, clips+CNN+MTLN, indRNN, and ST-GCN. A 3D action analysis was also done to compare the results of cross-object and cross-view action recognition. It was concluded that depth-based action recognition techniques are better at recognizing objects with greater details.They performed an extensive evaluation of the HDG representation with various variants of the descriptor types. They also introduced four variants of the P-LSTM framework.

3. Planned Procedure

The main objective is to develop a ConvLSTM and LRCN model at the same time to predict the accuracy of both the models. And is to perform activity prediction on YouTube videos using the LRCN model.

As the proposed model ConvLSTM [11] achieved an accuracy of 80.33% and LRCN achieved 92.62%. Since we are considering highest accuracy LRCN is best among both. We test LRCN model on YouTube videos.

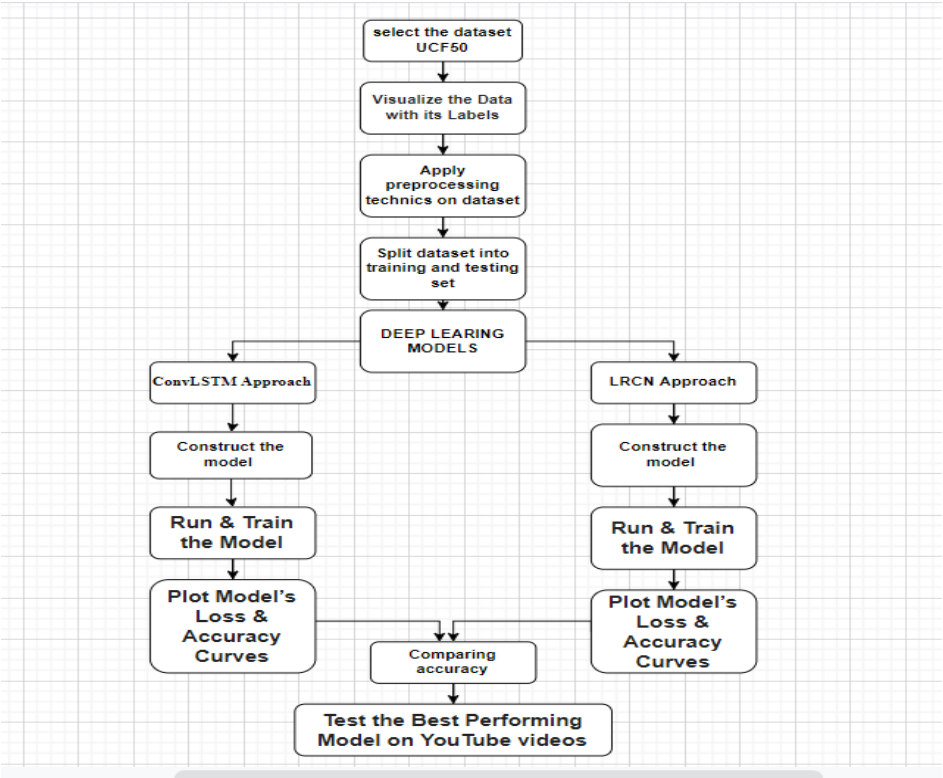

Figure 1: Proposed Architecture

Above mentioned diagram is about our proposed architecture. Our proposed work involves 4 modules and they are preprocessing the dataset, Divide the structured dataset into training and testing set, implementing ConvLSTM, implementing LSTM, to **test accurate model on YouTube videos.**

3.1. Module - 1 Pre-Processing the Dataset

Data pre-processing is a process of extraction use full information from collected raw data. To perform data pre-processing, initially we must choose a relevant raw data. The raw data we chose is UCF50- which consists raw data of different activities.

In which our data set consists of 50 activity categories. In each category there are 25 videos. And on average our dataset has 133 videos per each action category. On an average each video has 199 frames. Average frames per width is 320 and height is 240. Averagely there are 26 frames per second for each video. For testing, we only selected 20 random categories. The first frame of the selected video is represented by its associated labels. This method will allow us to identify the first 20 random videos in the dataset. Now, we perform some pre-training on the dataset. This process involves 2 steps. First step is to create a extract_frames() function. And the second step is to create dataset_creation() function.

