5-2023

HUMAN SUSPICIOUS ACTIVITY DETECTION

Nilamben Bhuva

HUMAN SUSPICIOUS ACTIVITY DETECTION

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

by

Nilamben Bhuva

May 2023

HUMAN SUSPICIOUS ACTIVITY DETECTION

A Project

Presented to the

Faculty of

California State University,

San Bernardino

by

Nilamben Bhuva

May 2023

Approved by:

Dr. Jennifer Jin, Advisor, Computer Science and Engineering

Dr. Bilal Khan, Committee Member

Dr. Ronald Salloum, Committee Member

ABSTRACT

The detection of suspicious human activity is a crucial aspect of ensuring public safety and security. The aim is to identify suspicious behavior. To accomplish this, we employ the LRCN, a long-term recurrent convolutional network, to detect anomalous activity. It is important to consider the temporal data of the video when classifying suspicious behavior, and the framework uses a combination of CNNs and RNNs to analyze video frames and extract relevant features. The key milestones of this project include conducting research, collecting and pre-processing data, designing and training the model, and evaluating its performance. The resulting detection system can accurately identify suspicious behavior in real-time. To build the model, we used the KTH dataset, which includes 600 frames of walking and running, as well as the Kaggle dataset, which consists of 100 training videos. Our model analysis shows that the system and video can detect suspicious events with an accuracy of 86%, and we anticipate that this accuracy will improve as the dataset size increases.

ACKNOWLEDGEMENTS

It is with great pleasure that I deliver this research report on "Human Suspicious Activity Detection." Our involvement in a Deep Learning and Machine Learning project has been a fascinating experience for us. Without the help and encouragement of those around us, this project would not have been possible.

I want to express my sincere appreciation to Dr. Jennifer Jin, my project's adviser, as well as to the committee members Dr. Bilal Khan and Dr. Ronald Salloum for assistance throughout the entire project development. They inspired us to put forth extra effort and adopt cutting-edge technology. Additionally, we appreciate that our university has included self-development classes in our curriculum that enable us to experiment with various technologies.

TABLE OF CONTENTS

LIST OF IMAGES

CHAPTER ONE

INTRODUCTION

Background

This Project centers around detecting suspicious human behavior. The proliferation of image capture and the abundance of data led to a rising demand for automated systems capable of identifying suspicious behavior in real-time. This is particularly crucial in general areas such as shopping malls, public parks, and train stations where security and safety are of utmost importance [5].

The goal is to create a framework for identifying abnormal activity.

The invention of a technology to recognize suspect human activity that can precisely pinpoint suspicious behavior in real-time is the project's result. Results from the proposed system's evaluation on a dataset of video footage were contrasted with those from earlier trials [6].

Project Milestones:

<u>Research and Literature:</u> The first step in the project is to conduct research and literature review to gain a deep understanding of the present stage in detecting suspicious human activity and the different deep learning-based strategies that have been put forth.

<u>Collection of Data:</u> Gather a collection of labeled videos that can be

Utilized to train and test the suggested system. The dataset contains a diverse range of abnormal behaviors, also typical activity for comparative analysis.

Image Pre-Processing: The video clip should be pre-processed so that deep learning framework can use it. The frames might be enlarged, pertinent are may be cropped, and pixel values may be normalized.

Model Design: Create the deep learning model, which is probably going to Include a CNN & RNN. While RNNs is utilized for temporal analysis, the CNNs is employed for feature extraction.

Model Training & Testing: The collected dataset is utilized to train model & test performance with various metrics such as recall, accuracy and precision. The model can be improved and tuned using this technique.

Significance

The detection of suspicious human behavior has the potential to enhance result significantly. The main aims to contribute to the creation of effective & precise system to detect abnormal activity in surveillance video footage. The suggested system implemented in a range of real-world settings, including public safety, homeland security, and may lead to decrease criminal and terrorist activities.

Purpose

The key point is to enhance its accuracy in detecting anomalous activities. The project's purpose is to identify suspicious human behavior and enhance security and safety in public areas. Such a system implemented at various open locations, including theater and bart stations, to offer latest information about crowd sentiment to identify concern or suspicious individuals, and gives a hazardous signal.

CHAPTER TWO

DESCRIPTION

Libraries, Modules and Software

Google Colab

The cloud-based platform Google Colab is used to train machine learning models. The Colab is fitting to machine learning, coding and analysis. Graphical processing units (GPUs), which can hasten model training, are needed for larger neural networks. The efficient matrix operations needed for activities classification via images are best handled by GPU. Google-Colab is therefore used to train the deep learning models for classifying suspicious activities. It also functions as a version control tool and can keep track of versioning.

Python

Python is a popular general-purpose, garbage-collected, dynamically typed programming language. This programming language is a member of the high-level group and is widely used in the creation of software, websites, computation, automation, and data analytics. Python frameworks for various uses, such as Flask and Django for development.

SKlearn

The SKlearn is arrangement to integrate with scientific and numerical libraries, in order to offers different clustering, classification, and regression methods.

TensorFlow

The ML platform from beginning to end is called TensorFlow. It offers distributed and multi-GPU processing, automatic differentiation, model development, training, and assessment, as well as multidimensional based numerical calculation [3].

NumPy

NumP NumPy is an all-purpose toolkit for dealing with arrays and fundamental Python module for scientific computing. It provides extremely quick multidimensional array objects as well as the ability to manipulate these arrays. Pandas is among the most well-liked Python packages used in data research. It provides fast, user-friendly tools for data analysis.

Pafy

When downloading YouTube videos, Pafy collects metadata including count, rate, and duration. Pafy depend on youtube-dl, so installing it prior is advised for more reliable use.

Moviepy

Moviepy is a module that makes it easy to edit films by allowing you to do things like trim and join, concatenate, insert, compose, animate, add images and objects, and apply video-audio effects like color correction and noise reduction before exporting the finished product in a variety of codecs and formats. It has a user-friendly interface for processing audio-video files and is built using Pygame, a Python module and dependencies.

Keras

Developed with a focus on facilitating quick experimentation, Keras is a Python-based, highly minimalist, and modular neural network library that can operate on top of TensorFlow.

Dataset Information

Six movements are included in the KTH datasets: hand-clapping, hand-waving, walking, running, and boxing.

www.csc.kth.se/cvap/aaction

One movement from Kaggle set.

www.kaggle.com/naveenk903/movie-fight-detection-dataset

There are 100 sequences for each of the six activities in the KTH dataset. The video is shot at 25 frames per second, and each sequence contains over 600 frames. Over 100 footages from movies and YouTube videos make up the Kaggle dataset, which can be used to train algorithms for suspicious behavior (run, fight).

Image Pre-processing

Read and Label Video Frame:

The OpenCV library is used to read videos from class and store label inside an array.

Break Video to Generate a Frames:

The OpenCV Library is utilized to read video, and a sequence of 30 frames is formed by reading only 30 frames at equal time intervals.

.

Frame Resize:

When the total number of pixels needs to be increased or decreased, image scaling is required. To retain the homogeneity of the input photos with the architecture, therefore scaled all the frames to 64 pixels width and 64 pixels height.

Normalization:

Normalization will assist the learning algorithm in learning more quickly and identifying key features from the photos. In order to make each pixel's value range from 0 to 1, we normalized the scaled frame by dividing it by 255.

Store in NumPy Arrays:

A NumPy array is used to hold the 30 scaled and normalized frames in order to provide them as input.

Divide Data into Two Cases

Data is used for training in 75% of cases.

The testing process uses 25% of the data.

Various Human Activities

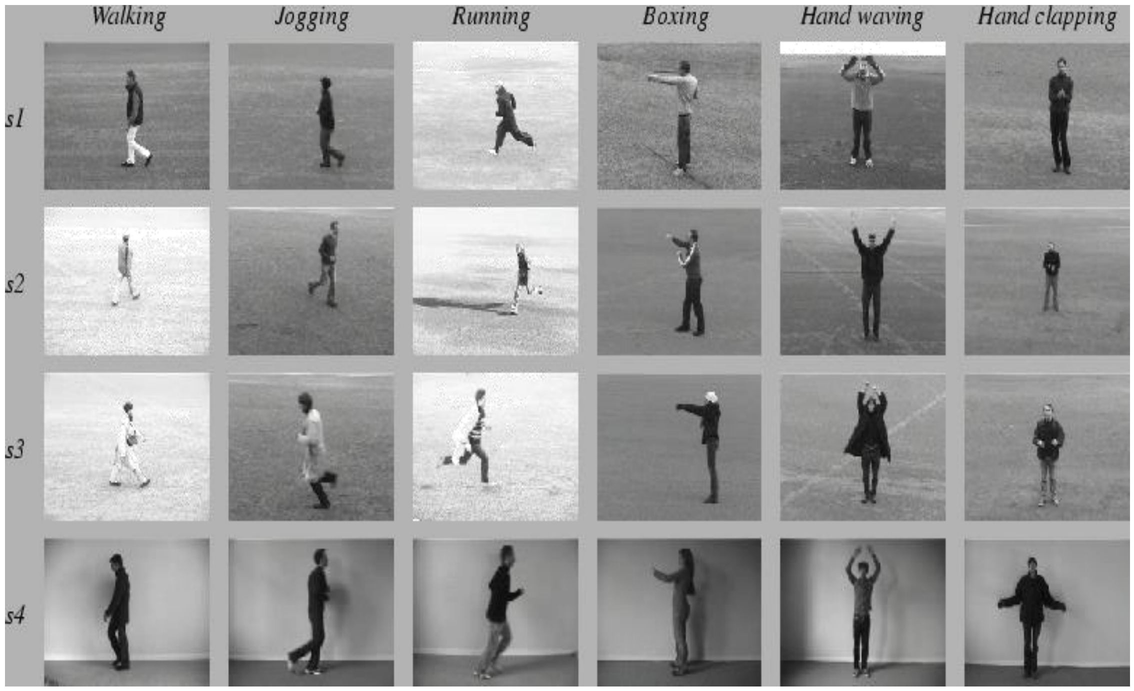


Image 1: Dataset Information

Usage of Technology

The background of the technology utilized in this research for picture categorization has been discussed in general terms in this chapter. This image classification methodology uses deep learning and machine learning methods like LRCN to identify suspicious behavior in video surveillance.

Machine Learning

Computer vision is one of the fields where Deep Learning is used. Deep learning relies solely on computational intelligence, to complete tasks without the use of explicit knowledge. It provides the foundation for the development of statistical tools that can learn from accumulated data and estimate complex functions. The machine learning method known as deep learning is built on artificial neural networks and pattern recognition. The model training process divided into three parts: reinforcement, supervised, and unsupervised learning. The computer is given input with the intention of producing the desired output in supervised learning.

In contrast to the first two methods, reinforcement learning does not require labeled input-output pairs or pattern knowledge. Instead, achieving a balance between exploring newly uncharted territory and making use of existing information is the main goal. Machine learning is utilized in Computer Vision and Image Processing task like category wise image and object, as well as in processing language like analyst text. It is also applied in Speech Processing tasks, such as speech recognition, identification, and enhancement.

The research of machine learning focuses on how to teach computers to pick up new skills without being explicitly instructed, as well as how these machines might make decisions based on all the data machine has about this specific object.

Deep-Learning Technology

Deep learning techniques that can extract properties from raw input, frequently employing an Artificial Neural Network architecture with a large number of layers. A machine learning technique called deep learning teaches computers to learn by mimicking human behavior.

