fight detection system

Using Deep Learning – LRCN MODEL

UNIVERSITY COLLEGE OF ENGINEERING

OSMANIA UNIVERSITY

*MINI PROJECT REPORT*

*BHANU**SUMANTH**DONTHALA*

*1005-20-733013*

*VAMSI**KRISHNA**DUSSA*

*1005-20-733014*

**FIGHT DETECTION SYSTEM USING**

**DEEP-LEARNING LRCN MODEL**

**MINI PROJECT REPORT**

*SUBMITTED BY:*

***BHANU SUMANTH DONTHALA* (100520733013)**

***VAMSI KRISHNA DUSSA* (100520733014)**

In partial fulfilment for the award of the degree

of

**BACHELOR OF ENGINEERING**

IN

**Computer Science and Engineering**

**University College of Engineering**

**OSMANIA UNIVERSITY: HYDERABAD-500007**

JULY-2023.



Department of Computer Science and Engineering,

University College of Engineering,

Osmania University.

**CERTIFICATE:**

This is to certify the bonafide work of **D. BHANU SUMANTH, VAMSI KRISHNA DUSSA** bearing roll numbers **100520733013,100520733014** respectively, for their mini project, in partial fulfilment of

Bachelor of Engineering Degree,

offered by

Department of Computer Science and Engineering, University College of Engineering, Osmania University.

page2image15451840

Project guide Head of the Department

(Asst. Prof) Prof. Dr. P.V. SUDHA A. Gayatri

Department of CSE, OU. Department of CSE, OU.

**STUDENT DECLARATION**

We declare that the work reported in the project report entitled “FIGHT DETECTION SYSTEM using Deep Learning” submitted by Donthala Bhanu Sumanth and Dussa Vamsi Krishna is a record of the work done by us in the DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, UNIVERSITY COLLEGE OF ENGINEERING, OSMANIA UNIVERSITY. No part of the report is copied from books/ journals/internet and wherever referred, the same has been duly acknowledged in the text. The reported data is based on the work done entirely by us and not copied from any other source or submitted to any other Institute or University for the award of a degree or diploma.

**Signatures**

Bhanu Sumanth:

Vamsi Krishna:

**Acknowledgement**

It is our privilege and pleasure to express our profound sense of respect, gratitude and indebtedness to our guide **A. Gayatri**, Assistant Professor, Department of Computer Science and Engineering, UCEOU, for her inspiration, guidance, cogent discussion, constructive criticisms, and encouragement throughout this dissertation work.

We also thank **Prof**.**Dr. P.V. SUDHA**, Head Department of Computer Science and Engineering for her support from the Department and allowing all the resources available to us students. We would also like to extend our thanks to the entire faculty of the Department of Computer Science and Engineering, University College of Engineering, Osmania University, who encouraged us throughout the course of our bachelor’s degree and allowing us to use as many resources present in the department. Our sincere thanks to our parents and friends for their valuable suggestions, morals, strength and support for the completion of our project.

**ABSTRACT**

A Fight Detection system is safety system designed to warn, alert, or assist drivers to avoid or reduce the severity of the imminent fights and reduce the occurrence of incidents. Fight Detection algorithms are essential for safe and efficient operation of autonomous policing structures. This work proposes using deep reinforcement (RL) learning as a framework to model the complex interactions and cooperation with nearby, decision-making agents, such as pedestrians.

Existing RL-based works assume homogeneity of agent properties, use specific motion models over short timescales, or lack a principled method to handle a large, possibly varying number of agents. Therefore, this work develops an algorithm that learns fight detection among a variety of heterogeneous, noncommunicating, dynamic agents without assuming they follow any particular behaviour rules. This project introduces a strategy using Long Short-Term Memory (LSTM) that enables the algorithm to use observations of an arbitrary number of other agents, instead of a small, fixed number of neighbours. The proposed algorithm is shown to work in an environment with dynamic number of agents.

The training process involves creating a simulated environment where the agent interacts with virtual people, learning from both successful and unsuccessful fight scenarios. By applying the principles of reinforcement learning, the agent gradually improves its decision-making capabilities through trial and error, optimizing its behaviour to minimize false alarms.

The findings of this study have significant implications for the development of intelligent collision fight detection systems in real-world applications, such as security cameras, university campuses, and public transportation. By leveraging deep reinforcement learning, our approach offers a promising avenue for enhancing people safety and enabling smooth and efficient policing in crowded urban environments.

