

UNITEDWORLD INSTITUTE OF TECHNOLOGY

(UIT)

Summative Assessment (SA)

Submitted BY

Shaily Antala

(Enrl. No.: 20220701037)

**Course Code and Title: 21BSAI99E43 - Artificial Neural Networks**

B.Sc. (Hons.) Computer Science / Data Science / AIML

IV Semester – December – April 2024

UIT

Dec/Apr 2024

# **REPORT ON HUMAN ACTIVITY RECOGNITION (CNN + LSTM)**

## **INTRODUCTION**

The area of HAR, or human activity recognition, has garnered substantial interest in recent times owing to its extensive applicability across several domains, including healthcare, sports, security, and human-computer interaction. HAR is the automated detection and categorization of human actions using sensor data, usually gathered from cell phones or wearable technology. The capacity of Artificial Neural Networks (ANNs) to recognize intricate patterns and correlations in data has made ANNs an effective tool for HAR.

## **OBJECTIVE**

This report's goal is to give a general overview of the topic concerning artificial neural network-based human activity recognition, including a discussion of the methods, difficulties, applications, and future possibilities.

## **METHODOLOGY**

The architecture and operation of artificial neural networks, or ANNs, are computer models influenced by biological brain networks. They are made up of linked nodes arranged in layers, including an output layer, an input layer, and one or more hidden layers. The following procedures are commonly included in HAR employing ANNs:

Data collection: Wearable technology or cellphones worn by people engaged in a variety of activities are the source of sensor data, such as accelerometer and gyroscope readings.

Preprocessing of the Data: The gathered data are preprocessed to eliminate noise, screen outliers, and standardize characteristics. This is a critical stage in optimizing the neural network model's performance.

Feature extraction: From the preprocessed sensor data, pertinent features are taken out. Time-domain features (such as mean and standard deviation), frequency-domain features (such as Fourier transformations), and statistical attributes are often utilized characteristics.

Model Training: An artificial neural network model is trained using the preprocessed data and features that were retrieved. During training, the network's weights and biases are modified by the application of optimization methods like gradient descent or its variations.

Model Evaluation: To determine how well the model that has been trained recognizes human behaviors, it is tested on a different dataset. Assessment measures that are frequently used to assess HAR model effectiveness include accuracy, precision, recall, and F1-score.

## **CHALLENGES**

Artificial neural networks are quite good at recognizing human action, but there are still a number of issues that need to be resolved. These include:

limited Training Data: It may be difficult to construct reliable neural network simulations due to little or unbalanced annotated training data for HAR.

Generalization: Domain adaptation and transfer learning strategies are needed since HAR models that have been trained on particular activities may find it difficult to generalize to other activities or people.

Real-time Processing: The complex computations and storage demands for neural network models are constrained by the need to handle sensor input in real-time for HAR applications.

Sensor Location and Orientation: Wearable device sensor location and orientation can have an impact on the accuracy and dependability of the data gathered, which can impact how well HAR models work.

## **APPLICATIONS**

Artificial neural networks for human activity recognition have significance for a number of fields, such as:

Healthcare: Tracking activity levels and identifying odd behavior patterns in patients with long-term illnesses or the elderly.

Sports and Fitness: Monitoring sports performance, examining motion patterns, and offering input to enhance training regimens.

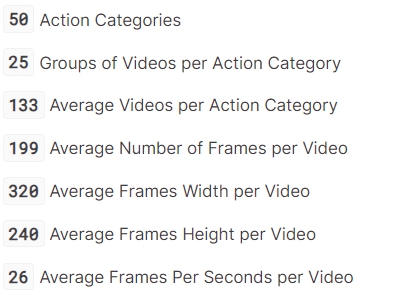
Security: To improve security and protection by spotting unusual or suspicious activity in surveillance footage.

Human-Computer Interaction: Facilitating context-aware interaction and gesture detection in wearables and smart surroundings.

## **DATASET DESCRIPTION**

We will be utilizing the UCF50 - Action Recognition Dataset, which sets itself apart from a majority of motion detection data sets obtainable since it is made up of genuine movies that were produced by performers.

Dataset consists :



Twenty randomly chosen categories and one randomly chosen video from each category will be used for the depiction. The first frame of each of the chosen movies will be displayed along with the labels that correspond to each. We will be able to see a portion of the dataset—20 randomly selected videos—in this fashion.

