

# **HUMAN ACTIVITY DETECTION AND ANOMALY RECOGNITION**

Minor project report submitted in partial fulfillment of the requirement for the  
degree of Bachelor of Technology

in

**Computer Science and Engineering**

By

Manas Tiwari (211165)

Ritesh Dhiman (211138)

**UNDER THE SUPERVISION OF**

Dr. Amit Kumar Jakhar



Department of Computer Science & Engineering and Information  
Technology

**Jaypee University of Information Technology, Waknaghat,  
173234, Himachal Pradesh, INDIA**

## TABLE OF CONTENT

<b>Title</b>	<b>Page No.</b>
<b>Certificate</b>	<b>1</b>
<b>Acknowledgement</b>	<b>2</b>
<b>Abstract</b>	<b>3</b>
<b>Chapter-1 (Introduction)</b>	<b>4 - 7</b>
<b>Chapter-2 (Feasibility Study, Requirements Analysis and Design</b>	<b>8 - 12</b>
<b>Chapter-3 (Implementation)</b>	<b>13 - 30</b>
<b>Chapter-4 (Results)</b>	<b>31 - 33</b>
<b>References</b>	<b>34</b>

## **CERTIFICATE**

I hereby certify that the work which is being presented in the project report titled “Human Activity Detection and Anomaly Recognition” in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science and Engineering and submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out during the period from January 2024 to May 2024 under the supervision of Dr. Amit Kumar Jakhar, Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.

The matter presented in this project report has not been submitted for the award of any other degree of this or any other university.

**Ritesh Dhiman**

**211138**

**Manas Tiwari**

**211165**

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

**Dr. Amit Kumar Jakhar**

**Assistant Professor**

Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat,

## ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to almighty God for His divine blessing makes it possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to Supervisor **Dr. Amit Kumar Jakhar, Assistant Professor**, Department of CSE Jaypee University of Information Technology, Wazirpur, New Delhi. Deep Knowledge & keen interest of my supervisor in the field of “**Research Area**” to carry out this project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to **Dr. Amit Kumar Jakhar**, Department of CSE, for his kind help to finish my project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

Ritesh Dhiman (211138)

Manas Tiwari (211165)

## ABSTRACT

Applications such as intelligent video surveillance and retail behavior analysis are becoming more and more dependent on the ability to recognize human behavior in real-world settings. Nowadays, video surveillance—which is essential for security—is a part of daily existence. Since it is not feasible to watch CCTV footage manually, automated solutions are required for the effective identification and analysis of anomalous events.

Artificial intelligence (AI), machine learning (ML), and deep learning (DL) are used in video surveillance systems to identify suspicious activity in human behavior. This technology is generally used in public locations such as airports, railway stations, banks, and businesses. AI integration makes it possible for computers to simulate human cognitive functions. Using training data to inform predictions about new data is the main goal of machine learning. The application of deep learning (DL), and in particular deep neural networks (DNNs), has been made easier by the development of GPU processors and the availability of large datasets.

DNNs can automatically extract and represent high-level features from picture data and are skilled at difficult learning tasks. While long short-term memory (LSTM) models manage video streams by learning long-term dependencies, which are necessary for detecting sequences of events, convolutional neural networks (CNNs) learn visual patterns directly from image pixels. LSTM networks are very good at recalling prior inputs, which is important for deciphering continuous actions in video recordings.

The suggested system will make use of CCTV footage to keep an eye on how people behave on campus and to send out alerts when it notices questionable activity. This intelligent video monitoring system's event detection and human behavior recognition are important features. Three stages make up the training process for such a surveillance system: inference, model training, and data preparation.

The process of preparing data entails gathering and preparing the video footage. In order for the CNN and LSTM networks to acquire pertinent patterns and behaviors, this data must be fed into them during model training. By using the trained model to analyze and react to suspicious activity in real-time video streams, inference improves security and safety in monitored situations.

# Chapter 01:INTRODUCTION

## 1.1 Introduction

Numerous real-world applications of human behavior identification exist, such as intelligent video monitoring and analysis of consumer purchasing patterns. The use of video surveillance has areas of applicability, particularly for indoor and outdoor spaces. Security requires surveillance as a fundamental component. For reasons of safety and security, security cameras have become a necessary element of modern living.