In brief, the system is designed to outperform previous approaches as the datasets increase.

**Table of Contents**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S.no** | | | **Content** | | **P.no.** | |
| **1.** | | | **Chapter 1:** Introduction   * 1. Aim   2. LRCN   3. CONVNET   4. LSTM | |  | |
| **2.** | | | **Chapter2:** Literary Survey | |  | |
| **3.** | | | **Chapter 3:** Existing Methods | |  | |
| **4.** | | | **Chapter 4:** Proposed Solutions  4.1Proposed Approach  4.2 Approach | |  | |
| **5.** | | | **Chapter 5:** Implementation  5.1 Environmental Setup  5.2 Training the Data Setup  5.3 Code Execution | |  | |
| **6.** | **Chapter 6:** Results  6.1 Accuracy Curves  6.2 Loss curves  6.3 Testing and Optimization  6.4 Testing on existing Datasets  6.5 Testing on outside Datasets | |  | |
| **7.** | **Chapter 7:** Conclusion and Future Scope  7.1 Conclusion  7.2 Future Scope | |  | |
| **8.** | **References** | |  | |
| **9.** | **Appendix** | |  | |

**CHAPTER 1**

**INTRODUCTION**

* 1. **AIM**

*Human behaviour recognition* in the real-world environment finds plenty of applications including intelligent video surveillance, shopping behaviour analysis. Video surveillance has vast application areas especially for indoor outdoor and places. Surveillance is an integral part of security. Today security camera becomes part of life for the safety and security purposes.

Today, manual monitoring of all the events on the *CCTV (Closed Circuit Television)* camera is impossible. Even if the event had already happened, searching manually the same event in the recorded video wastes a lot of time. Analysing abnormal events from video is an emerging topic in the domain of *automated video surveillance systems*.

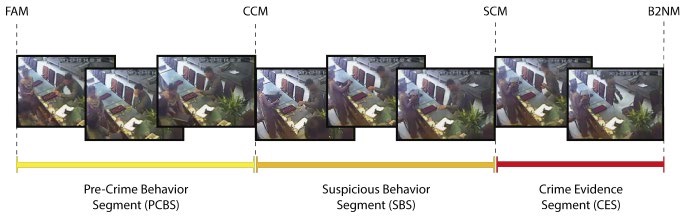
Human behaviour detection in video surveillance system is an automated way of intelligently detecting any suspicious activity. Number of efficient algorithms is available for the automatic detection of human behaviour in public areas like airports, railway stations, banks, offices, examination halls etc

*Video surveillance* is the emerging area in the application of *Artificial Intelligence, Machine Learning and Deep Learning*. Artificial intelligence helps the computer to think like human. In machine learning, important components are learning from the training data and make prediction on future data. Nowadays *GPU (Graphics Processing Unit)* processors and huge datasets are available, so the concept of deep learning is used.

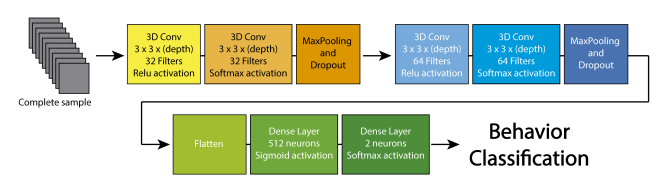
*Deep Neural Networks* is one of the best architectures used to perform difficult learning tasks. Deep Learning models automatically extract features and builds high level representation of image data. This is more generic because the process of feature extraction is fully automated. From the image pixels, *convolutional neural network (CNN)* can learn visual patterns directly. In the case of video stream, *long short-term memory (LSTM)* models are capable of learning long term dependencies. LSTM network has the ability to remember things.

The proposed system will use footage obtained from CCTV camera for monitoring the human behaviour in a campus and gently warn when any suspicious event occurs. The major components in intelligent video monitoring are event detection and human behaviour recognition.

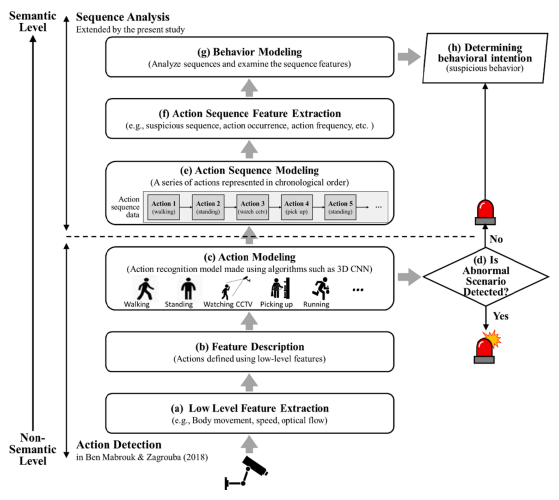
The entire process of training a surveillance system can be summarized in to three phases: data preparation, training the model and inference.