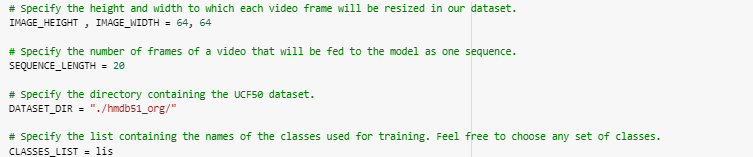
### **Setting the Requirements for the Session**

### **Importing Libraries**

### **Data Visualization with Labels**

### **Dataset Preprocessing**

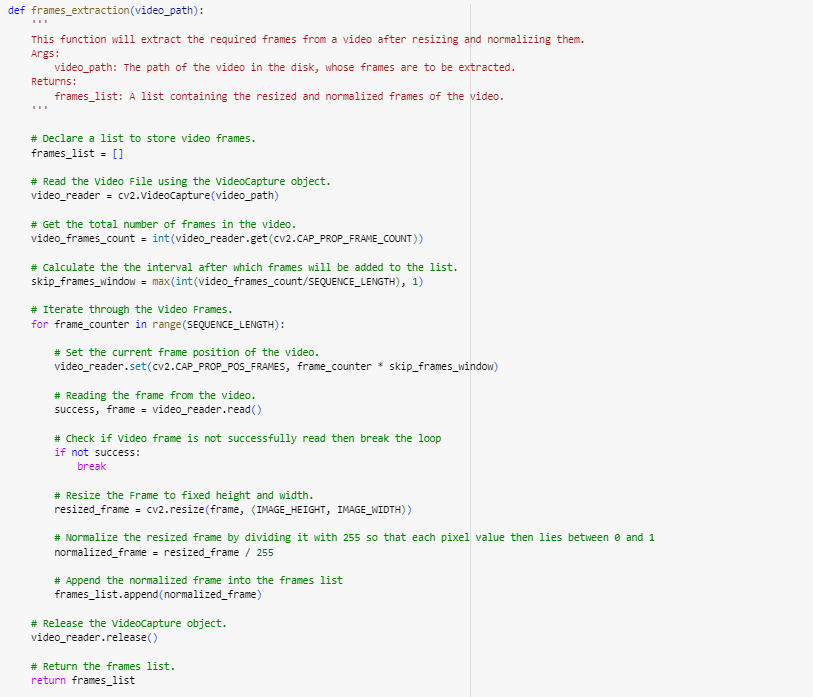
We're going to preprocess the dataset a little. In order to minimize calculations, we begin by reading the video clips from the database, compress each frame of video to a fixed width and height, then normalize the data to the range [0-1] by dividing the pixel values by 255. This will speed up convergence while the network is being trained.

However, let's establish a few constants first.

For improved results, you can raise the values of the IMAGE\_HEIGHT, IMAGE\_WIDTH, and SEQUENCE\_LENGTH constants. However, note that raising the sequence length will only have a limited effect, and doing so will make the process more complex and costly.

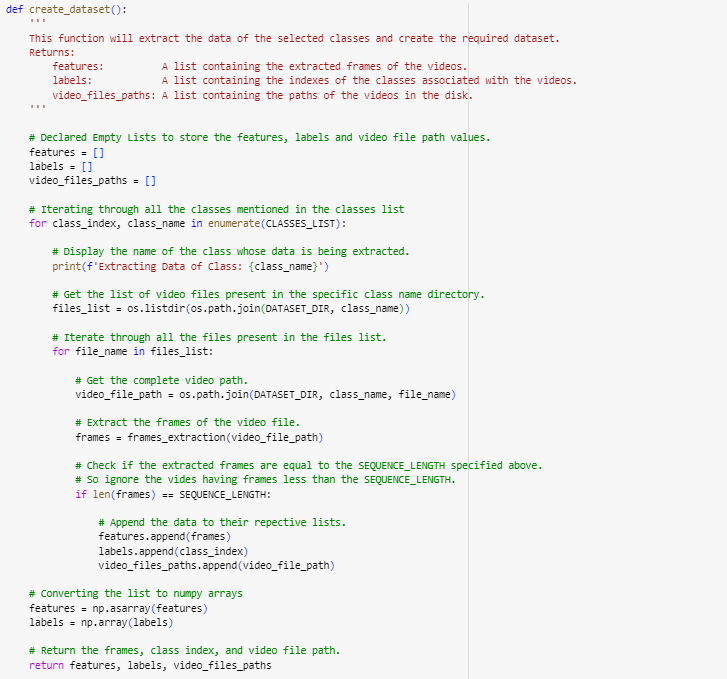
**Develop a Function for Extracting, Resizing, and Normalizing Frames**

We'll write the method frames\_extraction(), which takes a path as an input and returns a list of the normalized and compressed frames from the movie. Since we just require an equally dispersed sequential length of frames, not every frame will be added to the list when the function reads the video file frame by frame.