In the first step, extract_frames () function is created to extract the frames and extracted frames are resized into 255 pixels. And the normalization of frames is done. That is unwanted frames that which doesn't contains any information of activity are removed. And finally, this function returns useful frames. Then after we set frame width and height to 64 x 64 pixels which is a general format for any pre-processing technic. And Sequence length is set to 20 which we will use as default case in entire project.

In the second step, a dataset_creation () function is created. In which this method involves mapping of features, labels and video path for each category of video that we selected above. For skipping the frames, we use the following formula.

Formula: skip_f = max(int(v_f_c/S_L), 1)

Where, v_f_c represents count of video frames
S_L represents sequence length i.e., S_L = 20.

Since, our dataset UCF50 [10] consists of on average 199 frames per video so we chose sequence length as 20. How our concept skipping the frames works is like if there are 199 frames in video then 199/20 i.e., 9.95~10 that is we skip every 10^{th} frame from the sequence of frames from the video. Finally, we convert different classes of encoded indexes to one-hot. For this we use keras's which is a built-in library in python.

Dataset Name : UCF-50
Source : UCF Research Center
Link : https://www.crcv.ucf.edu/data/ucf-cc-50/

Figure 2: Dataset Information

3.2. Module - 2 divide the structured data-set into training and testing set

Before training and testing [12] of our proposed models first we need to divide our constructed structured dataset into training dataset and testing dataset. The most important requirements for splitting of data are features and one_hot_encoded_labels on categorical data. So, with the help of these requirements we divide the dataset into 75% as training dataset and 25% as testing dataset. To avoid any kind of bias we rearrange the dataset by putting shuffle = True. And also, we set random_state to 27. We split the dataset using sklearn library in python. Where training and testing dataset is used on both the models ConvLSTM and LRCN models.

3.3. Module - 3 Implementing convLSTM approach

In this module, we introduce the concept of ConvLSTM cells. They are typically an integral part of an LSTM network that can be used to identify spatial features of the data.

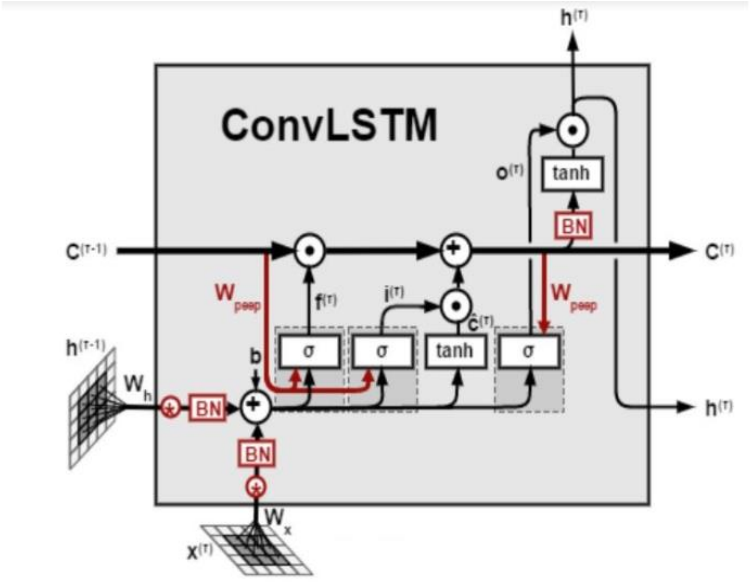

Figure 3: ConvLSTM Architecture

This method is useful for video classification as it captures the spatial relation between the various frames. It can also take in 3D input, which is different from the usual approach for modeling Spatio-temporal data.

Step – 1: We construct our model by using Keras ConvLSTM2d recurrent layers. The layers are inputted to the Dense layer, which predicts and match the output to its associated activity. MaxPooling3D [14] layers are used to minimize the number of frames and prevent overfitting the model. Here we focus on limited data thus we do not require a larger capability model.