The amount of computer energy required for deep learning is enormous. GPUs, which makes them effective for deep learning. Clusters or cloud computing, enables software developers to reduce the training period for a deep learning network. The training phase used to last for several weeks. Deep learning is a vital component of the technology that enables self-driving cars to recognize a sign. There is a good reason why deep learning is currently attracting so much interest. It is the achievement of exceptional results that were previously unattainable.

The Project’s Flow

This project makes use of (LRCN) to identify unusual behavior. Recognizing the temporal information in the video is vital for accurate classification of anomalous behaviors. CNN has recently been utilized mostly for extracting important characteristics from each video frame. It is vital for CNN to extract the features in order to effectively categorize the input; as a result, CNN must be able to recognize and extract the required features from the video frames.

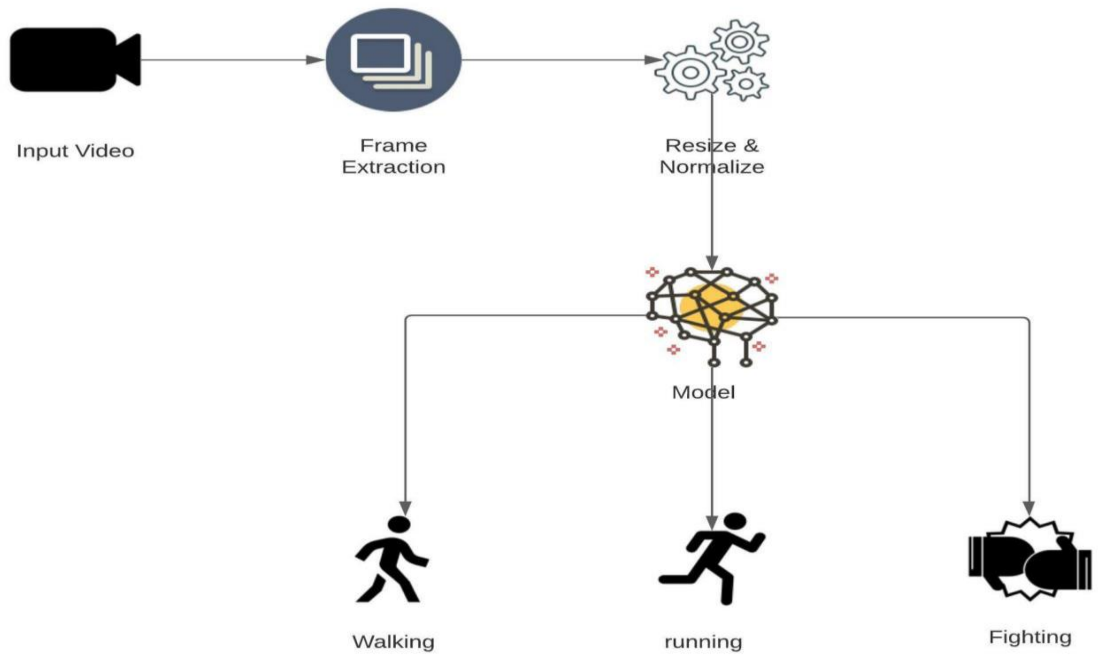


Image 2: The Project’s Flow

Model Construction

In suggested approach for suspicious Activity identification from video surveillance, LRCN is used as a deep learning network.

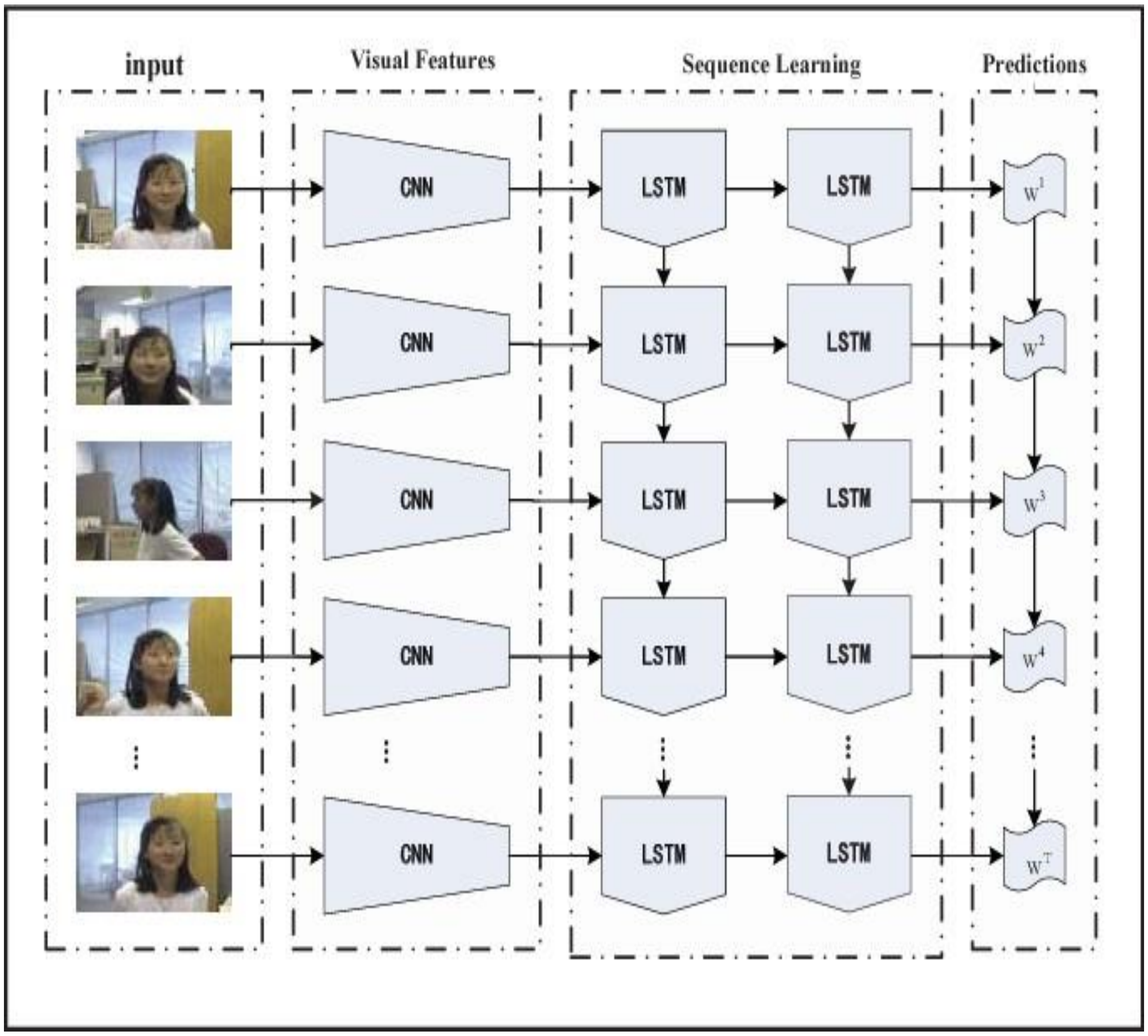


Image 3: LRCN Model

(Long-term Recurrent Convolutional Network) for detecting anomalous behavior

The LRCN model is shown in Figure along with the various scaling options that can be used to increase the accuracy of a deep learning model. These options include baseline and scaling like width, deep resolution, compound.

Primary principle of LRCN is to combine CNN model to learn visual characteristics from frames and LSTM to convert a series of images into a class label, phrase, and probability. As a result, unprocessed visual input is first sent through a CNN, then fed output to the recurrent sequence model [4].

In time series data, there can be random intervals between occurrences, and LSTM networks are suitable for making predictions, categorizing, and processing based on this type of data. LSTMs have addressed the gradient problem that can arise during the training of conventional RNNs.