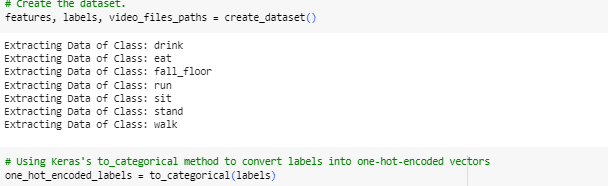


**Establish a Function for Dataset Generation**

We will now create a function called create\_dataset() that will repeat the process through every one of the classes listed in the CLASSES\_LIST constant. It will call the function frame\_extraction() on each video file belonging to the classes that we have chosen, returning the video file path (video\_files\_paths), the class index (labels), and the frames (features).

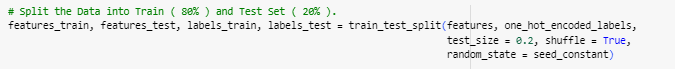


We will now use the previously constructed method create\_dataset() to extract the data from the chosen classes and generate the necessary dataset.



### **Dataset split into Train and Test set**

The necessary features are already available: one\_hot\_encoded\_labels, another Numpy array that has every class label in a single hot encoded format, and a NumPy array that contains all of the retrieved frames from the films. To generate training and testing sets, we will now divide our data. Before the split, we will rearrange the dataset to remove bias and ensure that the divides accurately reflect the distribution of the data as a whole.



### **Implementing ConvLSTM**

This stage will use a mix of ConvLSTM cells to accomplish the first strategy. An LSTM network variation that incorporates convolutional operations is known as a ConvLSTM cell. Since convolution is a part of the design of this LSTM, it can recognize spatial elements in the data while taking the temporal relationship into consideration.

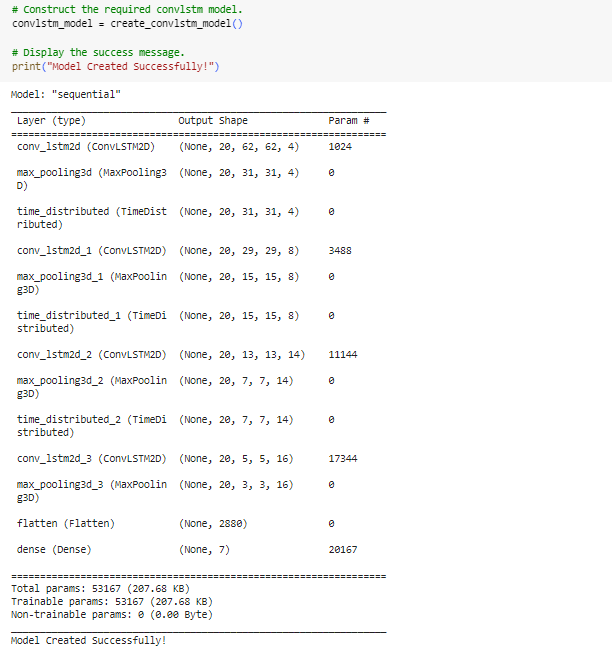
This method successfully captures the temporal relationship between several frames as well as the spatial relationship inside each frame for the purpose of classifying videos. Because of its convolution structure, the ConvLSTM can process three-dimensional input (width, height, and number of channels), but a basic LSTM can only process one-dimensional input; hence, an LSTM by itself is not suitable for modeling spatiotemporal data.

**Build the Model**

We are going to employ Keras ConvLSTM2D recurrent layers for developing the model. The variety of kernel dimensions and filters needed to apply the convolutional operations are also taken into account by the ConvLSTM2D layer. Upon completion, the layers' output is flattened and supplied to the Dense layer, which utilizes softmax activation to provide the likelihood of every action category.

Additionally, Dropout layers will be applied to prevent the model from being overfit to the data and MaxPooling3D layers to minimize the frame dimensions and eliminate needless computations. There aren't many trainable parameters in this straightforward design. This is so that a large-scale model is not necessary because we are just working with a tiny portion of the information.