It is now difficult to manually watch every incident captured by a CCTV (closed-circuit television) camera. It takes a lot of time to manually look for the identical incident in the recorded video, even if it has already happened. Within the field of automated video surveillance systems, the analysis of anomalous events from video is a developing field. Video surveillance systems use human behavior detection as an automated method of intelligently identifying any suspicious conduct.

Many effective algorithms are available for automatically identifying human behavior in public spaces such as banks, workplaces, exam rooms, airports, and train stations. The use of deep learning, machine learning, and artificial intelligence in the field of video surveillance is new. A computer that uses artificial intelligence is able to think like a person. Predicting future data and learning from training data are crucial aspects of machine learning. With the availability of large datasets and GPU (Graphics Processing Unit) processors nowadays, deep learning is applied. One of the best architectures for challenging learning problems is Deep Neural Networks. Deep Learning models provide high-level representations of picture data and automatically extract features.

Because the feature extraction procedure is entirely automated, this is more widely applicable.

Convolutional neural networks (CNN) can directly learn visual patterns from the image pixels. Long short-term memory (LSTM) models can learn long-term dependencies in the context of video streams. The LSTM network is able to retain information. The suggested system will gently wear itself whenever a suspicious incident happens and use footage from CCTV cameras to track people's conduct on campus. Event detection and the identification of human activity are the two main elements of intelligent video surveillance. Three stages make up the complete training process of a surveillance system: data preparation, model training, and inference.

## 1.2 Objective

The goal of this research is to create a system that uses deep learning and sophisticated artificial intelligence (AI) to identify suspicious activity in surveillance footage.

With the proliferation of CCTV cameras, manual monitoring and event detection have become impractical due to the volume of footage generated. This project aims to automate the detection of abnormal and suspicious behaviors in real-time, enhancing security and safety in various settings, including university campuses, banks, airports, and other sensitive areas.

The main objective is to develop a real-time, accurate, and efficient system that can recognize unusual behaviors like theft, vandalism, fighting, and other types of activities. To identify and categorize human behaviors from video streams, the project will make use of deep neural networks, namely a combination of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. This method, which is appropriate for video data processing, combines the power of CNNs in feature extraction with the knowledge of temporal sequences offered by LSTMs.

To achieve this, the system will undergo several key phases:

- 1. Data Preparation:** Capturing and preprocessing video footage, extracting frames, and normalizing them for model training.
- 2. Model Training:** Using a Long-term Recurrent Convolutional Network (LRCN) to train on labeled datasets, including both normal and suspicious activities, ensuring the model can accurately classify behaviors.
- 3. Inference:** Implementing the trained model to monitor live video feeds and generate alerts for detected suspicious activities.

This system is designed to operate in real-time, providing timely warnings and allowing for prompt intervention. By automating surveillance, the project aims to enhance security measures, reduce the workload on human monitors, and increase the overall efficiency and effectiveness of surveillance systems in various high-risk and high-traffic environments. The project also addresses limitations of previous models by optimizing computational efficiency, making it feasible for real-world applications.

## 1.3 Motivation

Video surveillance is crucial in identifying suspicious human activity in order to stop theft, terrorist attacks using abandoned objects as explosives, vandalism, fighting, and personal attacks, as well as fire in various highly sensitive locations like banks, hospitals, malls, bus and train stations, airports,

refineries, nuclear power plants, schools, university campuses, borders, etc. In order to keep student belongings safe from theft and damage, video surveillance can be deployed in academic settings such as university campuses.

Additionally, it will assist in preventing pupils from acting inappropriately and from fighting with one another. For the protection of the instructors and pupils, it will keep an eye on the perimeter of the university, school, and academic facilities. When exams are being administered, video surveillance can be utilized to keep an eye on any questionable behavior by students in the exam room. Finally, academic institutions can use video surveillance to uphold law and order while using fewer guards.

## **1.4 Language Used**

Python

## **1.5 Technical Requirements ( Hardware)**

RTX 3060

i7 11800h

min 8GB RAM

## **1.6 Deliverables/Outcomes**

The primary deliverables of this project include the development and implementation of an efficient and effective suspicious activity detection system using deep learning techniques. The outcomes can be summarized as follows:

### **1. Developed LRCN Model for Suspicious Activity Detection:**

- Convolutional neural networks (CNNs) and long short-term memory (LSTM) networks are combined in the Long-term Recurrent Convolutional Network (LRCN) model to evaluate video frames and record temporal dependencies, leading to precise predictions of suspicious actions.