STATE DIAGRAM

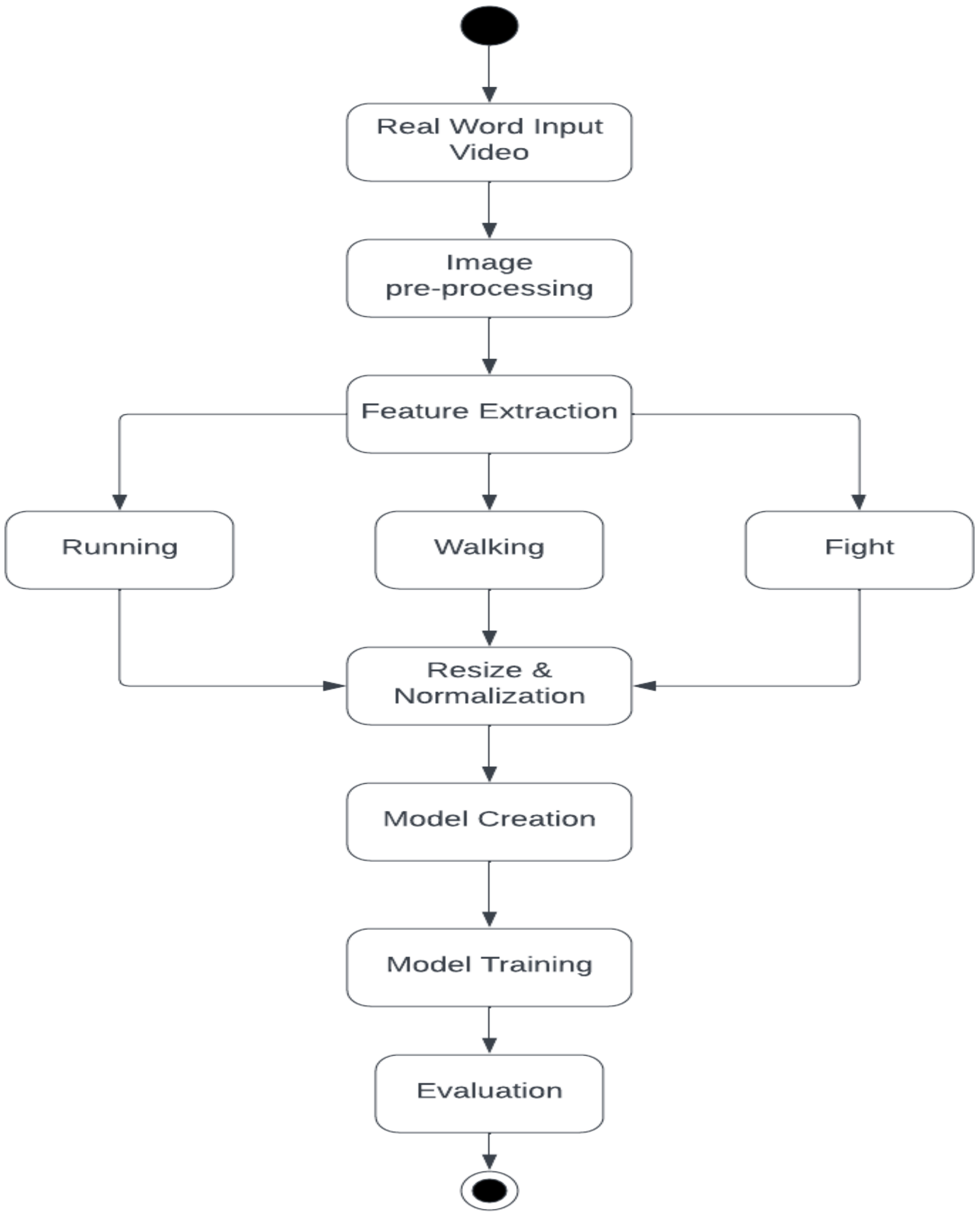


Image 4: State Diagram

<u>USECASE DIAGRAM</u>

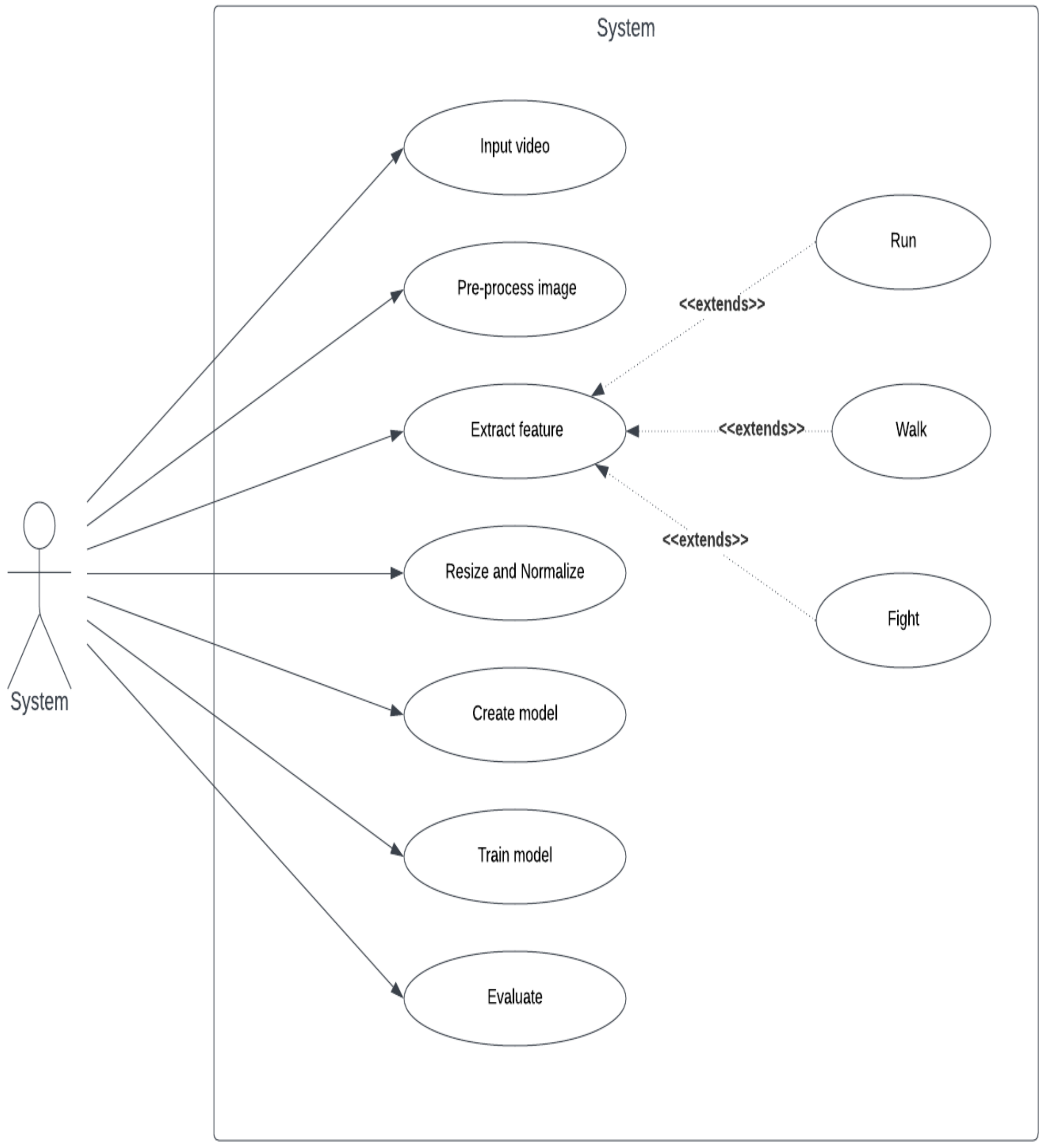


Image 5: Use-case Diagram

CLASS DIAGRAM

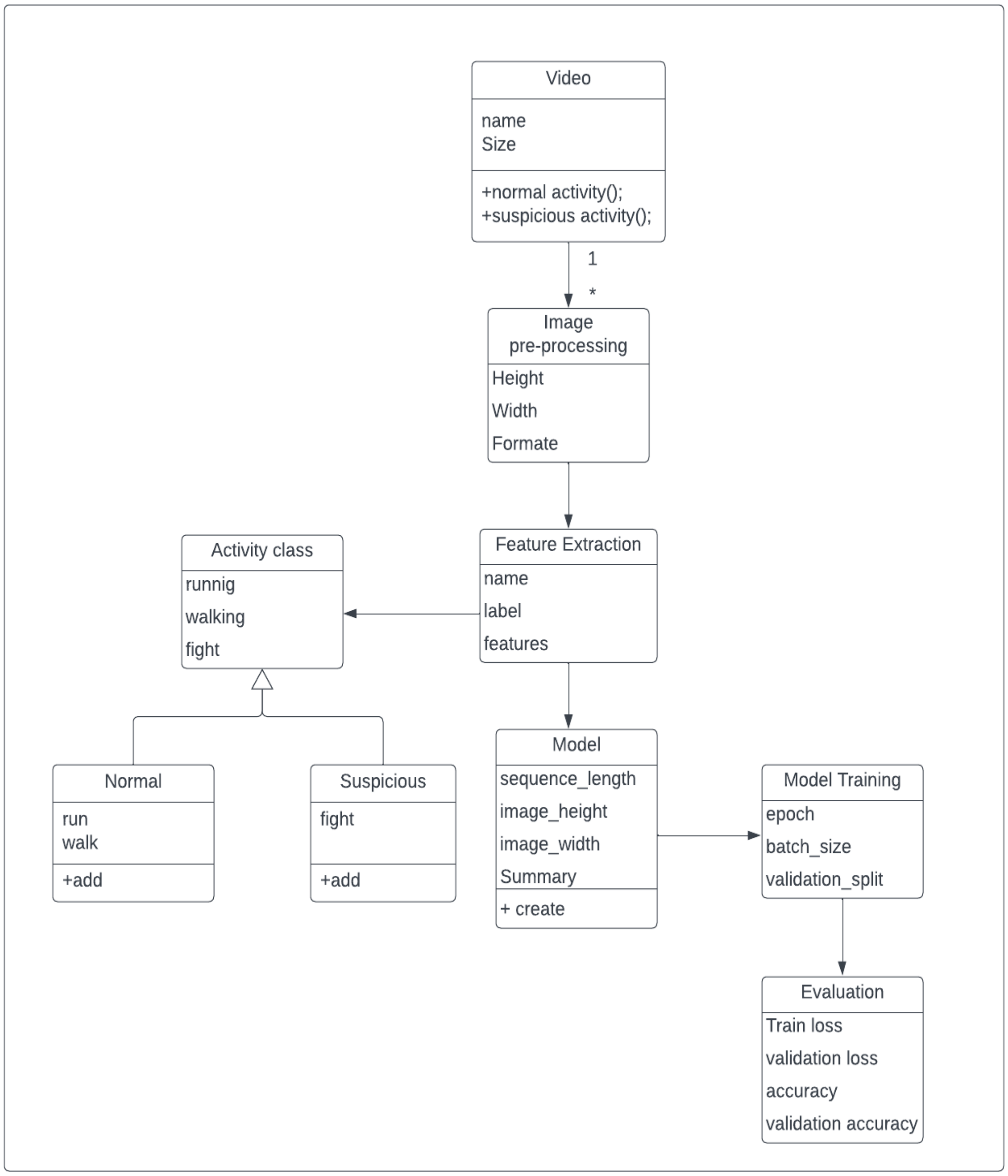


Image 6: Class Diagram

CHAPTER THREE

METHODS

Transformation of the Image

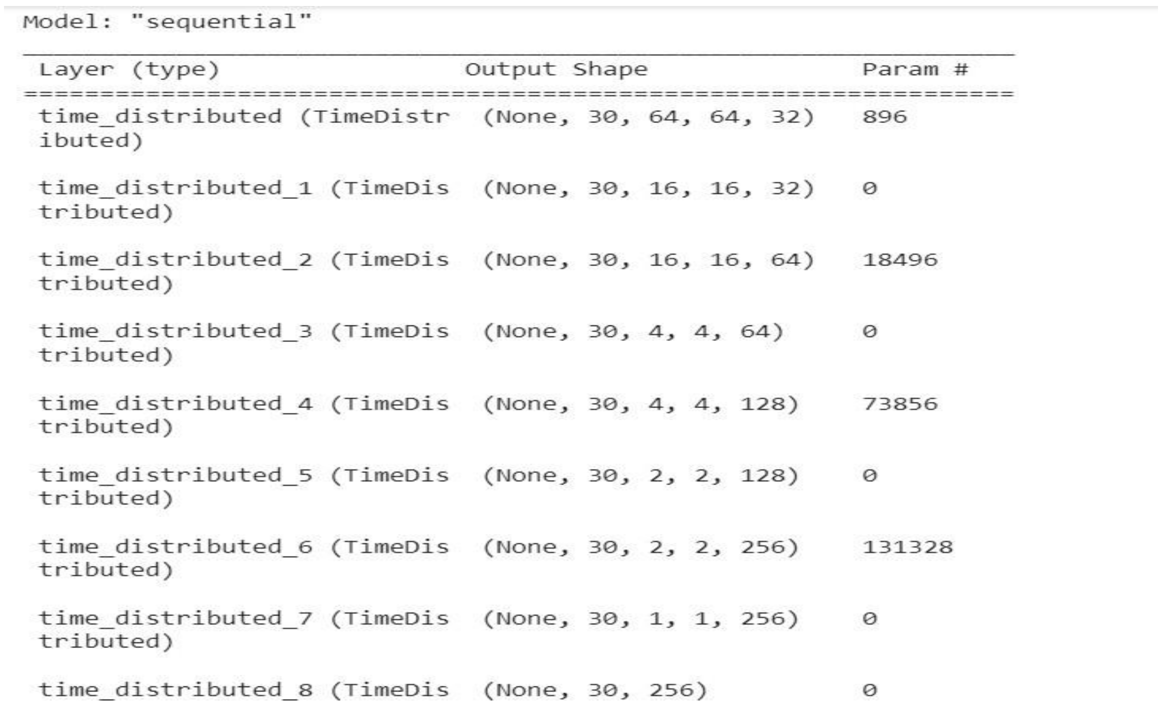
The initial frame should be read and converted from Blue-Green-Red (BGR) to Red-Geen-Blue (RGB). To change the RGB file format to BGR, we can utilize the OpenCV function cvtColor(). Each image was reduced in size by [64, 64], and [64, 64] also defined the duration of the sequence of 30 frames will be sent to the model as input data. Dataset transformations including resizing and normalization were utilized for the validation.