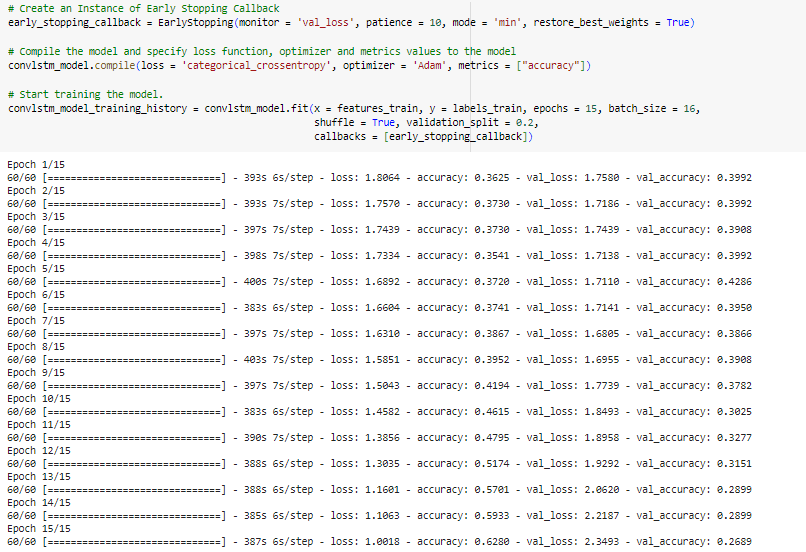
To build the necessary conv-lstm model, we will now use the method create\_convlstm\_model() that was previously developed.



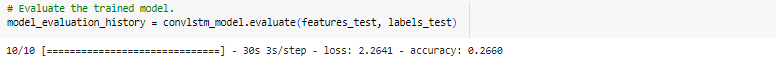
**Verify the Model's Structure**

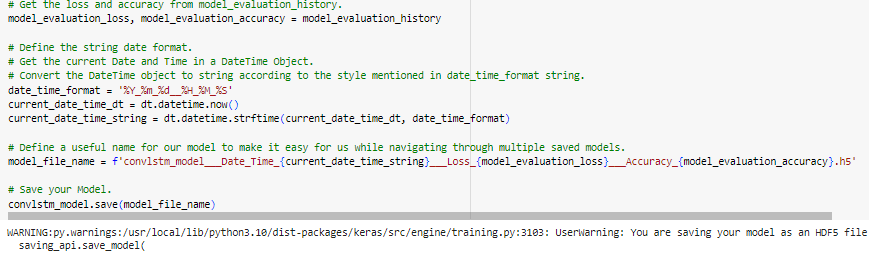
In order to ensure that the network is generated correctly and is complicated, we will now utilize the plot\_model() method to verify the structure of the constructed model.

### **Compile and Train the Model**

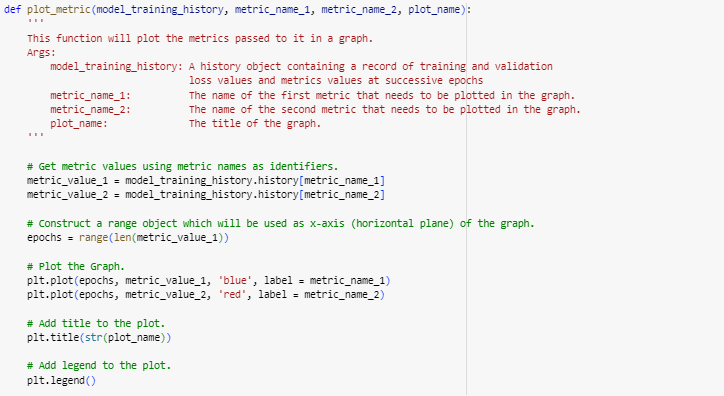
After the model is compiled, we will add an early halting callback to avert overfitting and begin the training.