### **2. Improved Real-Time Detection Capability:**

- By reducing the frame size to 64x64 pixels and optimizing the model architecture to include only 12 layers, the system achieves faster processing times. This enables real-time detection and monitoring of activities, making it practical for real-world surveillance applications.



### **3. High Accuracy with Diverse Dataset:**

- The system is trained on a large dataset, which includes the fighting detection dataset from Kaggle and the running and walking activities dataset from KTH. With the help of this varied training set, the model is able to discern between normal and suspicious behaviors with an accuracy of 82%.

### **4. Efficient Data Pre-processing Pipeline:**

- A robust pre-processing pipeline was developed, which involves reading video frames, resizing, normalizing, and storing them in numpy arrays. This ensures that the model receives consistent and high-quality input data for training and prediction.

### **5. Comprehensive Evaluation Metrics:**

- The model's performance is evaluated using training and validation accuracy and loss metrics. Graphs illustrating these metrics show the model's learning curve and its effectiveness in distinguishing between different activity classes.

### **6. Scalable and Deployable System:**

- The system is made to be expandable and installable in a variety of monitoring environments, including public transit hubs, college campuses, and other sensitive locations. Its application is further enhanced by the fact that it can run on low-end devices.

### **7. Enhanced Security and Safety Measures:**

- By integrating this automated surveillance system, institutions can significantly enhance their security protocols, promptly detecting and addressing suspicious activities. This contributes to the safety of individuals and the protection of assets.

These deliverables demonstrate the project's potential to transform video surveillance systems through advanced AI and deep learning techniques, ensuring improved security and real-time monitoring capabilities.

## Chapter 02: Feasibility Study, Requirements Analysis and Design

### 2.1 Feasibility Study

The linked work offers many methods for identifying human activities in video. The works' aim was to identify any unusual or suspicious activity in a video surveillance system. [1] The technique known as Advance Motion Detection (AMD) was employed to find an illegal entry into a prohibited location. Using background subtraction, the item was identified in the first step, and it was then retrieved from frame sequences. The identification of suspicious activity was the second stage. The system's advantage was that the algorithm's low computing complexity allowed it to process videos in real time. However, the system's storage capacity was constrained. Moreover, it could not be used with a high-tech method of filming in the monitoring zones.

[2] References–[2] presented a semantic-based method. After the video data was analyzed, background removal was used to identify the things in the foreground. Following subtraction, a Haar-like algorithm is used to categorize the objects as either living or non-living. An technique for Real-Time blob matching was used to track objects. This paper also detected the presence of fire. [3] People monitoring could identify the anomalous events in the video footage. The footage is analyzed using background subtraction techniques to identify human presence. CNN is used to extract the features, which are then input into a DDBN (Discriminative Deep Belief Network).

Additionally, the DDBN receives labeled videos of certain suspicious situations, from which its features are also retrieved. Subsequently, a DDBN was used to compare the features recovered from the labeled sample video of classified suspicious actions with features retrieved from CNN, resulting in the detection of a variety of suspicious activities from the provided video. [4] To stop crowds or athletes from acting violently, a deep learning-based real-time violence detection system was created. Frames from real-time videos were retrieved in a spark environment. The security personnel will be notified by the system if it detects any violent football.

The system recognizes the video actions in real time and notifies the security personnel to avert the violence beforehand. Using the VID dataset, violence in football stadiums was detected with 94.5% accuracy.

#### 2.1.1 Problem Definition

The project's goal is to use deep learning techniques to create an automated system that can identify suspicious activity in surveillance videos. It is not feasible to manually monitor CCTV footage because of time and budget limitations. This method leverages video streams recorded in vulnerable regions like university campuses to detect anomalous behaviors like fighting using Long-term

Recurrent Convolutional Networks (LRCN). The system classifies activities in real-time by pre-processing video frames and utilizing a combination of Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks. This improves security by swiftly informing authorities of potential dangers.

### **2.1.2 Problem Analysis**

The challenge at hand is identifying questionable human behavior from surveillance footage, an essential part of maintaining security in sensitive locations.