While constructing this ConvLSTM [11] model we use 4 filters for initial layer and then go on increase the filters layer by layer i.e., in the second layer we increase to 8 filters. In the third layer we go on increase to 14 filters and in the 4th layer we give 18 filters to the model. In each layer we commonly set kernel_size to (3,3). And we set activation layer as tanh, format of data to channels last, 2% is set as dropout for each state, we enable sequence to be returned at each state. After adding each layer, we do maxpooling3D [14] to minimize the number of frames. Below figure is the summary of our model ConvLSTM.

```
[ ]  Model: "sequential_2"
     _________________________________________________________________
     Layer (type)                 Output Shape              Param #
     =================================================================
     conv_lst_m2d_8 (ConvLSTM2D)  (None, 20, 62, 62, 4)     1024
     _________________________________________________________________
     max_pooling3d_8 (MaxPooling3 (None, 20, 31, 31, 4)     0
     _________________________________________________________________
     time_distributed_6 (TimeDist (None, 20, 31, 31, 4)     0
     _________________________________________________________________
     conv_lst_m2d_9 (ConvLSTM2D)  (None, 20, 29, 29, 8)     3488
     _________________________________________________________________
     max_pooling3d_9 (MaxPooling3 (None, 20, 15, 15, 8)     0
     _________________________________________________________________
     time_distributed_7 (TimeDist (None, 20, 15, 15, 8)     0
     _________________________________________________________________
     conv_lst_m2d_10 (ConvLSTM2D) (None, 20, 13, 13, 14)    11144
     _________________________________________________________________
     max_pooling3d_10 (MaxPooling (None, 20, 7, 7, 14)      0
     _________________________________________________________________
     time_distributed_8 (TimeDist (None, 20, 7, 7, 14)      0
     _________________________________________________________________
     conv_lst_m2d_11 (ConvLSTM2D) (None, 20, 5, 5, 16)      17344
     _________________________________________________________________
     max_pooling3d_11 (MaxPooling (None, 20, 3, 3, 16)      0
     _________________________________________________________________
     flatten_2 (Flatten)          (None, 2880)              0
     _________________________________________________________________
     dense_2 (Dense)              (None, 4)                 11524
     =================================================================
     Total params: 44,524
     Trainable params: 44,524
     Non-trainable params: 0
     _________________________________________________________________
     Model Created Successfully!
```

Figure 3: Summary of ConvLSTM model

The created ConvLSTM model has a total of 44,524 parameters and Trainable parameters are 44,524. And there are zero non trainable parameters. Below figure contains detailed layers information of ConvLSTM model.