Image 7: Activity Images

LRCN Model

We employed a sequential model to construct the model. A sequential model consists stages that contains only single input and single output per stages.

model = create_LRCN_model()



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDistr  (None, 30, 64, 64, 32)   896
 ibuted)

 time_distributed_1 (TimeDis  (None, 30, 16, 16, 32)   0
 tributed)

 time_distributed_2 (TimeDis  (None, 30, 16, 16, 64)   18496
 tributed)

 time_distributed_3 (TimeDis  (None, 30, 4, 4, 64)     0
 tributed)

 time_distributed_4 (TimeDis  (None, 30, 4, 4, 128)    73856
 tributed)

 time_distributed_5 (TimeDis  (None, 30, 2, 2, 128)    0
 tributed)

 time_distributed_6 (TimeDis  (None, 30, 2, 2, 256)    131328
 tributed)

 time_distributed_7 (TimeDis  (None, 30, 1, 1, 256)    0
 tributed)

 time_distributed_8 (TimeDis  (None, 30, 256)          0
```

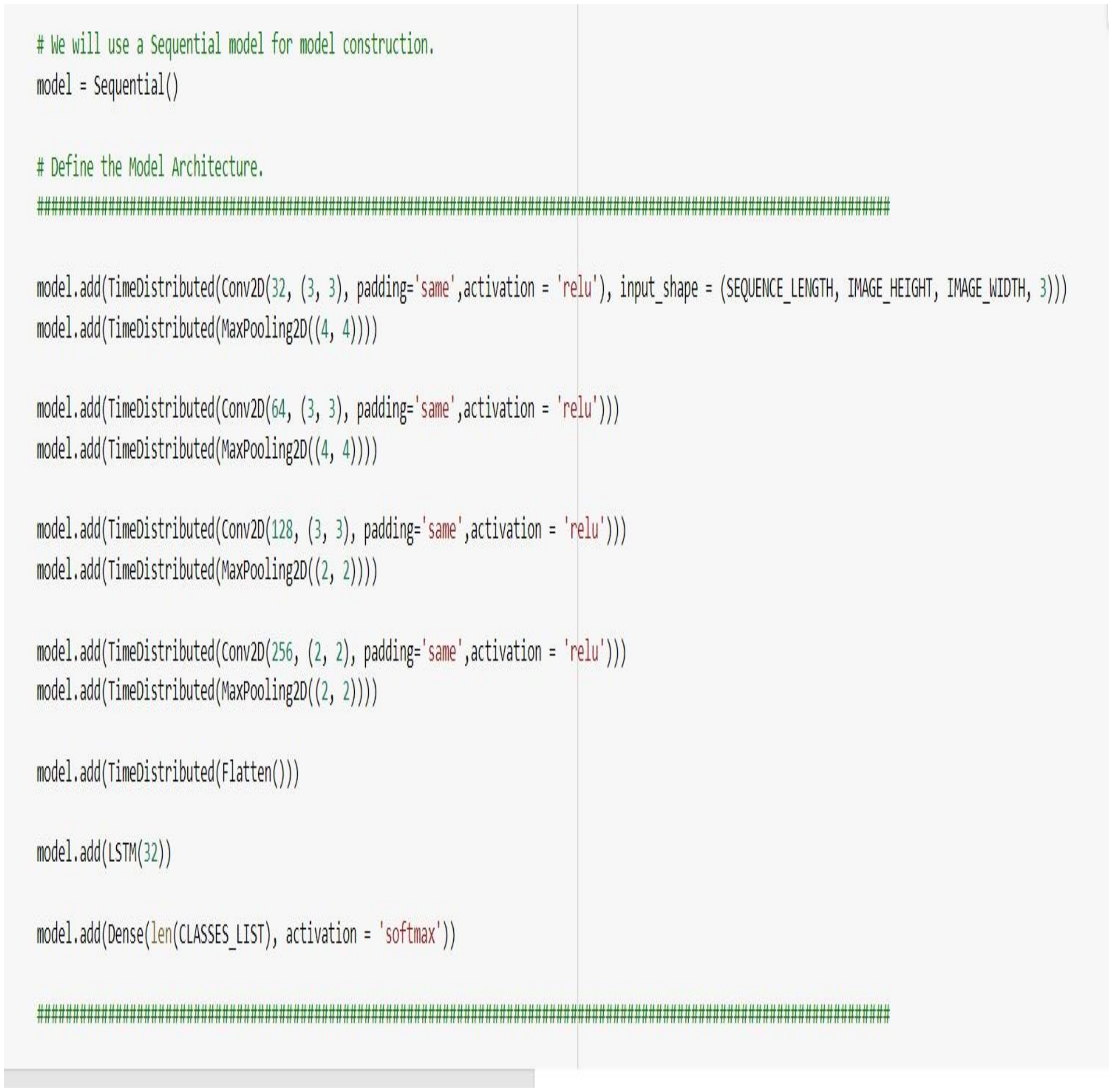
Image 8: Model Creation

When a sequential model is inappropriate:

- There are numerous inputs or outputs in the model.
- Sharing of layers is required.
- Requires a nonlinear topology.
- Every single one of your layers has various inputs and outputs.

Sequential Model Construction

The add () method use to incrementally build a sequential model.



```
# We will use a Sequential model for model construction.
model = Sequential()

# Define the Model Architecture.
########################################################################################################################

model.add(TimeDistributed(Conv2D(32, (3, 3), padding='same',activation = 'relu'), input_shape = (SEQUENCE_LENGTH, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))
model.add(TimeDistributed(MaxPooling2D((4, 4))))

model.add(TimeDistributed(Conv2D(64, (3, 3), padding='same',activation = 'relu')))
model.add(TimeDistributed(MaxPooling2D((4, 4))))

model.add(TimeDistributed(Conv2D(128, (3, 3), padding='same',activation = 'relu')))
model.add(TimeDistributed(MaxPooling2D((2, 2))))

model.add(TimeDistributed(Conv2D(256, (2, 2), padding='same',activation = 'relu')))
model.add(TimeDistributed(MaxPooling2D((2, 2))))

model.add(TimeDistributed(Flatten()))

model.add(LSTM(32))

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

########################################################################################################################
```

Image 9: Architecture

Stage Diagram

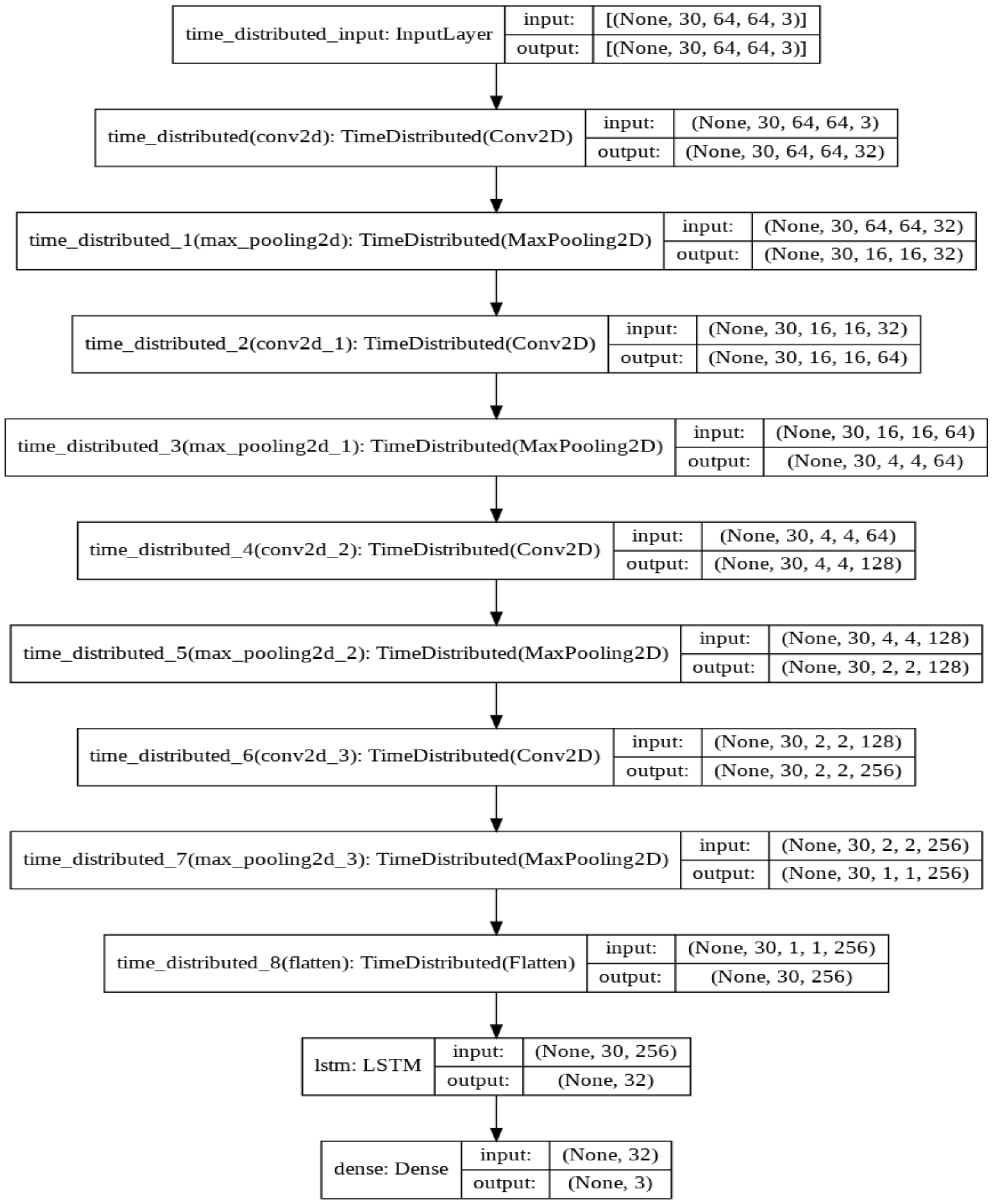
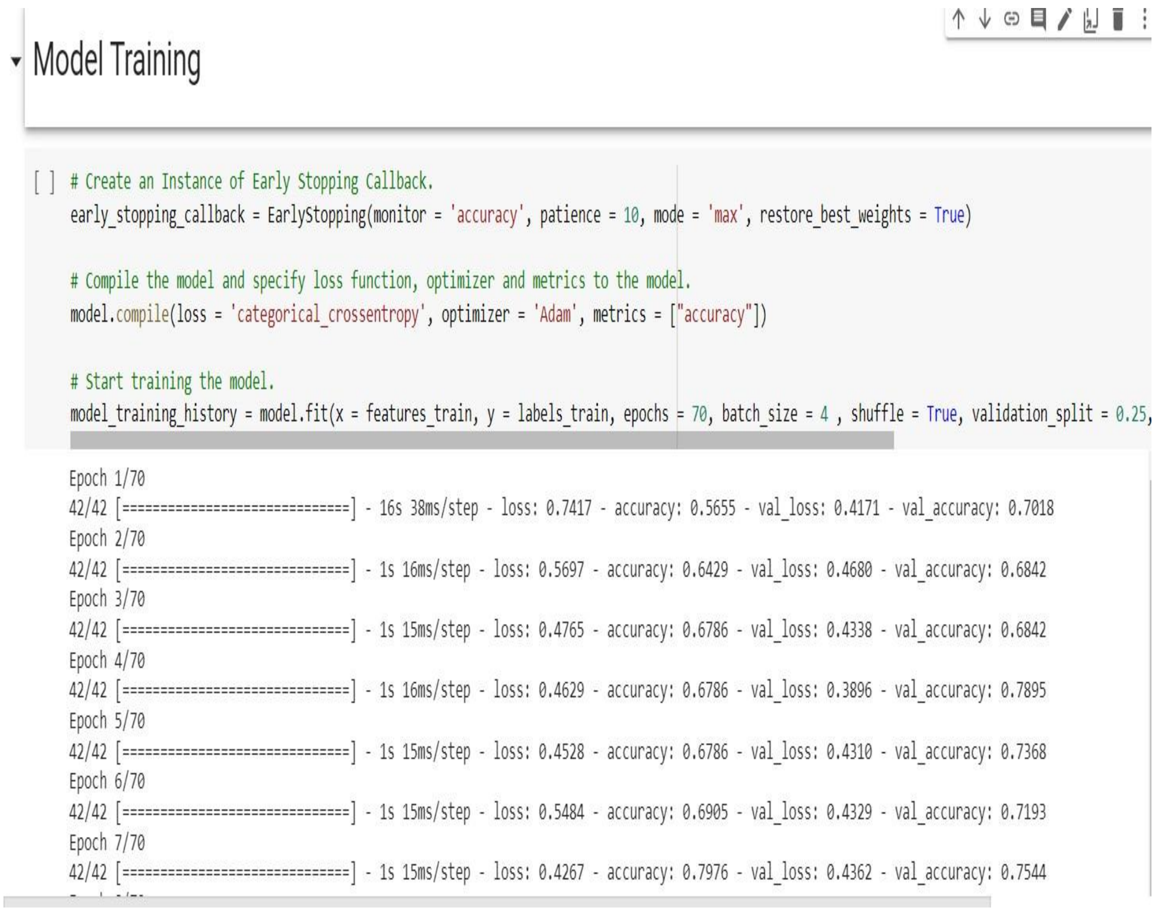

time_distributed_input: InputLayer | input: [(None, 30, 64, 64, 3)] | output: [(None, 30, 64, 64, 3)]
time_distributed(conv2d): TimeDistributed(Conv2D) | input: (None, 30, 64, 64, 3) | output: (None, 30, 64, 64, 32)
time_distributed_1(max_pooling2d): TimeDistributed(MaxPooling2D) | input: (None, 30, 64, 64, 32) | output: (None, 30, 16, 16, 32)
time_distributed_2(conv2d_1): TimeDistributed(Conv2D) | input: (None, 30, 16, 16, 32) | output: (None, 30, 16, 16, 64)
time_distributed_3(max_pooling2d_1): TimeDistributed(MaxPooling2D) | input: (None, 30, 16, 16, 64) | output: (None, 30, 4, 4, 64)
time_distributed_4(conv2d_2): TimeDistributed(Conv2D) | input: (None, 30, 4, 4, 64) | output: (None, 30, 4, 4, 128)
time_distributed_5(max_pooling2d_2): TimeDistributed(MaxPooling2D) | input: (None, 30, 4, 4, 128) | output: (None, 30, 2, 2, 128)
time_distributed_6(conv2d_3): TimeDistributed(Conv2D) | input: (None, 30, 2, 2, 128) | output: (None, 30, 2, 2, 256)
time_distributed_7(max_pooling2d_3): TimeDistributed(MaxPooling2D) | input: (None, 30, 2, 2, 256) | output: (None, 30, 1, 1, 256)
time_distributed_8(flatten): TimeDistributed(Flatten) | input: (None, 30, 1, 1, 256) | output: (None, 30, 256)
lstm: LSTM | input: (None, 30, 256) | output: (None, 32)
dense: Dense | input: (None, 32) | output: (None, 3)

Image 10: Stage Diagram

Train Model

Three human behaviors are taught to the model by passing different parameters.



Model Training