Manual monitoring of CCTV footage is impractical due to time and resource constraints.

Automated systems, leveraging AI, ML, and deep learning, can efficiently identify abnormal behaviors such as fights, theft, or unauthorized entry. The proposed solution uses a Long-term Recurrent Convolutional Network (LRCN) to classify activities by analyzing sequences of video frames. This system aims to provide real-time alerts with improved accuracy and reduced computational load, making it suitable for practical applications in diverse environments like campuses and public spaces.

### **2.1.3 Solution**

In order to identify questionable behavior, such as fighting among students, the project suggests implementing a Long-term Recurrent Convolutional Network (LRCN) in a video surveillance system. The system records video, preprocesses it by removing and standardizing frames, and then classifies behaviors using deep learning. The model achieves 82% accuracy in real-time detection by integrating Long Short-Term Memory (LSTM) networks for handling temporal input and Convolutional Neural Networks (CNNs) for feature extraction. This technology automatically detects and notifies authorities of potential threats, hence improving security in academic institutions.

## **2.2 Requirements**

### **2.2.1 Functional Requirements**

#### **1. Video Capture and Input Handling:**

- The system must capture real-time video footage from CCTV cameras installed at various locations within the campus.
- It should handle multiple video streams simultaneously, ensuring coverage of all surveillance areas.

## **2. Video Pre-processing:**

- Extract 30 frames per video at equal intervals, resize each frame to 64x64 pixels, and normalize pixel values for uniformity.
- Store the processed frames in a sequence within a numpy array for model input.

## **3. Class Prediction and Behavior Detection:**

- Utilize a Long-term Recurrent Convolutional Network (LRCN) to classify the sequences into three categories: suspicious (fighting), normal (walking, running).
- The system should generate real-time alerts to the relevant authorities when suspicious activities are detected.

## **4. Training and Model Enhancement:**

- Train the model using KTH and Kaggle datasets, ensuring 75% of data for training and 25% for testing.
- The model should achieve at least 82% accuracy with reduced training and prediction times suitable for real-time applications.

## **5. System Maintenance and Upgrades:**

- Enable easy integration of additional datasets for model improvement.
- The system should support updates to the LRCN model for enhanced performance and accuracy without significant downtime.

## **6. User Interface and Alert Mechanism:**

- Provide an intuitive user interface for monitoring video feeds and viewing alerts.
- Implement an automated alert system to notify security personnel of suspicious activities via messages or notifications.

These functional requirements ensure comprehensive and efficient surveillance, promoting campus safety through real-time monitoring and timely alerts.

### **2.2.2 Non-Functional Requirements**

**1. Performance:** The system must process video streams in real-time to detect and alert suspicious activities with minimal latency. The target detection time for any suspicious activity should not exceed 2 seconds.

**2. Scalability:** The system should handle multiple video streams simultaneously from various CCTV cameras across different locations within a campus. It should be capable of scaling up to accommodate additional cameras without significant degradation in performance.

**3. Reliability:** The system must provide consistent and accurate detection under different conditions such as varying lighting, weather, and crowded scenarios. It should achieve an uptime of 99.9%, ensuring continuous monitoring without interruptions.