***Figure 1 Video segmentation by using the moments obtained from the Pre-Crime Behaviour Segment (PCB) method.***



***Figure 2 Architecture of the DL Model used.***



**CHAPTER 2**

LITERARY SURVEY

The related works suggests different approaches for detecting human behaviors from video. The objective of the works was to detect any abnormal or suspicious events in a video surveillance.

[1] *Advance Motion Detection (AMD)* algorithm was used to detect an unauthorized entry in a restricted area. In the first phase, the object was detected using background subtraction and from frame sequences the object is extracted. The second phase was detection of suspicious activity. Advantage of the system was the algorithm works on real time video processing and its computational complexity was low. But the system was limited in terms of storage service and it can also be implemented with hi-tech mode of capturing of videos in the surveillance areas.

[2] A semantic based approach was proposed in [References- [2]]. The captured video data was processed and the foreground objects were identified using background subtraction. After subtraction, the objects are classified into living or non-living using Haar like algorithm. Objects tracking were done using Real-Time blob matching algorithm. Fire detection was also detected in this paper.

[3] The unusual events in video footage could be detected by tracking of people. Human beings are detected from the video using background subtraction method. The features are extracted using CNN and which was fed to a *DDBN (Discriminative Deep Belief Network)*. Labeled videos of some suspicious events are also fed to the DDBN and their features are also extracted. Then a comparison of features extracted using CNN and features extracted from the labeled sample video of classified suspicious actions was done using a DDBN and various suspicious activities are detected from the given video.

[4] A real time violence detection system using deep learning was developed to prevent the violence behavior of crowd or players in sports. In a spark environment, frames were extracted from real-time videos. If the system detects any violence in football, then alert the security people. To prevent the violence in advance, the system detects the video actions in real time and alerts the security forces. *VID dataset* was used and achieved an accuracy of 94.5% for detecting violence in football stadium.

**CHAPTER 3**

**EXISTING METHODS**

**1. ANAMOULY DETECTION**

**. Statistical Methods:** Algorithms like Z-score, Modified Z-score and Gaussian mixture Models(GMM) can identify data points and simultaneously detect the expected statistical derivation.

. **Isolation Forest:** This algorithm isolates anomalies by recursively partitioning the data until anomalies are isolated into individual trees.

. **One-Class SVM:** Support Vector Machines (SVMs) trained only on normal data can identify anomalies as points that fall outside the SVM's decision boundary.

**2.Machine Learning Classification:**

.**Supervised Learning**: Classification algorithms like Random Forest, Support Vector Machines, and Neural Networks can be trained to classify activities as normal or suspicious based on labeled data.

.**Semi-supervised Learning**: Combining labeled and unlabeled data can help in training models to identify anomalies and suspicious activities.

**3.Clustering Algorithms:**

. **K-Means:** Clustering techniques can group similar data points together. Unusual or isolated clusters could indicate suspicious behavior.

. **DBSCAN:** Density-based clustering can identify dense clusters of data points, with outliers potentially being indicative of suspicious activities.

**4.Graph-based Approaches:**

**. Social Network Analysis**: For scenarios involving interactions between entities, analyzing the structure and patterns in a social network can reveal unusual connections or behaviors.

. **Graph Anomaly Detection:** Detecting anomalies in graph-based data, such as identifying nodes with abnormal connections.

**5.Reinforcement Learning:**

. **Markov Decision Processes (MDPs**): In dynamic environments, reinforcement learning can model the agent's actions and rewards to learn optimal strategies and detect deviations.

**7.Fusion of Multiple Modalities:**

Combining data from multiple sources like video, audio, sensor data, and text can enhance the system's ability to detect suspicious activities by considering a broader context.

**8.Deep Learning:**

. **Recurrent Neural Networks (RNNs):** RNNs can model temporal sequences and identify deviations from learned patterns.

. **Convolutional Neural Networks (CNNs)**: CNNs can process visual data and identify anomalies in images or video frames.

. **Graph Neural Networks (GNNs):** GNNs can capture relationships between entities in graph data and detect unusual patterns.