```
[ ] # Create an Instance of Early Stopping Callback.
    early_stopping_callback = EarlyStopping(monitor = 'accuracy', patience = 10, mode = 'max', restore_best_weights = True)

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

    # Start training the model.
    model_training_history = model.fit(x = features_train, y = labels_train, epochs = 70, batch_size = 4 , shuffle = True, validation_split = 0.25,
```

```
Epoch 1/70
42/42 [==============================] - 16s 38ms/step - loss: 0.7417 - accuracy: 0.5655 - val_loss: 0.4171 - val_accuracy: 0.7018
Epoch 2/70
42/42 [==============================] - 1s 16ms/step - loss: 0.5697 - accuracy: 0.6429 - val_loss: 0.4680 - val_accuracy: 0.6842
Epoch 3/70
42/42 [==============================] - 1s 15ms/step - loss: 0.4765 - accuracy: 0.6786 - val_loss: 0.4338 - val_accuracy: 0.6842
Epoch 4/70
42/42 [==============================] - 1s 16ms/step - loss: 0.4629 - accuracy: 0.6786 - val_loss: 0.3896 - val_accuracy: 0.7895
Epoch 5/70
42/42 [==============================] - 1s 15ms/step - loss: 0.4528 - accuracy: 0.6786 - val_loss: 0.4310 - val_accuracy: 0.7368
Epoch 6/70
42/42 [==============================] - 1s 15ms/step - loss: 0.5484 - accuracy: 0.6905 - val_loss: 0.4329 - val_accuracy: 0.7193
Epoch 7/70
42/42 [==============================] - 1s 15ms/step - loss: 0.4267 - accuracy: 0.7976 - val_loss: 0.4362 - val_accuracy: 0.7544
```

Image 11: Train model

Various Graphs:

```
# 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()
```

```
plot_metric(model_training_history, 'loss', 'val_loss', 'Total Loss vs Total Validation Loss')
```

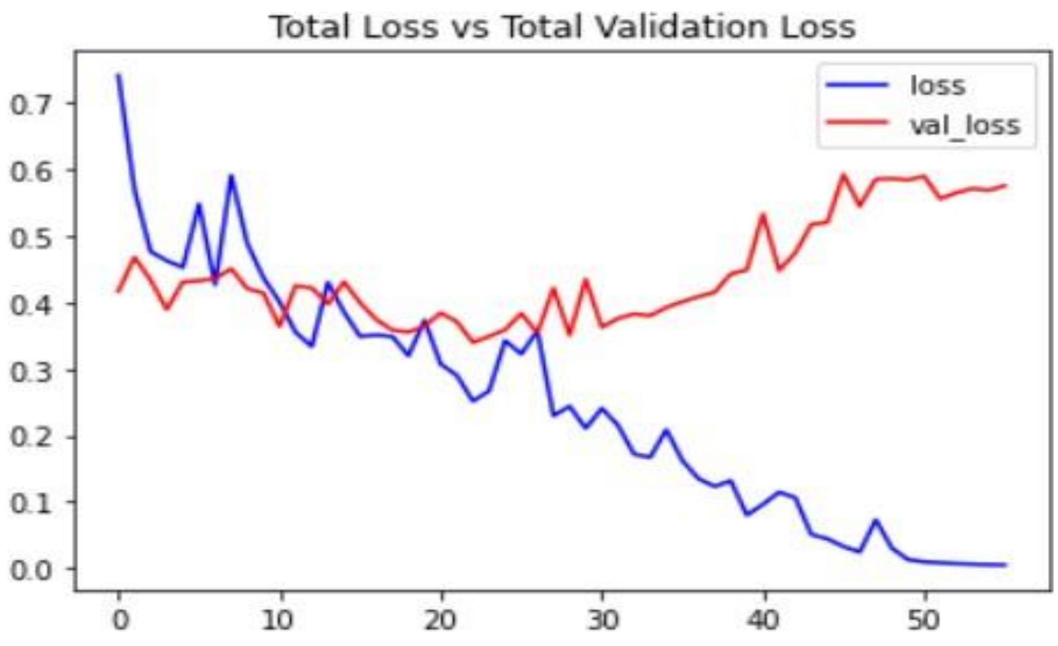
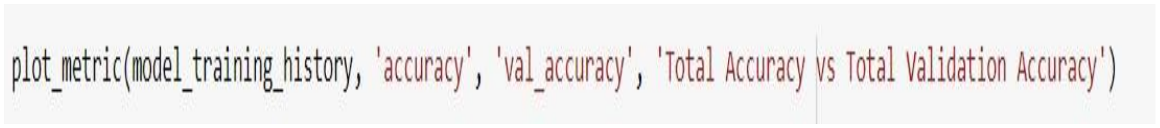


Image 12: Total Loss v/s Total Valid. Loss