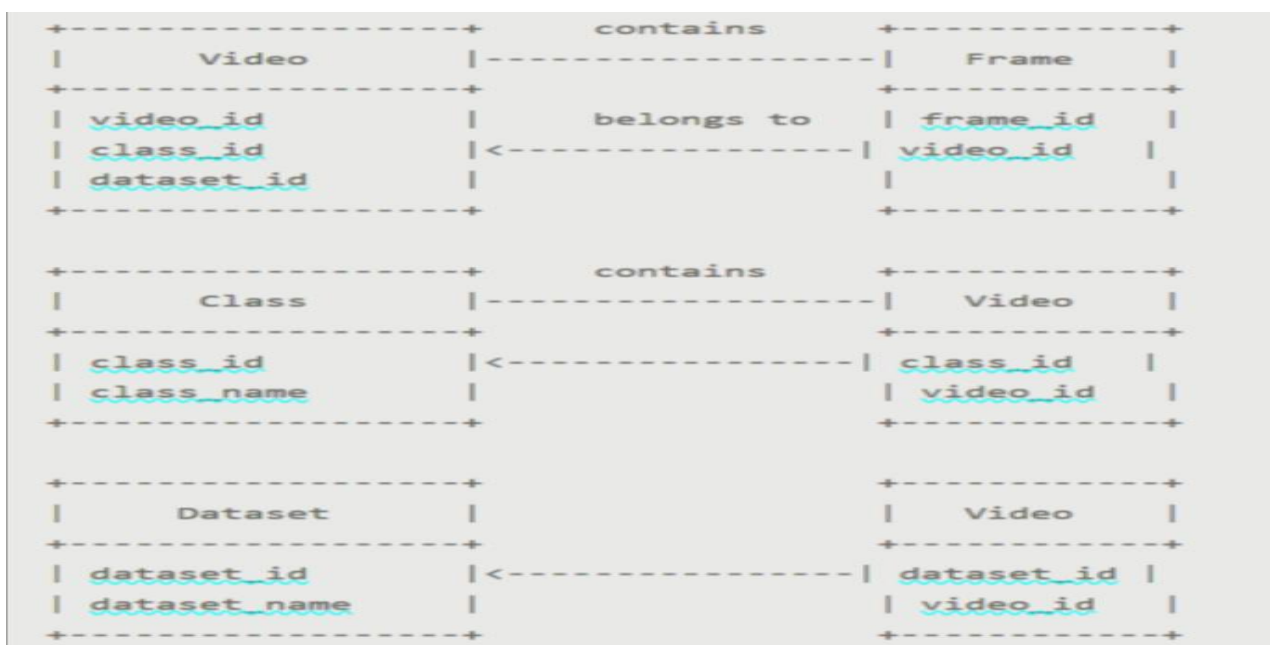
**4. Usability:** The system should offer an intuitive interface for security personnel to review alerts and flagged events quickly. Alerts must be clearly presented with timestamped video snippets highlighting the suspicious activity.

**5. Maintainability:** The system must be designed for easy updates and maintenance. This includes modular components for the algorithm, which allows for updates to the detection models without affecting the entire system.

**6. Security:** The system must ensure that video data and alerts are securely transmitted and stored, protecting against unauthorized access and tampering. Compliance with data protection regulations is mandatory.

**7. Compatibility:** The system should be compatible with various types of CCTV hardware and support integration with existing campus security infrastructure and communication systems.

## 2.3 E-R Diagram / Data-Flow Diagram (DFD)



```
+-----+ belongs to +-----+
|   Video   |-----|   Class   |
+-----+          +-----+
| video_id   |<-----| class_id   |
| file_path   |          | class_name |
| predicted_class |      +-----+
| confidence   |
+-----+
```

## Chapter 03: IMPLEMENTATION

### 3.1 Data Set Used in the Minor Project

#### KTH Human Motion:

<https://www.kaggle.com/datasets/beosup/kth-human-motion?resource=download>

#### Video Fight Detection Dataset:

<https://www.kaggle.com/datasets/naveenk903/movies-fight-detection-dataset>

### 3.2 Data Set Features

#### 3.2.1 Types of Data Set

Image dataset

Video Dataset

#### 3.2.2 Number of Attributes, fields, description of the data set

##### Description:

- **Type:** Action recognition binary classification dataset
- **Name:** Video Fight Detection Dataset and KTH Human Motion dataset
- **Source:** Kaggle
- **Purpose:** The dataset is designed for binary classification tasks specifically for recognizing fight actions within videos.

##### Attributes/Fields:

1. **Video files.** 8 folders each containing 100 videos.
2. **Labels:** Binary labels indicating whether the action in the video is a fight or not.

##### Additional Information:

- **KTH Human Motion Dataset:** A widely-used dataset in action recognition research, containing various human motions captured in video format.
- **Action Recognition:** The task of identifying actions or activities performed by humans in videos, often used for surveillance, sports analysis, and human-computer interaction.