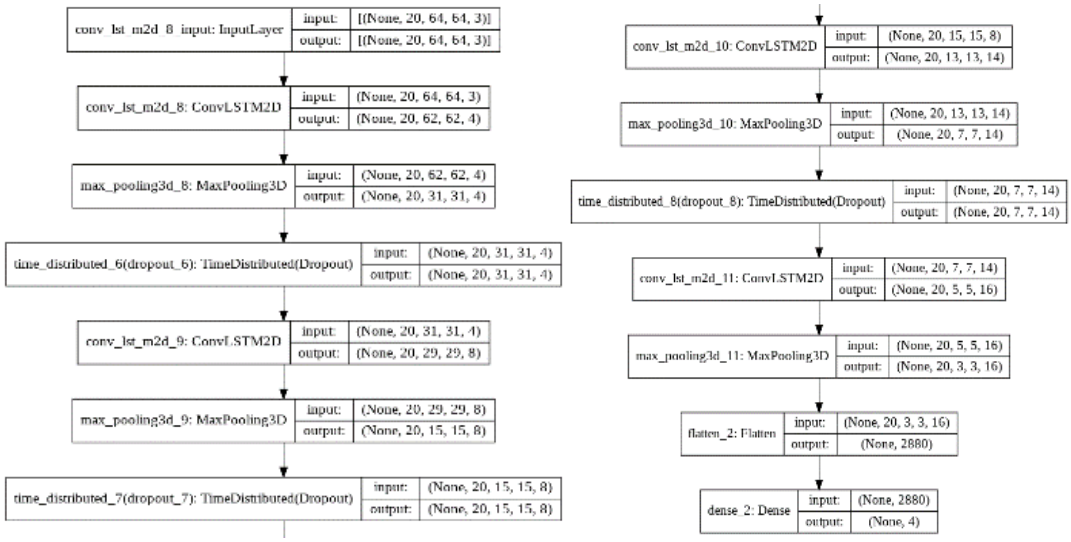

Figure 4: Layers of ConvLSTM2D

Step – 2: After constructing the model now we need to train the model. Initially we Create an Instance for Early Stopping_Callback(). And then we test our model accuracy by considering loss percentage and another parameter. And then we start training the model. During the training of the model, we set maximum length of batch to 4, before training we randomize our data- set. Below is the summary of trained model ConvLSTM.

```
Epoch 18/50
73/73 [==============================] - 153s 2s/step - loss: 0.0219 - accuracy: 0.9966 - val_loss: 0.4479 - val_accuracy: 0.9041
Epoch 19/50
73/73 [==============================] - 154s 2s/step - loss: 0.0049 - accuracy: 1.0000 - val_loss: 0.4653 - val_accuracy: 0.8904
Epoch 20/50
73/73 [==============================] - 154s 2s/step - loss: 0.0033 - accuracy: 1.0000 - val_loss: 0.4832 - val_accuracy: 0.8904
Epoch 21/50
73/73 [==============================] - 154s 2s/step - loss: 0.0034 - accuracy: 1.0000 - val_loss: 0.4561 - val_accuracy: 0.8904
Epoch 22/50
73/73 [==============================] - 153s 2s/step - loss: 0.0023 - accuracy: 1.0000 - val_loss: 0.4832 - val_accuracy: 0.8904
Epoch 23/50
73/73 [==============================] - 152s 2s/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.4463 - val_accuracy: 0.8904
Epoch 24/50
73/73 [==============================] - 153s 2s/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.4655 - val_accuracy: 0.8904
Epoch 25/50
73/73 [==============================] - 154s 2s/step - loss: 9.4172e-04 - accuracy: 1.0000 - val_loss: 0.4625 - val_accuracy: 0.9041
Epoch 26/50
73/73 [==============================] - 153s 2s/step - loss: 8.4565e-04 - accuracy: 1.0000 - val_loss: 0.4504 - val_accuracy: 0.9178
Epoch 27/50
73/73 [==============================] - 154s 2s/step - loss: 7.3753e-04 - accuracy: 1.0000 - val_loss: 0.4469 - val_accuracy: 0.9178
```

Figure 5: Summary of training

Step – 3: After the training we test the model using 25% test dataset. The evaluation of trained model as follows.

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

     4/4 [==============================] - 14s 3s/step - loss: 0.8976 - accuracy: 0.8033
```

Figure 6: Evaluation on trained model

Finally using ConvLSTM we got an accuracy of 80.33% on test dataset.

3.4. Module - 4 LRCN approach

Step – 1, In this module we merge the CNN and the LSTM layers as a single unit. Form extracted frames to generate feature map we use CNN model. The combination of both LSTM and LRCN uses CNN and LST in order to perform in the most effective manner. It learns spatiotemporal features through end-to-end training.

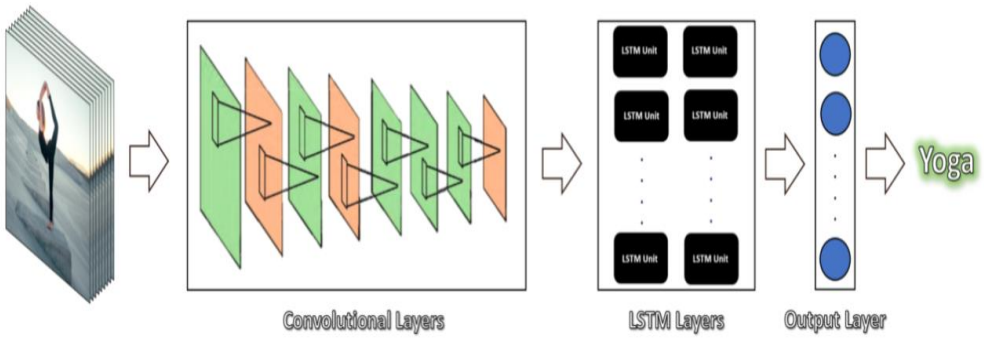