### **Evaluation and Saving the Model**

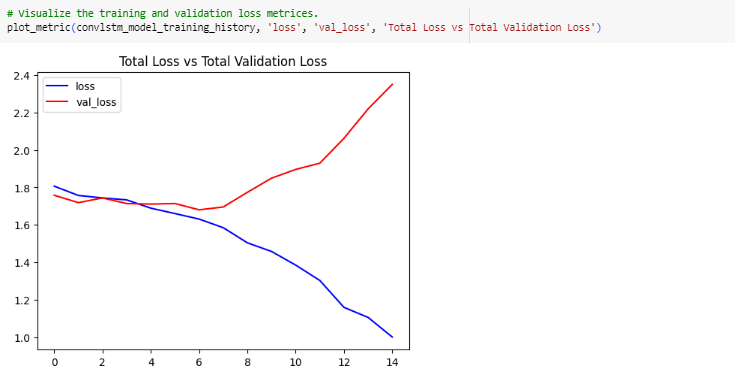
We will assess the model using the test set after training.

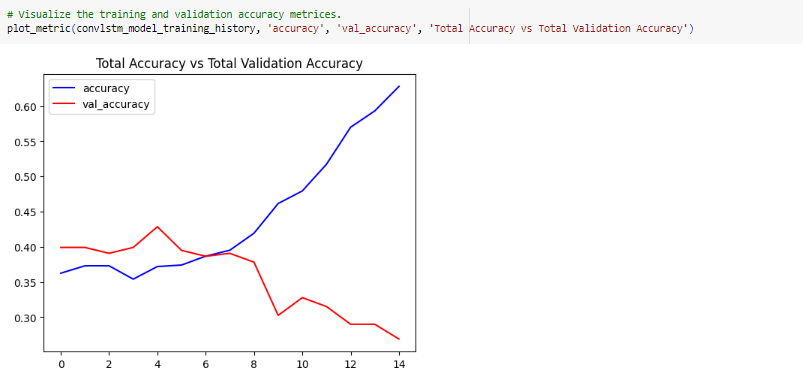
We will now store the model so that we won't have to train it from the start each time we use it.

### **Plotting Model’s Accuracy and Loss Curves**

To see the training and validation metrics, we will now develop a method called plot\_metric(). Our training and validation procedures already have distinct metrics, so all we need to do now is visualize them.

The function plot\_metric() that was previously built will now be used to see and interpret the metrics.





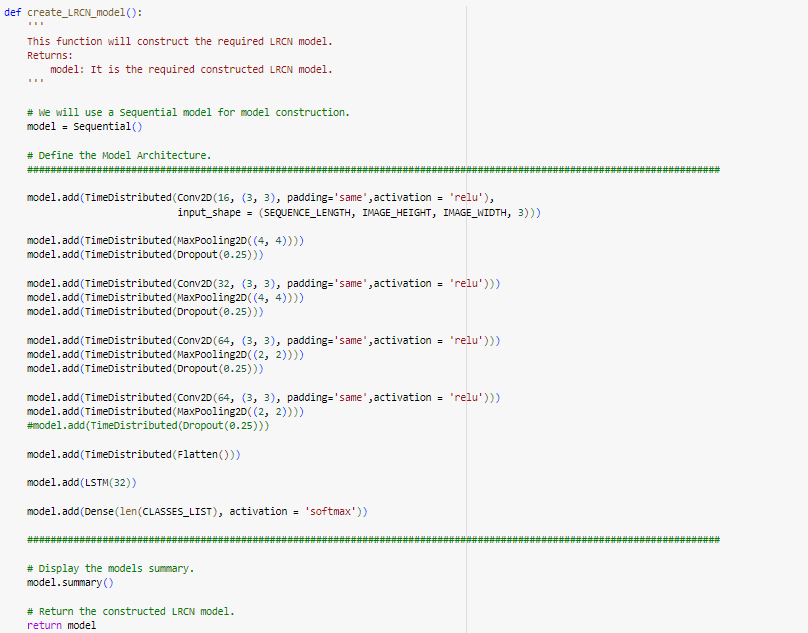
### **Implementing LRCN**

In this stage, we will integrate the LSTM and convolution layers into a single model to apply the LRCN Approach. Using a CNN model and an LSTM model that were trained independently is another comparable strategy. A pre-trained model that has been adjusted for the issue may be utilized to extract spatial characteristics from the video frames using the CNN model. Afterwards, the LSTM model can forecast the activity shown in the video by using the CNN-extracted characteristics.