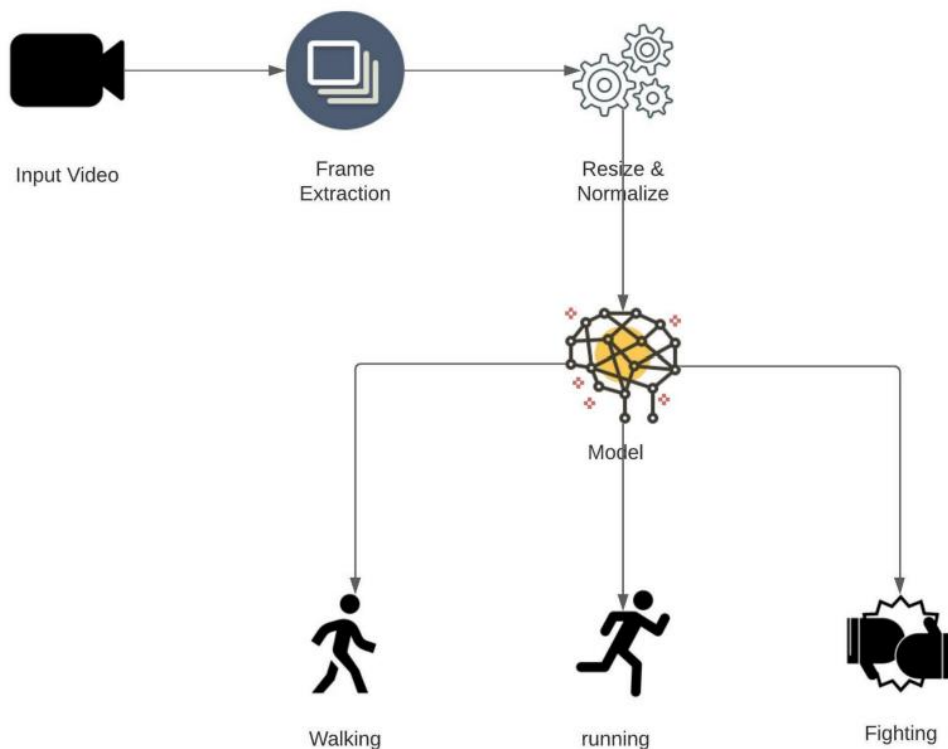
##### Summary:

The Video Fight Detection Dataset is a subset or extension of the KTH Human Motion

dataset, focusing specifically on detecting fight actions within videos. It consists of video files and corresponding binary labels indicating the presence or absence of fight actions. This dataset serves as a valuable resource for binary action recognition tasks, particularly in the context of surveillance and security applications.

### 3.3 Design of Problem Statement

Our suggested solution uses an LRCN (Long-term Recurrent Convolutional Network) to identify abnormal activity. Recognizing the temporal data in the video is crucial for the efficient classification of anomalous activity. CNN is now mostly utilized to extract important features from every video frame. In order to correctly classify the input, features must be retrieved from CNN; as a result, CNN must be able to recognize and extract the appropriate characteristics from video frames. The video's thirty-frame sequences are taken out and given to the LRCN Model.



### 3.4 Algorithm / Pseudo code of the Project Problem

#### 1. Data Collection and Preprocessing:

- Collect surveillance video footage from CCTV cameras.
- Preprocess the video data:
  - Extract frames from the video.
  - Adjust the frame sizes to a consistent size, such as 64 by 64 pixels.
  - Set the pixel values to a standard range of 0 to 1.



## **2. Model Training:**

- For anomaly detection, apply a deep learning architecture, such as the Long-term Recurrent Convolutional Network (LRCN).

- Describe the architecture of the model:

- Convolutional Neural Networks (CNNs) are combined to extract features from frames.

- To manage temporal dependencies, use Long Short-Term Memory (LSTM) networks.

- Divide the dataset into sets for testing and training.

- Utilizing the training data, train the model:

- Input the preprocessed video frames.

- Output the class labels corresponding to normal or suspicious activities.

- Optimize hyperparameters (e.g., epochs, batch size, learning rate) to improve model performance.

## **3. Model Evaluation:**

- Evaluate the trained model using the testing data:

- Assess accuracy, precision, recall, and F1-score metrics.

- Analyze confusion matrices to understand model performance.

- Fine-tune the model if necessary based on evaluation results.

## **4. Real-time Deployment:**

- Implement the trained model for real-time surveillance:

- Continuously monitor incoming video streams from CCTV cameras.

- Apply the trained model to classify activities as normal or suspicious.

- Trigger alerts or warnings for any detected suspicious activities.