**ETC ARE A FEW APPROACHES THAT ARE CONVENTIONALLY AVAILABLE**

**FEW OF THEM WERE TRIED BUT WERE NEGLECTED DUE TO POOR ACCURACY.**

**CHAPTER 4**

**PROPOSED SOLUTIONS**

**4.1 PROPOSED APPROACH**

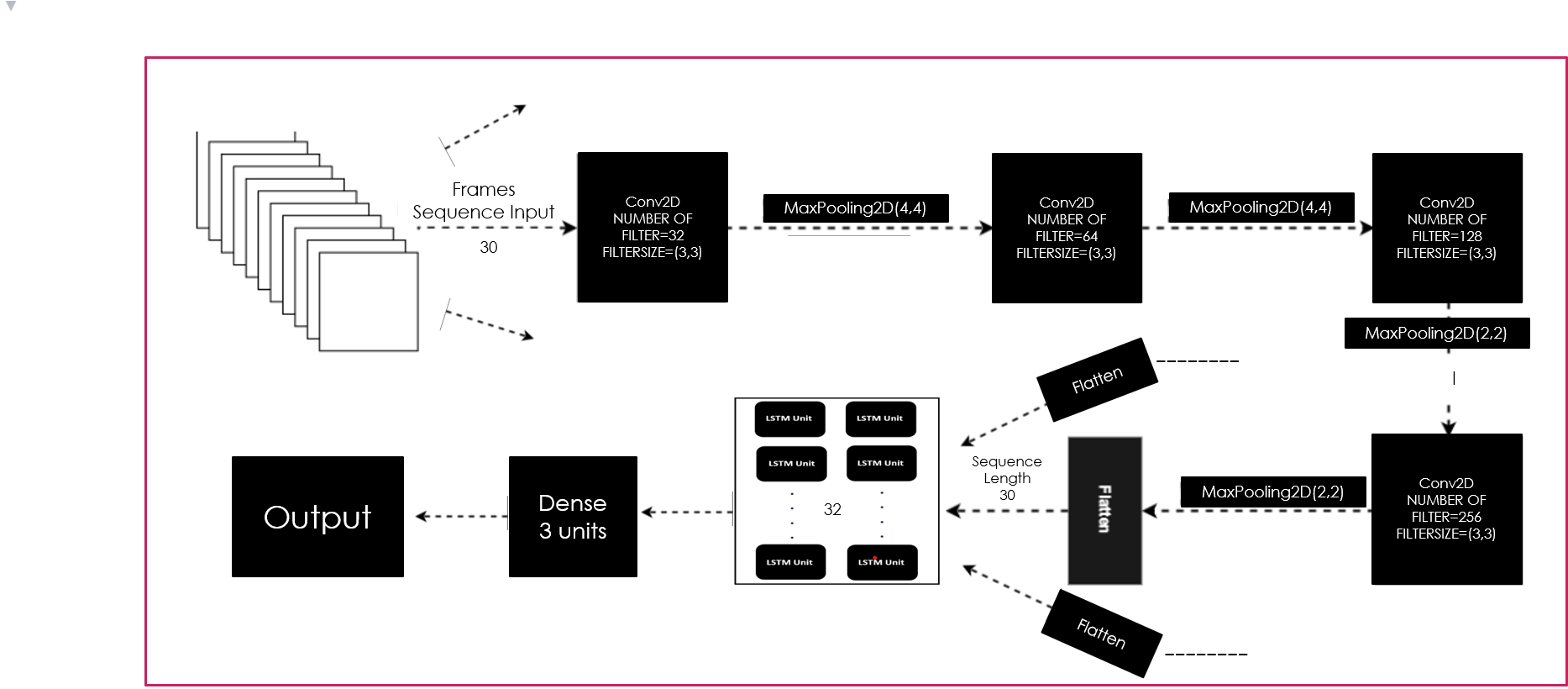
**LRCN APPROACH –** LONG-TERM RECURRENT CONVOLUTIONAL NETWORKS

The neural network that was used during the development stage includes two stages, the first includes extracting properties from the frames contained in the visual clips and passing those characteristics to the recurrent neural network layer and the long-term memory LSTM. The recurrent neural network and long-term memory are in the form of a one-dimensional array, where the long-term memory and the recurrent neural network study the relationship of the sequence of pixel values of each frame of the sequence of frames contained in the visual clips contained in the dataset used.