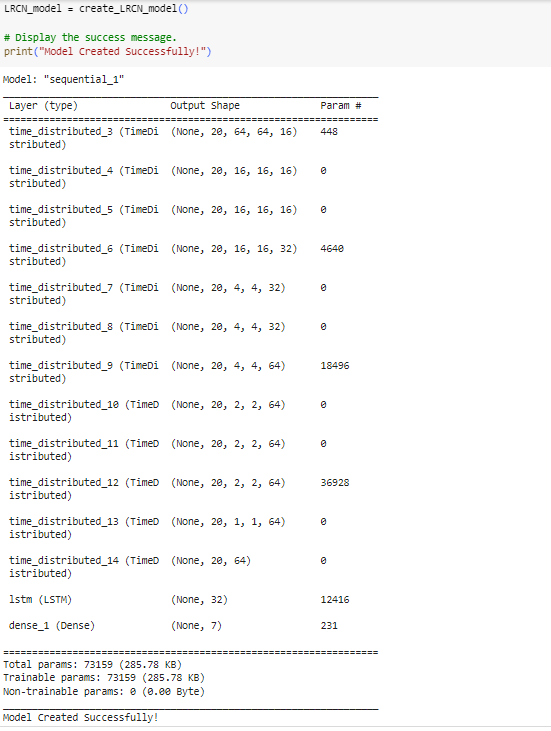
Here, however, we'll use a different strategy called the Long-term Recurrent Convolutional Network (LRCN), which integrates LSTM and CNN layers into a single model. Convolutional layers are utilized to extract spatial characteristics from the frames; the LSTM layer(s) receives the retrieved spatial data at each time step for temporal sequence modeling. In this manner, the network immediately picks up spatiotemporal information during end-to-end training, producing a reliable model.

### **Constructing the Model**

Time-distributed Conv2D layers, MaxPooling2D, and Dropout layers will be used in order to execute our LRCN architecture. The feature that was taken from the Conv2D layers will next be given to an LSTM layer after being flattened using the Flatten layer. The output of the LSTM layer is then used by the Dense layer with softmax activation to forecast the action that is being taken.



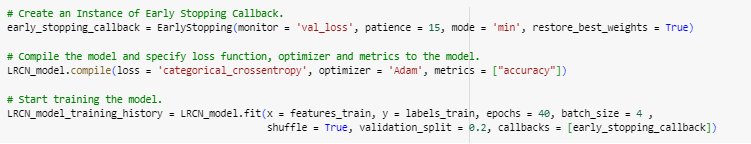
To build the necessary LRCN model, we will now use the method create\_LRCN\_model() that was previously built.



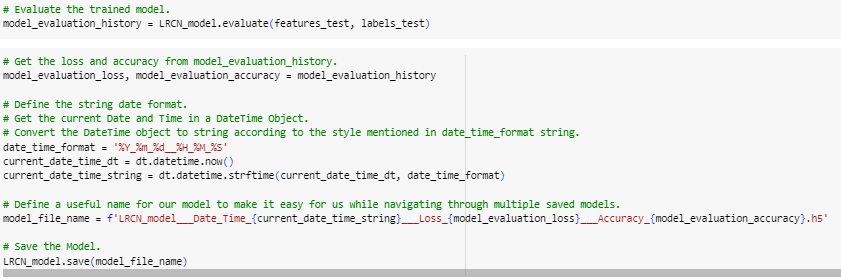
**Examine the structure of the model**

The structure of the created LRCN model will now be examined using the plot\_model() function. Similar to how we had looked for the prior model.