```
plot_metric(model_training_history, 'accuracy', 'val_accuracy', 'Total Accuracy vs Total Validation Accuracy')
```

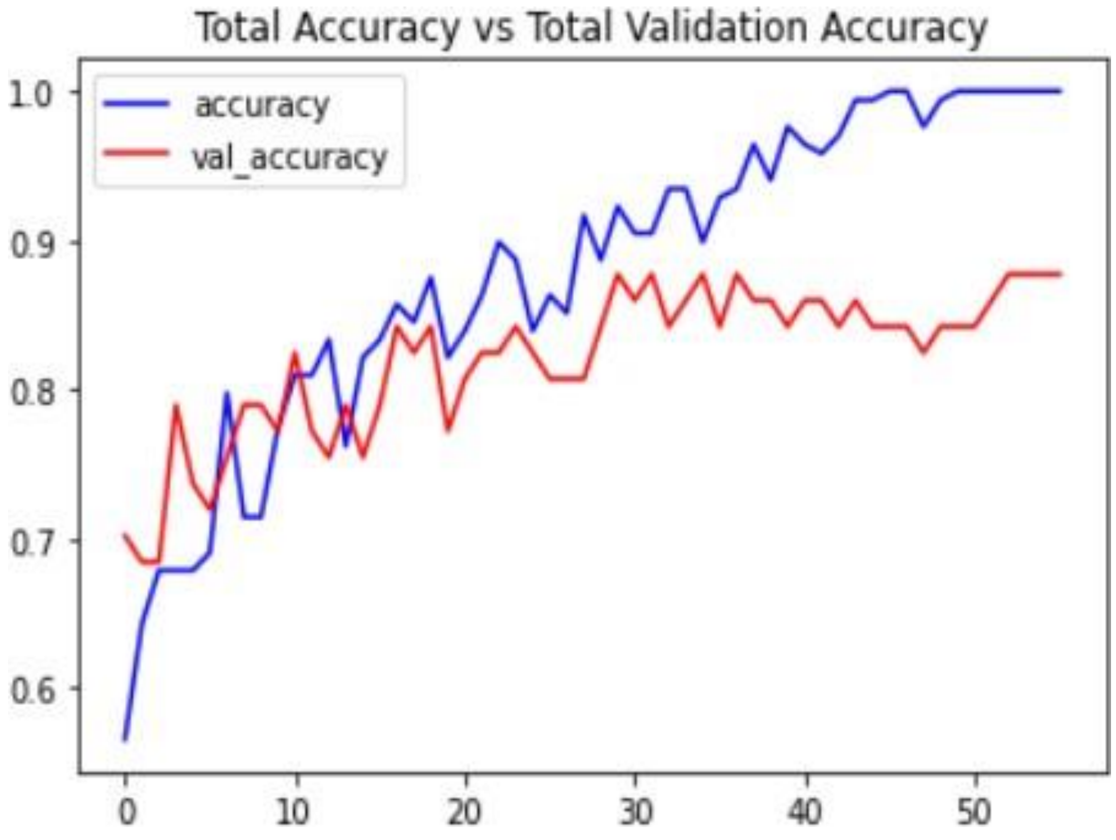


Image 13: Total Accuracy v/s Total Valid. Accuracy

Model Optimization

It is possible to maximize the length of the raw data and feedback using a variety of optimizers to produce better results. In this instance, Adam Optimizer is utilized to compress the data [2]. The Adam optimizer was employed in this case among other options like SGD and RMSProp. The extended stochastic gradient descent algorithm known as the Adam optimizer can be used in a variety of deep learning applications. The scheduler paused after numerous epochs and averaged the weights before using SWA. The SWA method was abandoned since it failed to provide any notable outcomes.

CHAPTER FOUR

RESULTS

According to our created data set, the model detects the suspicious behavior that is occurring in the video with an accuracy of 86%. The previous model utilized an LSTM architecture with 16 layers, which was time-consuming. However, the latest model reduced the stages from 16 to 12, making it faster and capable of real-time detection. Additionally, to save memory space, the frames were resized from 224px to 64px, and more videos were added to the dataset to improve accuracy. Videos depicting suspicious behavior, such fighting, as well as typical behavior, like walking and running, are both included in the dataset of the model.

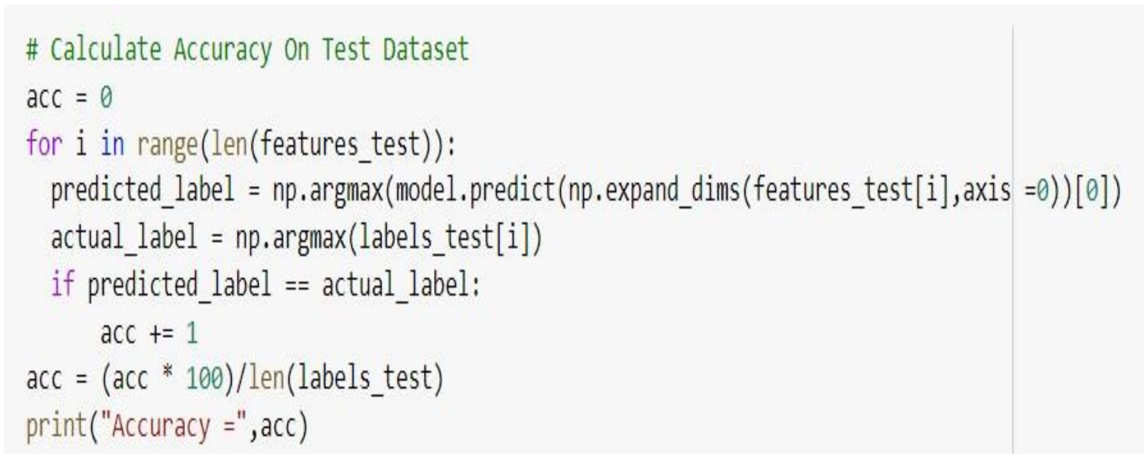
Train Loss

Through the use of the training data set as input, Train Loss measures the error of the created model. The digit produced by train loss provides look into the developed model result and how it broadcast the faulty prediction. Internally, each input's mistake is added up during the calculation. The epochs on the horizontal line and the error decided on the vertical line, image 9 is training error.

Validation Loss

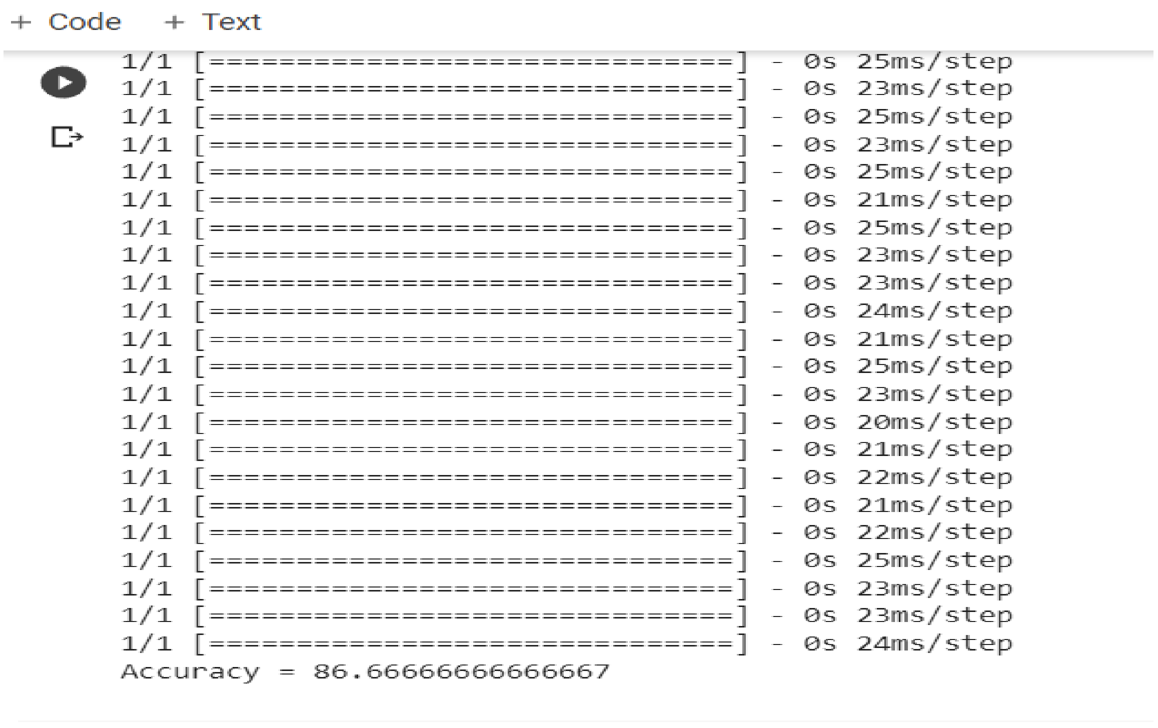
The definition of Valid Loss is the opposite of train loss. It is classified as loss, a statistic used to calculate how both learning model performed on the specified data set. The sum of each example's errors is used to calculate validation loss in a manner similar to how train loss is calculated. Image 9 displays the actual Loss.

Dataset Accuracy



```
# Calculate Accuracy On Test Dataset
acc = 0
for i in range(len(features_test)):
  predicted_label = np.argmax(model.predict(np.expand_dims(features_test[i],axis =0))[0])
  actual_label = np.argmax(labels_test[i])
  if predicted_label == actual_label:
      acc += 1
acc = (acc * 100)/len(labels_test)
print("Accuracy =",acc)
```

Accuracy: 86.6%



```
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
Accuracy = 86.66666666666667
```

Image 14: Test Dataset Accuracy

For 300 records of data, the model's accuracy of 86% is respectable.

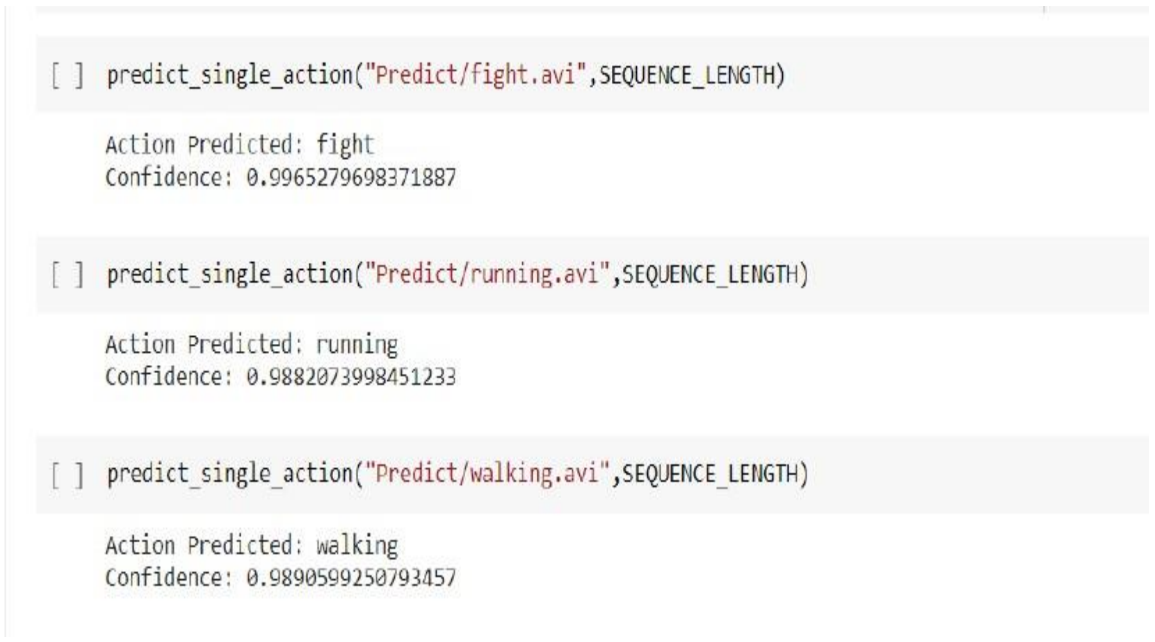
Train Accuracy

Train accuracy is the quantity of errors introduced while feeding the model with input training data. Accuracy is related to correctness. Technically, the model's accuracy might be inferred from the input data used to train it at the time it was constructed. The training accuracy is shown in image 10.

Validation Accuracy

The degree to which a method or model achieves what it measures with valid accuracy is specified. As an illustration, strong validity is measured in all study, which results in an assessment that yields true qualities. image 10 depicts a graph of the valid accuracy, with the vertical axis equal to the accuracy for each epoch and the horizontal axis showing the number of epochs.

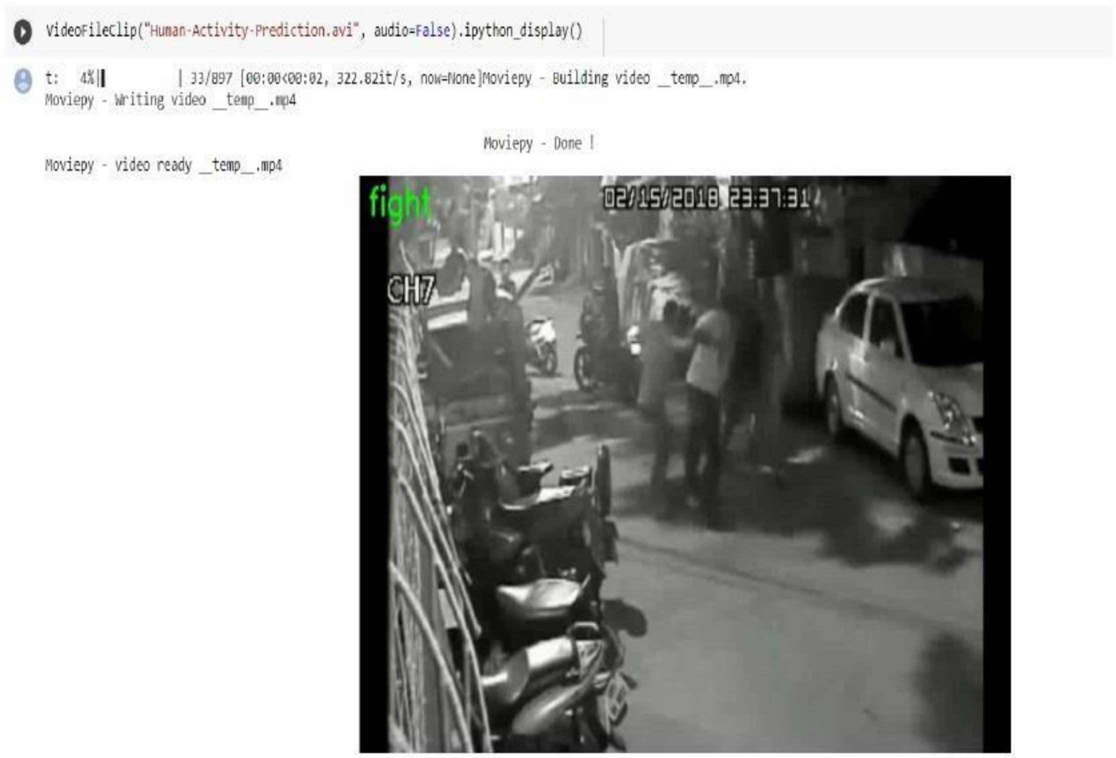
<u>Prediction of Specific Action</u>:



```
[ ]  predict_single_action("Predict/fight.avi",SEQUENCE_LENGTH)

     Action Predicted: fight
     Confidence: 0.9965279698371887

[ ]  predict_single_action("Predict/running.avi",SEQUENCE_LENGTH)

     Action Predicted: running
     Confidence: 0.9882073998451233

[ ]  predict_single_action("Predict/walking.avi",SEQUENCE_LENGTH)

     Action Predicted: walking
     Confidence: 0.9890599250793457
```

Image 15: Prediction of Specific Action

Activity - Fight:

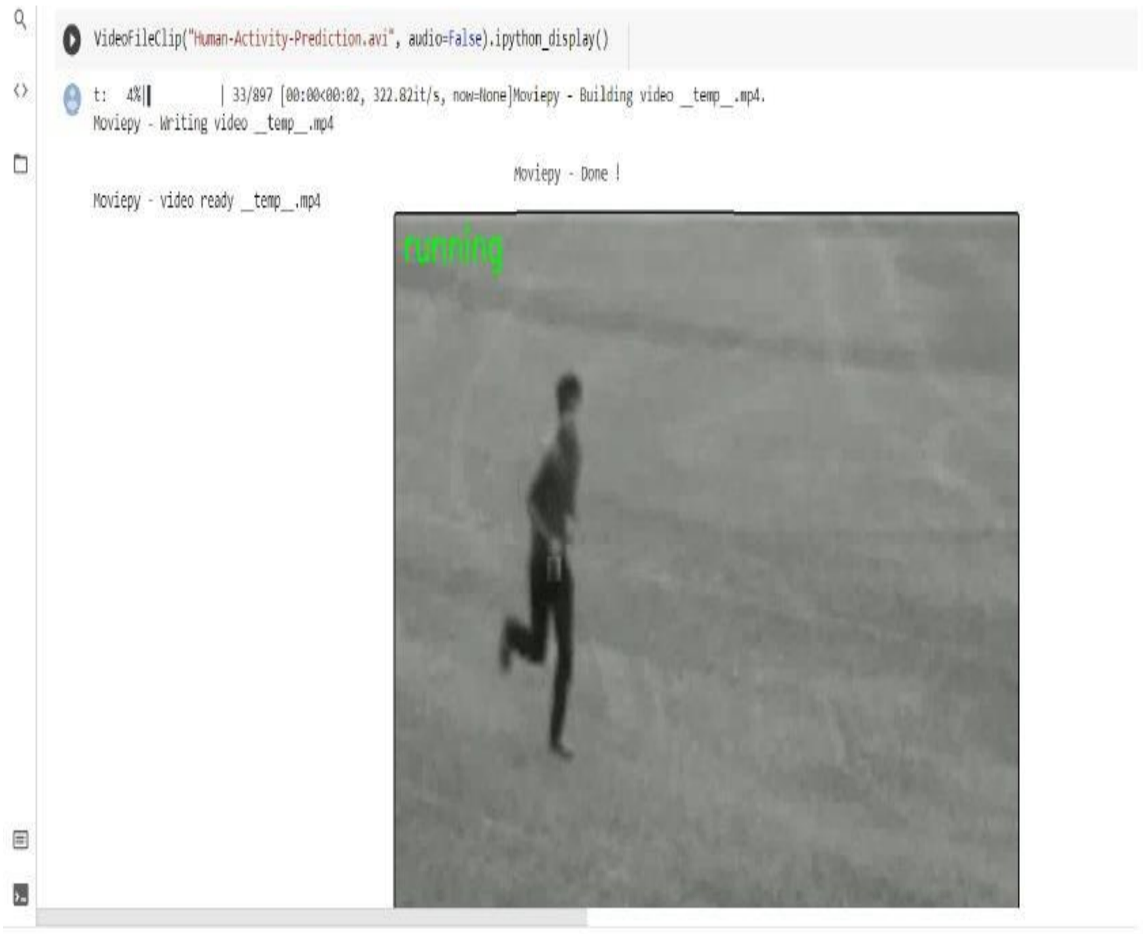


```
VideoFileClip("Human-Activity-Prediction.avi", audio=False).ipython_display()
```

```
t:   4%|▌         | 33/897 [00:00<00:02, 322.82it/s, now=None]Moviepy - Building video __temp__.mp4.
Moviepy - Writing video __temp__.mp4

                                                 Moviepy - Done !

Moviepy - video ready __temp__.mp4
```

Image 16: Fight

Activity - Running:

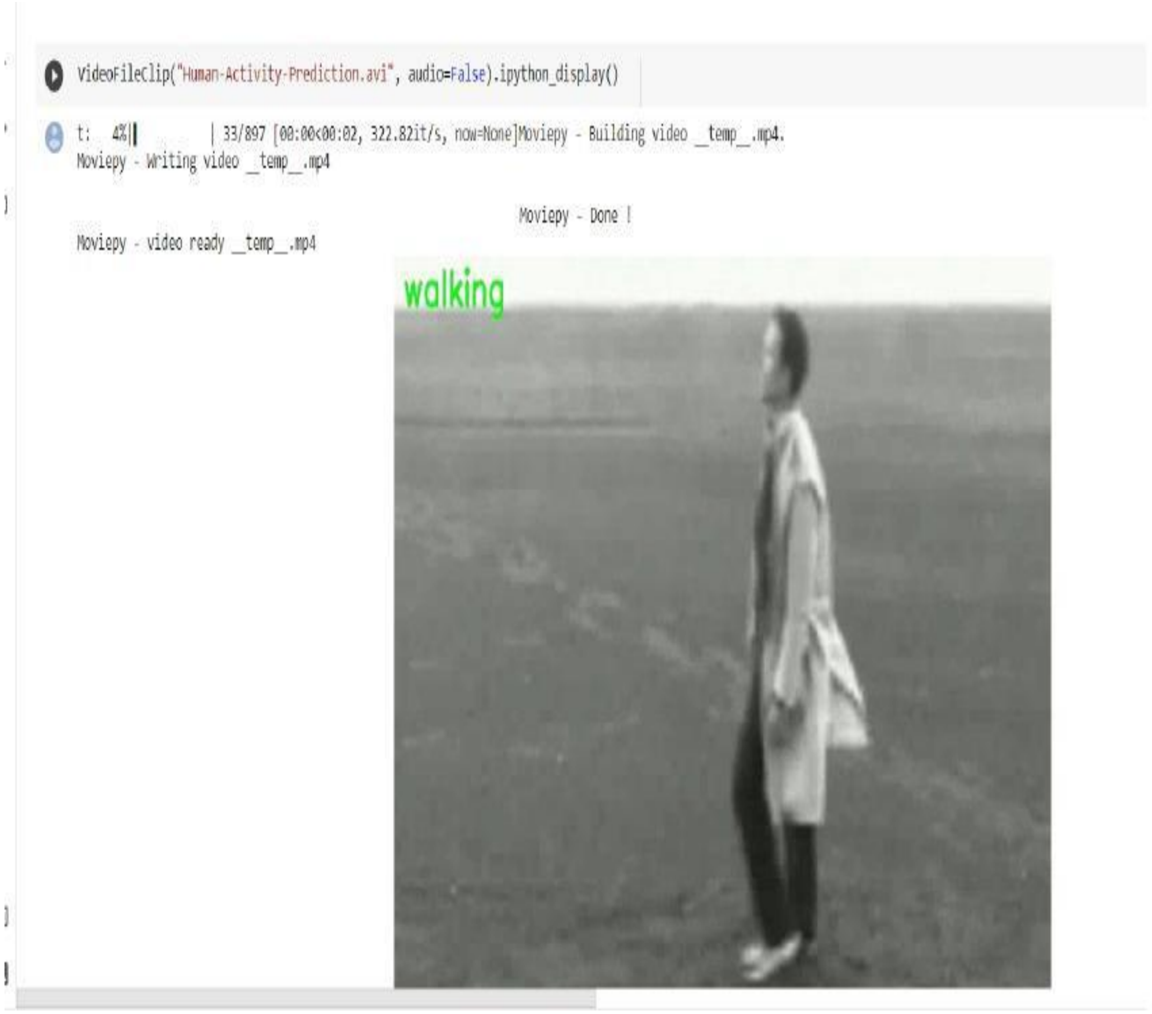


```
VideoFileClip("Human-Activity-Prediction.avi", audio=False).ipython_display()
```

```
t:   4%|▌         | 33/897 [00:00<00:02, 322.82it/s, now=None]Moviepy - Building video __temp__.mp4.
Moviepy - Writing video __temp__.mp4

                                          Moviepy - Done !
Moviepy - video ready __temp__.mp4
```

Image 17: Running

Activity - Walking:



```
VideoFileClip("Human-Activity-Prediction.avi", audio=False).ipython_display()
```

```
t:   4%|▌         | 33/897 [00:00<00:02, 322.82it/s, now=None]Moviepy - Building video __temp__.mp4.
Moviepy - Writing video __temp__.mp4