In this part, we will review the structure of the neural network that was used to study movement, which is a mixture of the convolutional neural network and the recurrent neural network, where we will review the proposed structure gradually.

***Convolutional Neural Network****:*

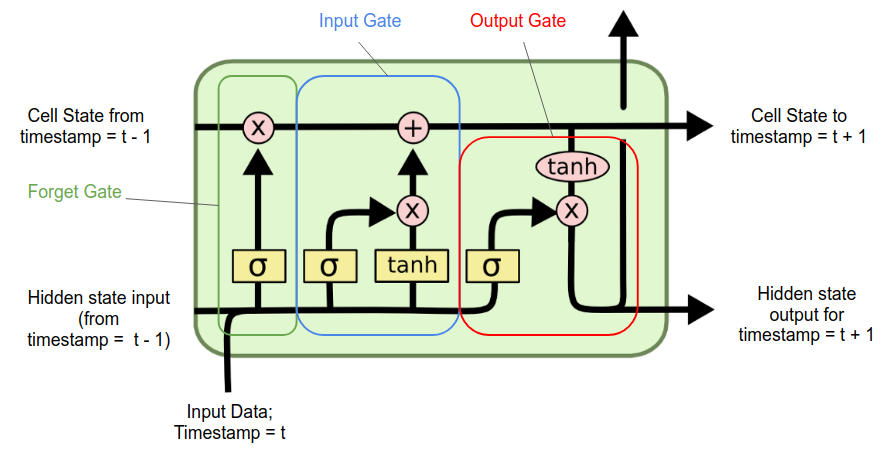
The convolutional neural network used to study the characteristics of each frame of the sequence of frames contained in each visible segment of the dataset consists of several layers in addition to determining the characteristics of each of the layers that are used, the following figure shows the structure of the neural network the convolution that was used to identify the properties contained in the frame.



***The architecture of a convolutional neural network that was used to extract properties from the frames.***

***Long Short-term Memory:***

We proposed the use of long-term memory to study the sequence of pixel values of the frames contained in each video clip. Gates, through which long time sequences can be studied, and since we use a sequence of frames up to analyse the movement resulting from the sequence of that number of frames, we suggested the use of long-term memory in studying the characteristics that were extracted from the CNN layers and thus trying to link those characteristics to each other To identify and sort the movement.



***A single LSTM Cell***

**Forget Gate:** Its task is to determine which information is ignored by the unit.

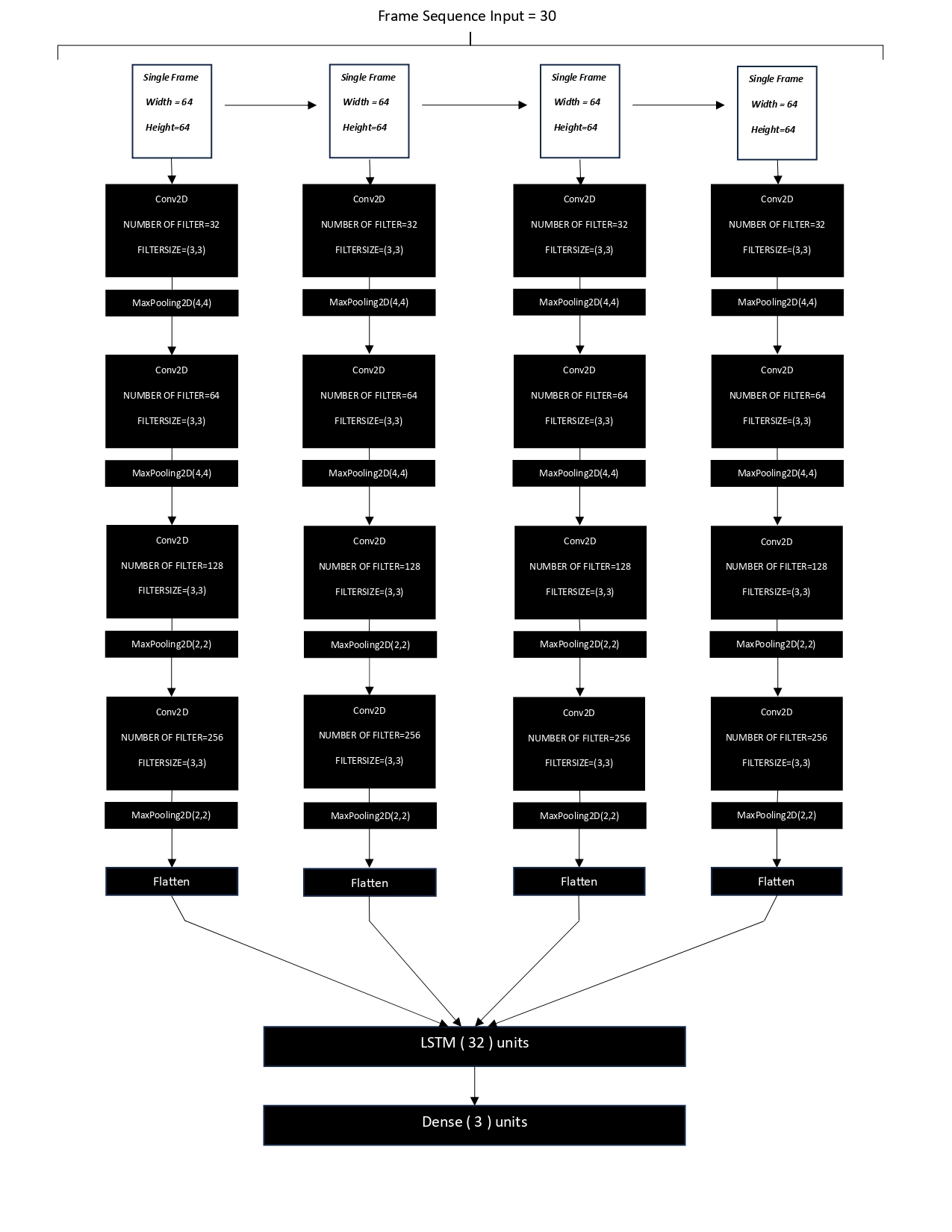
**Input Gate**: Define output values to update the memory state.