Figure 7: LRCN Architecture

Above mention fig.7 is the architecture diagram of LRCN model. Here, we implement the LRCN architecture by using time-distributed Conv2D [13] layers. The Conv2D layer is merged with LSTM layer. The next step is the Flatten layer, which will flatten the Conv2D feature. Here we used sequential model for constructing our model. Initially we add Time Distributed Convolution2D layer to our model and then we set 16 filters for initial layer and on consecutive layers we going on increasing filters to 32, 64, 64. Activation is set to relu. And at each layer we perform maxpooling 2d in order to removed unwanted frames, and also, we done dropout which helps our model to ignore randomly selected neurons. After adding these layers, we add Flatten layer to input to the next layer. And then after we add LSTM layer. And in the end, we add dense layer to collect all the inputs from the previous layers. Below figure is the summary of our proposed LSTM model.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed_22 (TimeDis (None, 20, 64, 64, 16)    448
_________________________________________________________________
time_distributed_23 (TimeDis (None, 20, 16, 16, 16)    0
_________________________________________________________________
time_distributed_24 (TimeDis (None, 20, 16, 16, 16)    0
_________________________________________________________________
time_distributed_25 (TimeDis (None, 20, 16, 16, 32)    4640
_________________________________________________________________
time_distributed_26 (TimeDis (None, 20, 4, 4, 32)      0
_________________________________________________________________
time_distributed_27 (TimeDis (None, 20, 4, 4, 32)      0
_________________________________________________________________
time_distributed_28 (TimeDis (None, 20, 4, 4, 64)      18496
_________________________________________________________________
time_distributed_29 (TimeDis (None, 20, 2, 2, 64)      0
_________________________________________________________________
time_distributed_30 (TimeDis (None, 20, 2, 2, 64)      0
_________________________________________________________________
time_distributed_31 (TimeDis (None, 20, 2, 2, 64)      36928
_________________________________________________________________
time_distributed_32 (TimeDis (None, 20, 1, 1, 64)      0
_________________________________________________________________
time_distributed_33 (TimeDis (None, 20, 64)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 32)                12416
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 132
=================================================================
Total params: 73,060
Trainable params: 73,060
Non-trainable params: 0
```