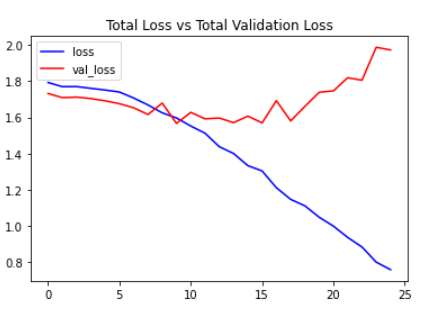
### **Compile and Train the Model**

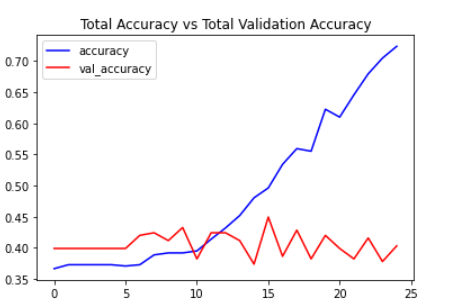


### **Evaluation and Saving the Model**



### **Plotting Model’s Accuracy and Loss Curves**

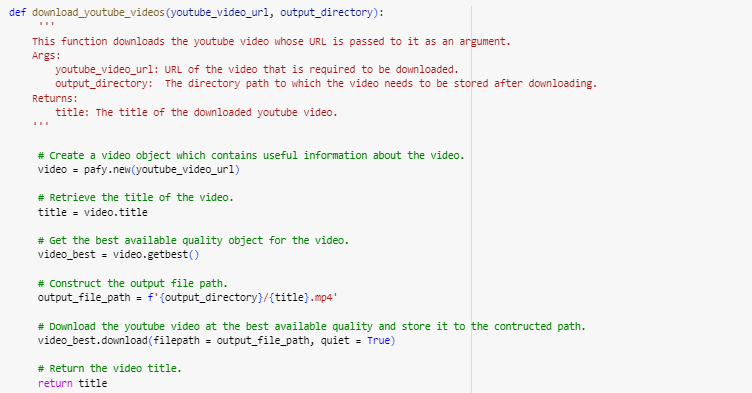




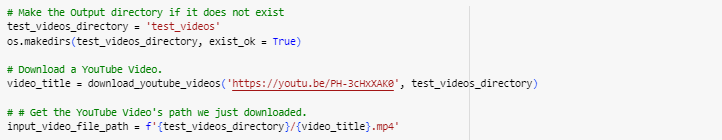
### **Testing best performing model on YouTube videos**

It appears from the data that the LRCN model did notably well for a limited set of classes. Thus, we will test the LRCN model on a few YouTube videos in this stage.

**Construct a Function to Download YouTube Videos**

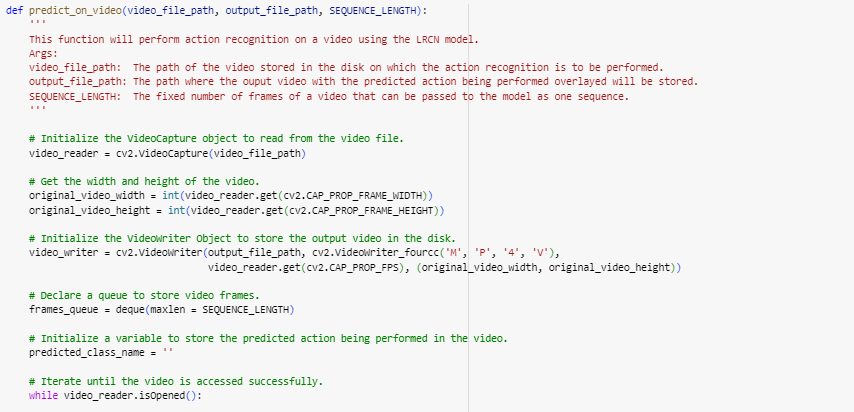
Using the pafy library, we will write the function download\_youtube\_videos() to begin downloading the YouTube videos. To download a video and its accompanying information, such as the film's title, the library simply needs the video's URL.

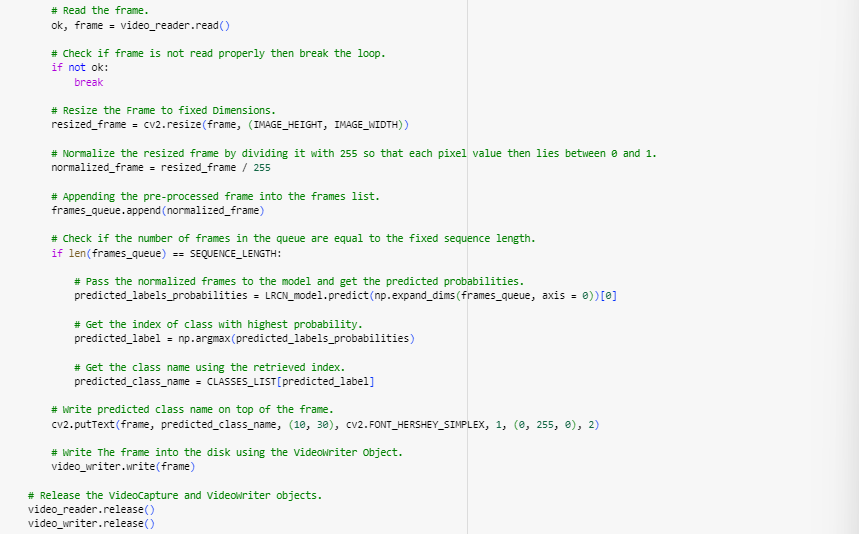
The procedure involves downloading a YouTube video that will be used to test the LRCN model. To achieve this, we will use the function download\_youtube\_videos() that was built before.