**Output Gate:** Determine what is output according to the input unit and memory.

***Time Distributed:***

The architecture of a convolutional neural network can extract the properties of one image at a time (one frame at a time), but here we have a sequence of frames, and therefore we need a way through which we can pass that sequence from frames to the two-dimensional convolutional neural network.

We proposed the use of Time Distributed so that the convolutional neural network can receive more than one input to it (more than one image at the same time), and thus we can pass the 160-frame visual segment to the two-dimensional convolutional neural network. A convolutional neural network extracts the properties of each frame independently from the next.



**CHAPTER 5**

**IMPLEMENTATION**

**5.1 ENVIRONMENT SETUP**

**Installing Python:**

1. To download and install Python visit the official website https://www.python.org/downloads/ and choose your version of Python

2. Once the download is complete, run the exe to install Python. Now click on Install Now.

3. You can see Python installed at this point.

4. When it finishes, you can see a screen that says the Setup was successful. Now click on “Close”.

**Installing and Configuration of PIP**

To install and configure pip on your computer, follow these instructions:

Note: The steps may vary slightly depending on your operating system. The following instructions assume you are using a Windows operating system.

1. Check if pip is already installed: Open a command prompt by pressing the Windows key + R,

type "cmd", and press Enter. In the command prompt, enter the following command and press Enter: pip --version

If you see output similar to `pip x.x.x from ...`, it means pip is already installed, and you can proceed to the configuration step. If you see an error or no output, move on to the installation step.

2. Install pip:

a) Download the get-pip.py script by visiting the official pip website: https://pip.pypa.io/en/stable/installing/

b) Right-click on the "get-pip.py" link and select "Save Link As" to save the file.

c)Open a command prompt and navigate to the directory where you saved the get-pip.py file using the `cd` command. For example, if you saved it in the Downloads folder, you would run: cd C:\Users\YourUsername\Downloads

d)Once you are in the correct directory, run the following command to install pip: python get-pip.py

Note: If you have multiple versions of Python installed, specify the Python version you want to use by replacing `python` with `python3` or `python2` accordingly.

3. Verify the installation:  
In the command prompt, enter the following command and press, Enter: pip --version. If you see output similar to `pip x.x.x from ...`, it means pip is installed successfully.

4. Configure pip:

a) Add the pip installation directory to the system's PATH environment variable to make it accessible from any location in the command prompt:

b) Right-click on the "This PC" or "My Computer" icon on your desktop and select "Properties".

c)Click on "Advanced system settings" on the left side of the window.

d)In the System Properties window, click on the "Environment Variables" button.

e) In the Environment Variables window, locate the "Path" variable under "System variables" and click on "Edit".

f) Add a new entry for the pip installation directory. By default, it is something like `C:\PythonXX\Scripts\` (replace `XX` with your Python version number).

g) Click "OK" to save the changes.