Figure 8: Summary of LRCN Model

Our proposed model LRCN has a total of 73,060 Trainable parameters. And has zero non-Trainable parameters. Out of them a total of 60,512 CNN parameters and 9548 LSTM parameters. On plotting our model, we got below results.

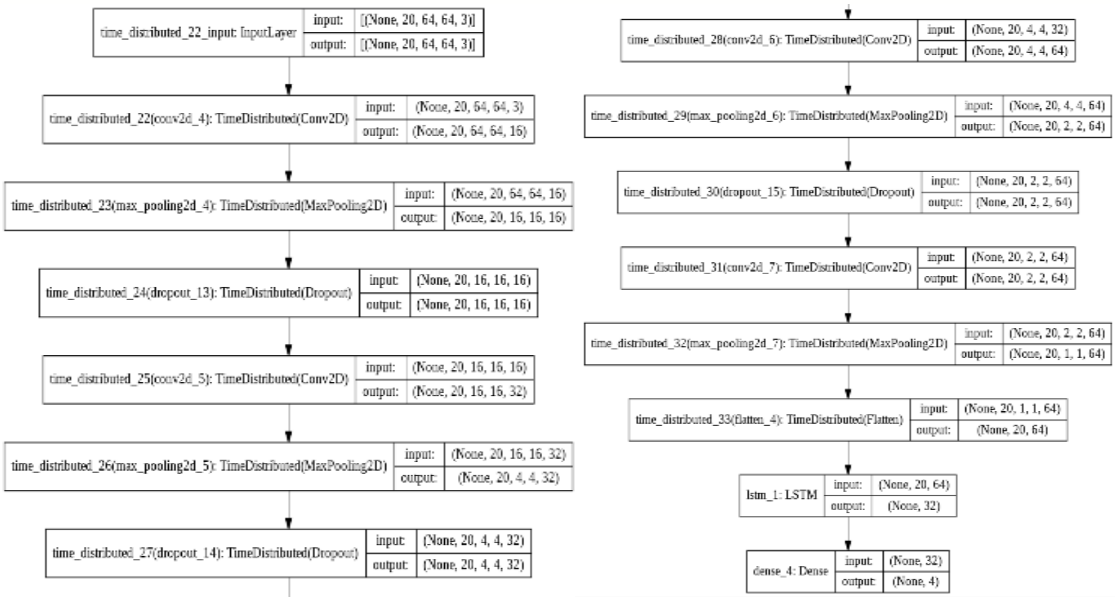

Figure 9: Layers of LRCN Model

Step – 2: After constructing our LRCN model we then train our LRCN model using 75% of train dataset of UCF50 dataset. For this we initially create instance for Early Stopping Callback. And then we compile and test the LRCN model using some loss factor and another parameter. We test our model by setting epochs to 70, batch_size to 4, shuffling of test dataset to true. Below is the accuracy summary on test dataset on LRCN model.

```
73/73 [==============================] - 14s 185ms/step - loss: 0.0816 - accuracy: 0.9658 - val_loss: 0.4080 - val_accuracy: 0.8904
Epoch 32/70
73/73 [==============================] - 14s 193ms/step - loss: 0.1015 - accuracy: 0.9726 - val_loss: 0.1573 - val_accuracy: 0.9452
Epoch 33/70
73/73 [==============================] - 15s 204ms/step - loss: 0.0198 - accuracy: 0.9966 - val_loss: 0.1103 - val_accuracy: 0.9726
Epoch 34/70
73/73 [==============================] - 13s 182ms/step - loss: 0.0887 - accuracy: 0.9760 - val_loss: 0.2069 - val_accuracy: 0.9452
Epoch 35/70
73/73 [==============================] - 13s 183ms/step - loss: 0.0355 - accuracy: 0.9932 - val_loss: 0.1648 - val_accuracy: 0.9589
Epoch 36/70
73/73 [==============================] - 13s 183ms/step - loss: 0.0222 - accuracy: 0.9966 - val_loss: 0.1908 - val_accuracy: 0.9178
Epoch 37/70
73/73 [==============================] - 13s 183ms/step - loss: 0.0105 - accuracy: 1.0000 - val_loss: 0.1827 - val_accuracy: 0.9589
Epoch 38/70
73/73 [==============================] - 13s 184ms/step - loss: 0.0642 - accuracy: 0.9760 - val_loss: 0.1956 - val_accuracy: 0.9178
Epoch 39/70
73/73 [==============================] - 13s 184ms/step - loss: 0.1323 - accuracy: 0.9623 - val_loss: 0.1693 - val_accuracy: 0.9178
Epoch 40/70
73/73 [==============================] - 13s 183ms/step - loss: 0.0305 - accuracy: 0.9966 - val_loss: 0.1708 - val_accuracy: 0.9315
Epoch 41/70
73/73 [==============================] - 13s 184ms/step - loss: 0.0349 - accuracy: 0.9932 - val_loss: 0.3548 - val_accuracy: 0.9178
Epoch 42/70
73/73 [==============================] - 13s 184ms/step - loss: 0.0119 - accuracy: 1.0000 - val_loss: 0.2215 - val_accuracy: 0.9315
Epoch 43/70
73/73 [==============================] - 13s 184ms/step - loss: 0.0067 - accuracy: 1.0000 - val_loss: 0.2456 - val_accuracy: 0.9452
Epoch 44/70
73/73 [==============================] - 13s 184ms/step - loss: 0.0079 - accuracy: 1.0000 - val_loss: 0.3804 - val_accuracy: 0.9178
```

Figure 10: Accuracy summary on Test data-set

The highest accuracy we got among the test dataset is 97.26% accuracy. Which is better than ConvLSTM. Below is the evaluation of LRCN model on 25% of test dataset.

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

     4/4 [==============================] - 2s 418ms/step - loss: 0.2242 - accuracy: 0.9262
```

Figure 11: Evaluation on LRCN Model

We got 92.62 accuracy by evaluating our LRCN model on test dataset. Since the accuracy of ConvLSTM is 80.33% and LRCN is 92.62. So, basing on highest accuracy i.e., 92.62% among both, we chose LRCN model to test on YouTube videos.

3.5. Module - 5 to test LRCN module on YouTube videos

Step – 1: In this module initial step is to implement the LRCN model to test TaiChi and horse racing acitives from the source YouTube. We create a function that which used for downloading selected the videos from YouTube. Another function that will be created will predict the video frame's path and save the results. We download YouTube videos using pafy library. Below is the function for downloading any YouTube video. We just need give the URL of video.

```
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 downloading.
    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 contructed path.
    video_best.download(filepath = output_file_path, quiet = True)

    # Return the video title.
    return title
```

Figure 12: Function for Downloading YouTube Video

Step – 2: In the second step we create a function that performs recognition on selected videos. This function predict_single_action () method takes URL of video, sequence length which we set 20 at initial stage as inputs. This method divides specified video by download using above mentioned download function into frames in order to predict the action. And we perform pre-processing technics on those frames. And finally, we give these pre-processed frames to our LRCN model to predict action recognition on given video.

4. Results and Analysis

Our proposed model LRCN gave an accuracy of 97.0059% to selected YouTube video. The Human Action that involved in our selected videos are one video it involves horse racing and another video it has TaiChi – which a involves a series of physical exercises and stretches. Our model LRCN on these YouTube videos recognized successfully with the accuracy of 97.0059.