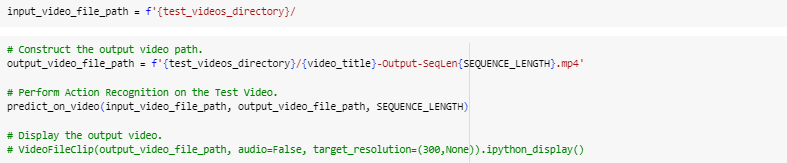
**Develop a Feature for Action Recognition in Videos**

Next, we'll write the function predict\_on\_video(), which will do nothing more than read a video file frame by frame from the path that is supplied as an input, recognize actions on the video, and save the output.



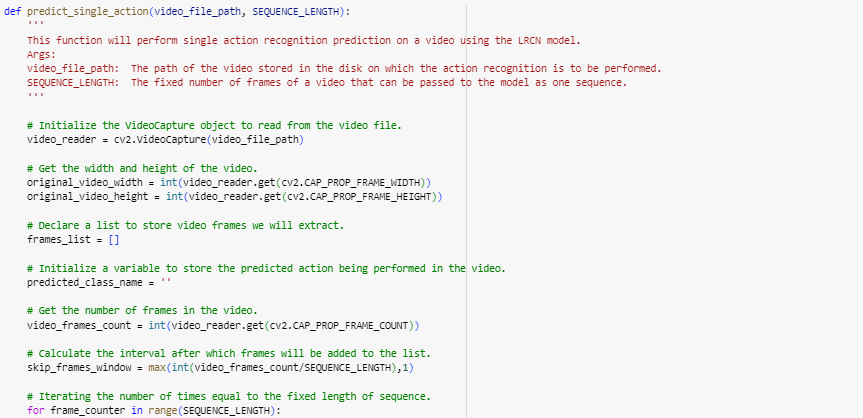


**Using the test video, carry out action recognition**

The test video we obtained using the download\_youtube\_videos() function will now be subjected to action recognition using the method predict\_on\_video(), which we have built previously. The result video will then be displayed with the predicted action superimposed on it.

**Make a Function That Can Only Make One Video Prediction**

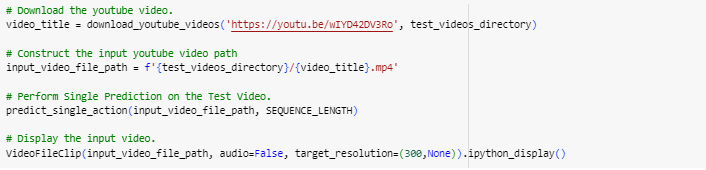
Let's now develop a function that will make a single prediction for each full video. From the full movie, we will extract N (SEQUENCE\_LENGTH) uniformly dispersed frames, which we will then feed to the LRCN model. This method saves a lot of time and calculation when working with films that only have one activity. It is really helpful in those situations.



### **Prediction**

**Utilizing a test video, make a single prediction**

Now, we will download the entire YouTube test video using the function download\_youtube\_videos() that we had previously developed, and use the method predict\_single\_action() to make a single prediction on it.



## **FUTURE DIRECTIONS**

Artificial neural networks have a bright future in the identification of human action, with many opportunities for further study and advancement. These include:

Multi-sensor fusion: combining information from several sensors (such as magnetometers, gyroscopes, and accelerometers) to increase the precision and resilience of HAR models.

Investigating sophisticated deep learning architectures for detecting temporal and spatial correlations in sensor data, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs).

Edge computing: creating thin neural network models to enable real-time processing and on-device HAR applications while maximizing inference efficiency on resource-constrained devices.

Privacy and Ethics: This section discusses privacy issues as well as moral issues surrounding the gathering and use of personal data to identify human activities.

## **CONCLUSION**

Artificial neural networks that recognize human behavior have a lot of potential uses in interactions between humans and machines, sports, healthcare, and security. Notwithstanding obstacles including scarce training data and instantaneous processing limitations, it is anticipated that continuous research and developments in deep learning architectures and sensor technologies would propel additional improvements in this sector. With the growing need for tailored services and context-aware computing, HAR with ANNs is expected to be a key component in determining the direction of human-machine interaction in the future.