                                                      Moviepy - Done !
Moviepy - video ready __temp__.mp4
```

Image 18: Walking

Comparison and Result

Shortcomings from past model.

- Slow Speed - even with high-end equipment, its slower because of 16 phases.
- Delay in result - long time.
- There are 223 x 223 frame, so hard to load synchronousness frame array.

Improvements with new model

- Faster speed – Decrease stages from 16 to 12 phase, so it’s faster.
- Easy to gain better result.
- Resize frame to 64 x 64.

Model	Dataset	size	Result	Time
LSTM	45	223*223px	77%	No
LRCN	100	64*64px	86%	Yes

Image 19: Overall Summary

Conclusion

In research aims to automatically detect suspicious human activity using computer vision, machine learning, and deep learning. An LRCN model was developed and trained, it was discovered that the most recent models are effective in spotting abnormal behavior. The model achieved an accuracy of 86% by analyzing CCTV images to identify activities such as fighting, running, and walking. The model's efficiency and predictive accuracy can be enhanced by incorporating more relevant data.

APPENDIX A

KEY PART CODE

View the Dataset

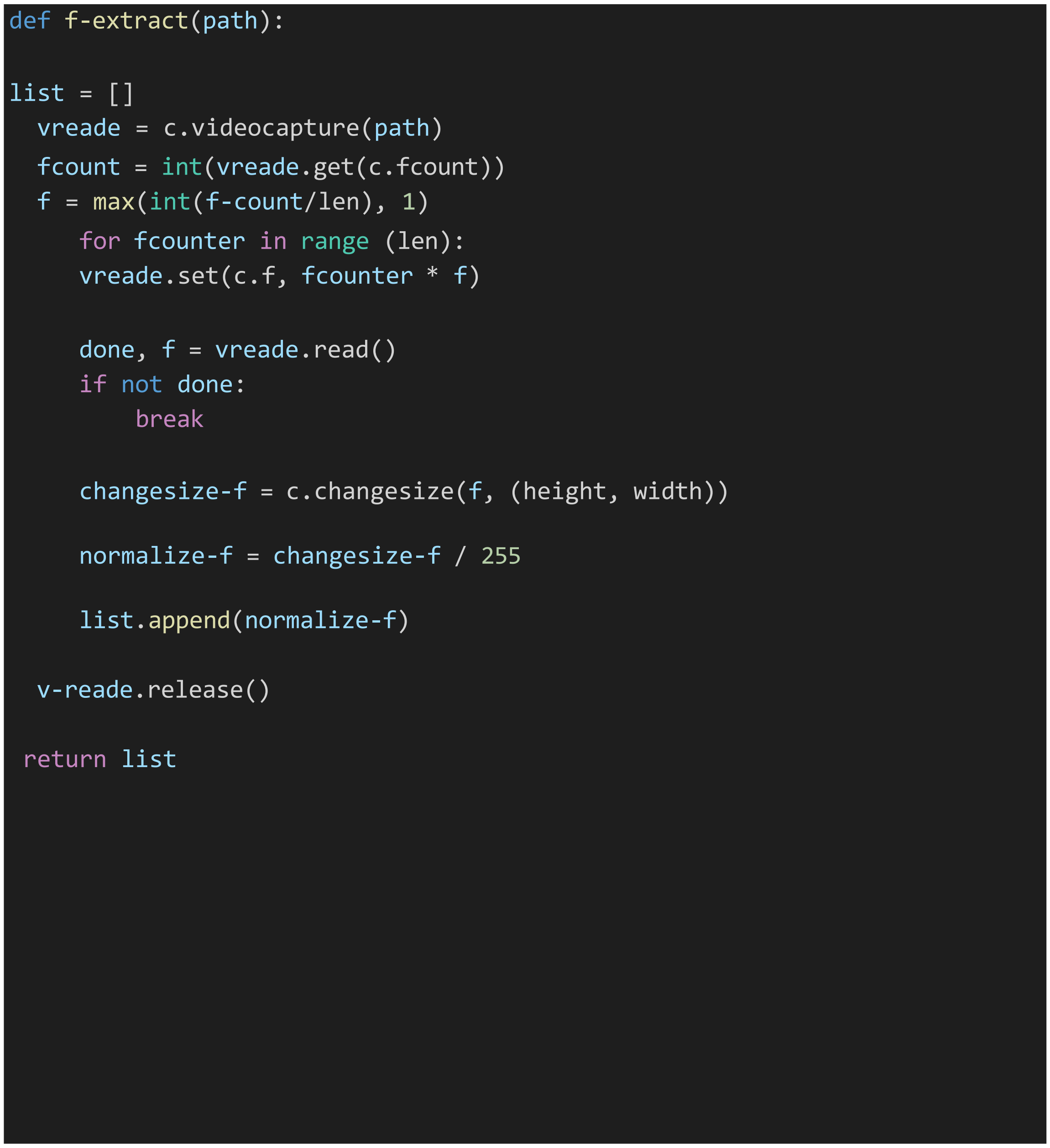
BGR is the image format used by OpenCV. Therefore, by default, cv2.imread() interprets images in BGR (blue, green, and red) format. However, in Pillow, it is believed that the colors are in RGB (red, green, and blue) sequence. So, in order to use the OpenCV function as well as the Pillow function, we must convert BGR and RGB. To recognize faces, objects, or even human handwriting, it can process photos and movies. When converting a BGR image to RGB or vice versa, we can utilize the cvtColor() method.

The class vise frames on display.

Data Pre-processing

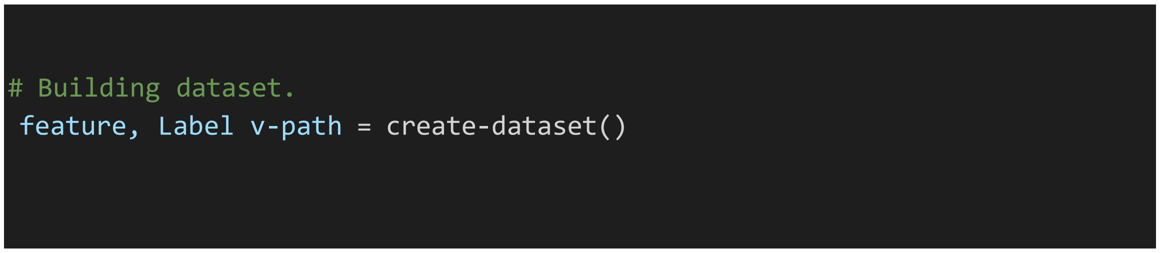
After resizing and normalizing the frames, function will extract from a video. video_path: The location on the disk of the video from which the desired frames should be taken. A list of the video's resized and normalized frames is called a "frame_list."

Data Pre-processing



```
def f-extract(path):

list = []
  vreade = c.videocapture(path)
  fcount = int(vreade.get(c.fcount))
  f = max(int(f-count/len), 1)
     for fcounter in range (len):
     vreade.set(c.f, fcounter * f)

     done, f = vreade.read()
     if not done:
         break

     changesize-f = c.changesize(f, (height, width))

     normalize-f = changesize-f / 255

     list.append(normalize-f)

  v-reade.release()

 return list
```

Building Class Category

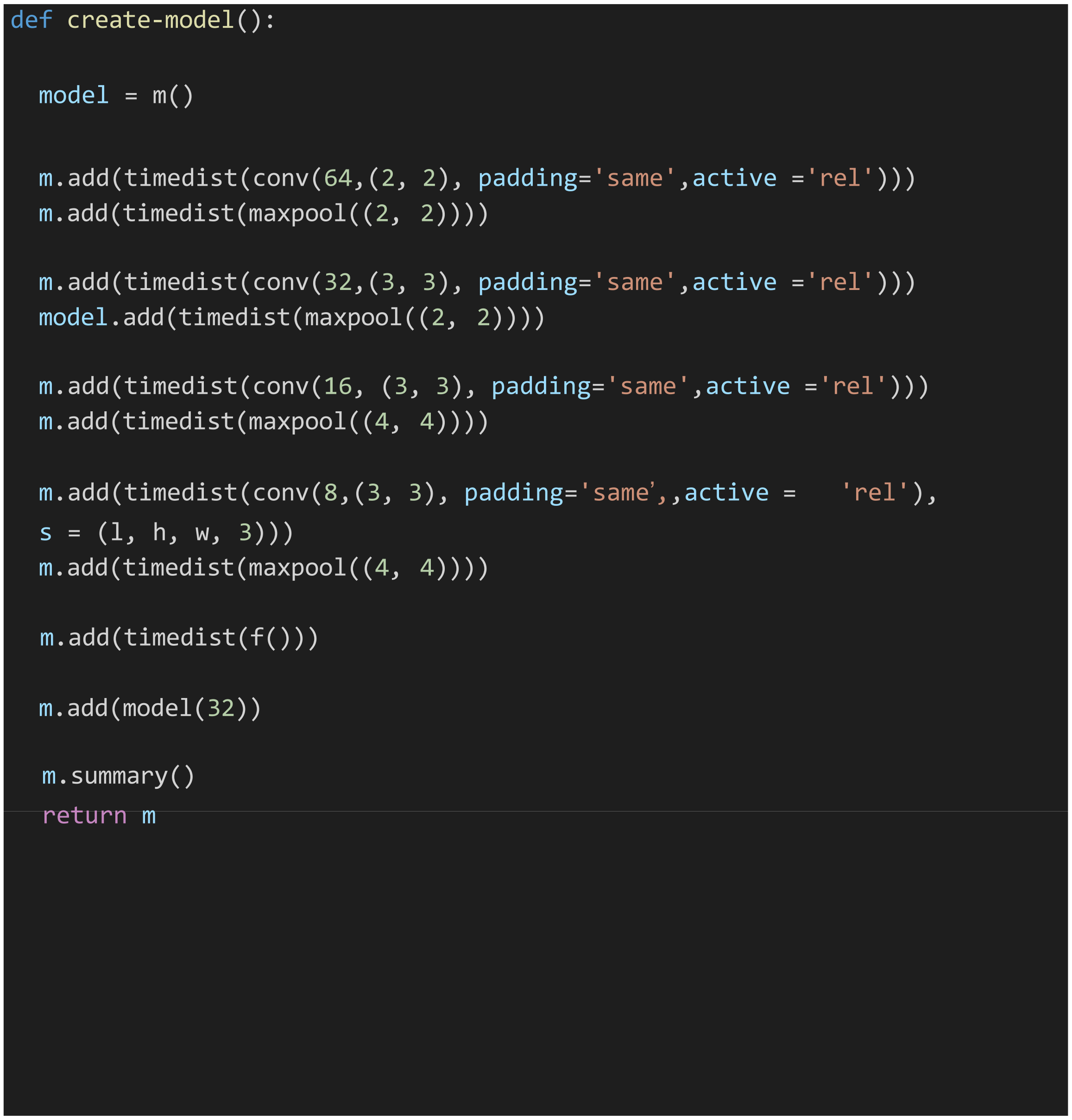


```
# Building dataset.
 feature, Label v-path = create-dataset()
```

Three classes: fight, running, and walking

Model Creation

We can build a Sequential model progressively using add () method. Add () can be used to incrementally stack layers, and printing model summaries periodically is helpful.



```
def create-model():

  model = m()

  m.add(timedist(conv(64,(2, 2), padding='same',active ='rel')))
  m.add(timedist(maxpool((2, 2))))

  m.add(timedist(conv(32,(3, 3), padding='same',active ='rel')))
  model.add(timedist(maxpool((2, 2))))

  m.add(timedist(conv(16, (3, 3), padding='same',active ='rel')))
  m.add(timedist(maxpool((4, 4))))

  m.add(timedist(conv(8,(3, 3), padding='same’,,active =   'rel'),
  s = (l, h, w, 3)))
  m.add(timedist(maxpool((4, 4))))

  m.add(timedist(f()))

  m.add(model(32))

  m.summary()
  return m
```

REFERENCES

[1] C. V. Amrutha, C. Jyotsna, J. Amudha (2020) Deep learning Approach for suspicious activity detection from surveillance video, Publisher IEEE Bangalore www.ieeexplore.ieee.org/document/9074920
(Original work published 2020)

[2] Jason Brownlee, Introduction to the adam-optimization-algorithm-for-deep-learning [python], Latest update on January 13, 2021.
www.machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning
(Original work published 2017)

[3] Mark Daoust (2022). Sequential model, Model Creation [python].TensorFlow is a platform that makes it easy to build and deploy ML models.
www.tensorflow.org/guide/keras/
(Original work published on 2022)

[4] Sik-Ho Tsang (2022) LRCN: Long-term Recurrent Convolution Networks[Python]
https://sh-tsang.medium.com/brief-review-lrcn
(Original work published on 2022.

[5] Dinesh Jackson, “suspicious activity detection in surveillance video using discriminative deep belief network,” International Journal of Control Theory and Applications, Volume 10, Number 29 - 2017.

[6] P. Bhagya Divya, Sravya Reddy, published an article in the International Research Journal of Engineering and Technology titled “Inspection of suspicious human activity in the crowdsourced areas captured in surveillance cameras”, December 2017.

[7] Elizabeth Scaria, Aby Abahai T and Elizabeth Isaac, “Suspicious Activity Detection in surveillance Video using Discriminative Deep Belief Netwok”, International Journal of control Theory and Applications Volume 10, Number 29 ---2017.