```
video_title = download_youtube_videos('https://youtu.be/fc3w827kwyA', test_videos_directory)

# 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_display()
```

```
Action Predicted: TaiChi
Confidence: 0.9700598120689392
100%|██████████| 6223/6223 [00:18<00:00, 327.54it/s]
```

Figure 13: Accuracy of LRCN Model

Above figure is about the accuracy of our model LRCN achieved i.e.97.0059%.

4.1. Results of action prediction on YouTube videos as follows

a. Horse Race

Figure 14.1: Horse-race Prediction

Figure 14.2: Horse-race Prediction

Our model successfully predicted HorseRace Action successfully.

b. TaiChi Human Action Prediction

Figure 15.1: Taichi Human Action Prediction in 26th second

Figure 15.2: Taichi Human Action Prediction in 31st second

Our Model Successfully predicted Taichi Huma Action Prediction using LRCN model.

5. Conclusion and Future Work

Human Action Recognition is often used in the development of surveillance systems and other systems designed for the elderly. It can help individuals with learning and memory loss. The agenda of our study is to develop an assistive technology system that which will allow our work to help elderly individuals to live a more connected life with an efficient way. As every human being on earth have a desire to live forever. But living by human earth is maximized to 100 years. Yet they live more than that they lose their memory and forget how to do all other human activities. So, for such desired persons we can implement our concept of action recognition. So that if we know what they have to done like walking, running, then we can use this system to predict and perform those activities that elderly person wishes to do. Not for elderly persons we implement this concept to who lose memory or who are not able to perform their activities.

6. References

[1] T. Liu, Y. Song, Y. Gu and A. Li, "Human Action Recognition Based on Depth Images from Microsoft Kinect," 2013 Fourth Global Congress on Intelligent Systems, 2013, pp. 200-204, doi: 10.1109/GCIS.2013.38.

[2] Kamel, B. Sheng, P. Yang, P. Li, R. Shen, and D. D. Feng, "Deep Convolutional Neural Networks for Human Action Recognition Using Depth Maps and Postures," in IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 49, no. 9, pp. 1806-1819, Sept. 2019, doi: 10.1109/TSMC.2018.2850149.

[3] N. Jaouedi, N. Boujnah, O. Htiwich, and M. S. Bouhlel, "Human action recognition to human behavior analysis," 2016 7th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT), 2016, pp. 263-266, doi: 10.1109/SETIT.2016.7939877.

[4] Q. Xiao and Y. Si, "Human action recognitan ion using autoencoder," 2017 3rd IEEE International Conference on Computer and Communications (ICCC), 2017, pp. 1672-1675, doi: 10.1109/CompComm.2017.8322824.

[5] Y. Ji, Y. Yang, F. Shen, H. T. Shen and X. Li, "A Survey of Human Action Analysis in HRI Applications," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 30, no. 7, pp. 2114-2128, July 2020, doi: 10.1109/TCSVT.2019.2912988.

[6] L. Wang, D. Q. Huynh and P. Koniusz, "A Comparative Review of Recent Kinect-Based Action Recognition Algorithms," in IEEE Transactions on Image Processing, vol. 29, pp. 15-28, 2020, doi: 10.1109/TIP.2019.2925285

[7] "A Comprehensive Guide to Convolutional Neural Networks". [Online] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53W

[8] "Long short-term memory (LSTM) " [Online] https://en.wikipedia.org/wiki/Long_short-term_memory

[9] "Long-term Recurrent Convolutional Network (LRCN) ". [online] https://kobiso.github.io/research/research-lrcn/

[10] "UCF50 - Action Recognition Data Set ". [Online] https://www.crcv.ucf.edu/data/UCF50.php

[11] "An introduction to ConvLSTM". [Online] https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7

[12] "How to Split a Dataset Into Training and Testing Sets with Python". [online] https://towardsdatascience.com/how-to-split-a-dataset-into-training-and-testing-sets-b146b1649830

[13] "Keras.Conv2D Class". [Online] https://www.geeksforgeeks.org/keras-conv2d-class/

[14] "MaxPooling3D layer". [Online] https://keras.io/api/layers/pooling_layers/max_pooling3d/

[15] "Introduction to Pafy Module in Python ". [Online] https://www.geeksforgeeks.org/introduction-to-pafy-module-in-python/