Note: If you had any command prompt windows open before modifying the PATH variable, you need to restart them for the changes to take effect.

After following these steps, you should have pip installed and configured on your computer. You can now use pip to install Python packages and libraries by running commands like `pip install package-name` in the command prompt.

**Installing NumPy**

To install a specific version of NumPy (version 1.19.2) using pip, follow these instructions:

1. Open a command prompt or terminal on your computer.

2. Type the following command and press Enter to execute it:

pip install numpy==1.19.2

This command tells pip to install NumPy version 1.19.2.

Note: Make sure you have pip installed and configured properly on your system. If you encounter any issues, you may need to upgrade pip or check your Python installation.

3. Wait for the installation process to complete. Pip will download and install the specified version of NumPy.

Once the installation is finished, you can use NumPy version 1.19.2 in your Python programs. Remember to import the module in your code using `import numpy`.

**Installing pafy**

1. To install pafy -a API to stream YouTube videos

**`**pip install pafy`

1. Wait till the progress bar completes

Once the installation is finished, you can use Pafy in your python programs.

Remember to import the module in your code using ‘import pafy’.

**Installing TensorFlow**

To install TensorFlow, it is important to have “Python” installed in your system. Python version 3.4+ is considered the best to start with TensorFlow installation.

Consider the following steps to install TensorFlow in Windows operating system.

**Step 1** − Verify the python version being installed.

**Step 2** − A user can pick up any mechanism to install TensorFlow in the system. We recommend “pip” and “Anaconda”. Pip is a command used for executing and installing modules in Python.

Before we install TensorFlow, we need to install Anaconda framework in our system.

**Step 3** − Execute the following command to initialize the installation of TensorFlow

conda create --name tensorflow python = 3.5

**Step 4** − After successful environmental setup, it is important to activate TensorFlow module.

activate tensorflow

**Step 5** − Use pip to install “Tensorflow” in the system. The command used for installation is mentioned as below −

pip install tensorflow

**Step 6** – Now you can see the Tensorflow installed.

# 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

**5.2 IMPLEMENTATION**

**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 Dataset consists of KTH dataset for walking (" [http://www.csc.kth.se/cvap/actions/walking.zip](https://colab.research.google.com/corgiredirector?site=http%3A%2F%2Fwww.csc.kth.se%2Fcvap%2Factions%2Fwalking.zip) ") and running (" [http://www.csc.kth.se/cvap/actions/running.zip](https://colab.research.google.com/corgiredirector?site=http%3A%2F%2Fwww.csc.kth.se%2Fcvap%2Factions%2Frunning.zip) ") and Kaggle dataset for fighting (" [https://www.kaggle.com/datasets/naveenk903/movies-fight-detection-dataset](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwww.kaggle.com%2Fdatasets%2Fnaveenk903%2Fmovies-fight-detection-dataset) "), consisting of videos staged by actors. The Dataset contains:

* **03** Action Categories
* **45** Videos per Action Category

For visualization, we will pick 03 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 ( 03 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('Dataset')

# 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)), len(all\_classes\_names))

# Iterating through all the generated random values.

for counter, random\_index in enumerate(range(len(all\_classes\_names)), 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'Dataset/{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'Dataset/{selected\_class\_Name}/{selected\_video\_file\_name}')

video\_reader.set(1, 25)

# Read the first frame of the video file.

\_, bgr\_frame = video\_reader.read()

bgr\_frame = cv2.resize(bgr\_frame ,(224,224))

# 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, 0.8, (0, 200, 255), 2)

# Display the frame.

plt.subplot(5, 4, counter);plt.imshow(rgb\_frame);plt.axis('off')

A person standing in a field

Description automatically generated

## ****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 = 30

# Specify the directory containing the UCF50 dataset.

DATASET\_DIR = "Dataset"

# Specify the list containing the names of the classes used for training. Feel free to choose any set of classes.

CLASSES\_LIST = ["walking", "fight", "running"]

**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.

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))

# Iterate through all the files present in the files list.

for file\_name in files\_list:

# Get the complete video path.

video\_file\_path = os.path.join(DATASET\_DIR, class\_name, file\_name)

# Extract the frames of the video file.

frames = frames\_extraction(video\_file\_path)

# Check if the extracted frames are equal to the SEQUENCE\_LENGTH specified above.

# So ignore the vides having frames less than the SEQUENCE\_LENGTH.

if len(frames) == SEQUENCE\_LENGTH:

# Append the data to their repective lists.

features.append(frames)

labels.append(class\_index)

video\_files\_paths.append(video\_file\_path)

# Converting the list to numpy arrays

features = np.asarray(features)

labels = np.array(labels)

# Return the frames, class index, and video file path.

return features, labels, video\_files\_paths

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()

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)

features = None

labels = None

**Step 4: 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.

A diagram of a diagram

Description automatically generated

We will also use **[TimeDistributed](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fkeras.io%2Fapi%2Flayers%2Frecurrent_layers%2Ftime_distributed%2F" \t "_blank)** 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.

A diagram of a layer of layers

Description automatically generated with medium confidence

### **Step 4.1: 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():

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'))

########################################################################################################################

# Display the models summary.

model.summary()

# Return the constructed LRCN model.

return model

**A diagram of a server

Description automatically generated with medium confidence**

**CHAPTER 6**

**RESULTS**

**LOSS AND ACCURACY CURVES**

**A graph of loss and loss

Description automatically generated**

**A graph of a graph with red and blue lines

Description automatically generated**

**A person running on a field

Description automatically generated**

**A screenshot of a video

Description automatically generated**

**A group of people in a room

Description automatically generated**

# Perform Single Prediction on the Test Video.

predict\_single\_action("Predict/fight2.mp4", SEQUENCE\_LENGTH)

1/1 [==============================] - 0s 77ms/step

Action Predicted: fight

Confidence: 0.9934989213943481

**As seen the model predicts the video with 0.99349 confidence and the prediction is "fight".**

**A group of people walking on a sidewalk

Description automatically generated**

# Perform Single Prediction on the Test Video.

predict\_single\_action("Predict/running.avi", SEQUENCE\_LENGTH)

1/1 [==============================] - 0s 70ms/step

Action Predicted: running

Confidence: 0.9945598244667053

**As seen the model predicts the video with 0.99455 confidence and the prediction is "Running".**

**Chapter 7**

**CONCLUSION AND FUTURE SCOPE**

**7.1 CONCLUSION**

The current research presented a study of the structure of a neural network capable of analysing and detecting suspicious movement (the movement that precedes the theft process), by using the dataset that we have, in addition to all the many videos on YouTube, we were able to propose a model capable of studying the sequence of several frames (160 frames) to identify and detect the nature of the movement within that period that represents (160 frames), the proposed model can provide great assistance to many stores in facilitating follow-up and monitoring, through the alerts that the model will appear when identifying a suspicious movement, and can also be used The system aims to provide the ability to review many of the previous video recordings to detect thefts that were not identified in the past.

**7.2 FUTURE SCOPE**

The fight detection system can be adapted to security contexts to identify instances of physical altercations or fights in various settings like public spaces, schools, transportation hubs.

The future holds bright for these systems: -

1. **Improved accuracy**
2. **Multi modal analysis**
3. **Real Time responses**
4. **Behavioural Analysis**
5. **Privacy concerns**
6. **Customization and Adaptability**
7. **Integration with IoT**
8. **Predictive analysis**
9. **Ethical and legal Implications**
10. **Augmented Reality and Virtual Reality**
11. **Global Adaptation**

**CHAPTER 8**

**REFERENCES**

[1] P.Bhagya Divya, S.Shalini, R.Deepa, Baddeli Sravya Reddy,*“Inspection of suspicious human activity in the crowdsourced areas captured in surveillance cameras”*,International Research Journal of Engineering and Technology (IRJET), December 2017.

[2] Jitendra Musale,Akshata Gavhane, Liyakat Shaikh, Pournima Hagwane, Snehalata Tadge, *“Suspicious Movement Detection and Tracking of Human Behavior and Object with Fire Detection using A Closed Circuit TV (CCTV) cameras ”*, International Journal for Research in Applied Science & Engineering Technology (IJRASET) Volume 5 Issue XII December 2017.

[3] 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.

[4] Dinesh Jackson Samuel R,Fenil E, Gunasekaran Manogaran, Vivekananda G.N, Thanjaivadivel T , Jeeva S , Ahilan A, *“Real time violence detection framework for football stadium comprising of big data analysis and deep learning through bidirectional LSTM”*,The International Journal of Computer and Telecommunications Networking,2019.