- Ensure the system's scalability and efficiency to handle large volumes of video data in real-time.

## **5. System Maintenance and Updates:**

- Update the model frequently with fresh information to accommodate changing surveillance scenarios.

- Monitor system performance and make necessary adjustments to improve accuracy and reliability.

- Stay informed about advancements in deep learning and surveillance technology for potential enhancements.

## **Pseudo-code:**

```

#This assumes that you have a dataset with labels for video frames that have been previously
processed.
# Define and put the LRCN model together (model = define_lrcn_model()).compile(adam as the
optimizer, binary_crossentropy as the loss, and accuracy as the metrics)
#Divide the dataset into testing and training sets.
# Train the model

history = model.fit(train_data, epochs=70, batch_size=4, validation_split=0.25)

# Train the model with test_data using split_dataset(data).
#Value the accuracy minus the model loss.assess(test_data)
## Real-time implementation when it's true:
capture_frame_from_CCTV() = frame; preprocessed_frame = preprocess_frame(frame);
prediction is equivalent to model.predict(preprocessed_frame).
Should the forecast be 'suspicious', alert_authority()
#Updating and maintaining the system
# Regularly add new data to the model
# Regularly update the model with new data

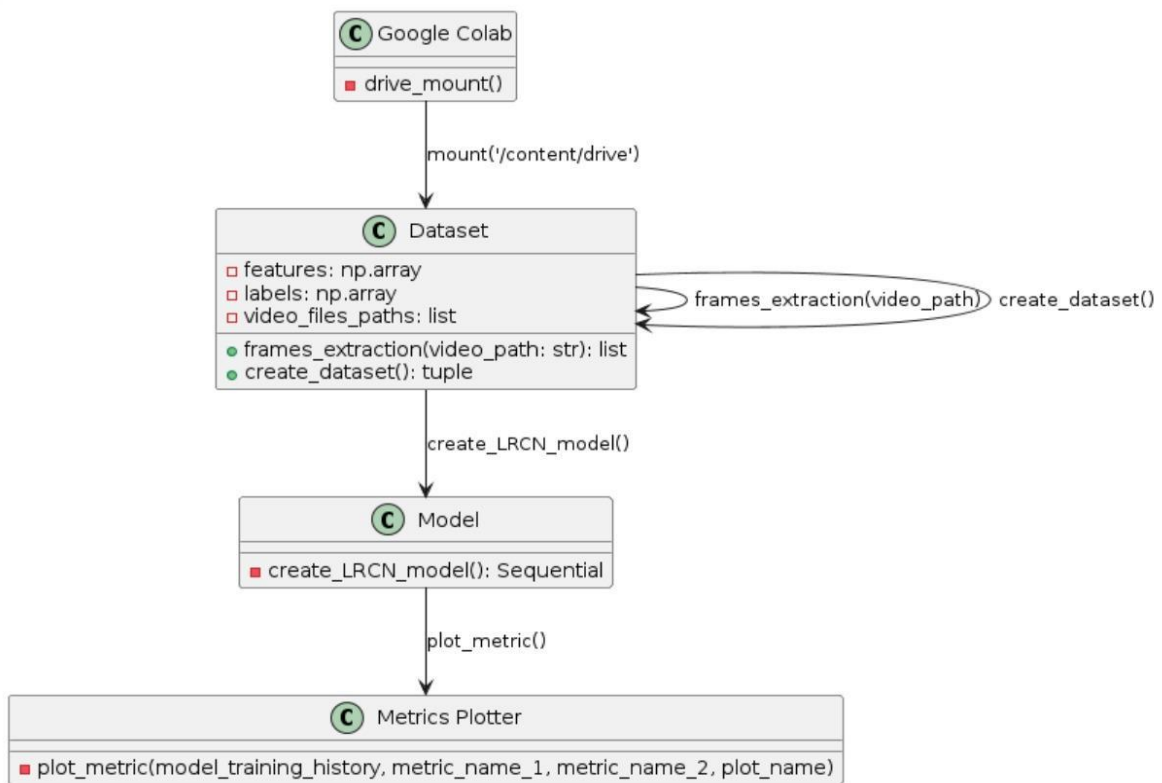
# Track system performance and make necessary parameter adjustments

```

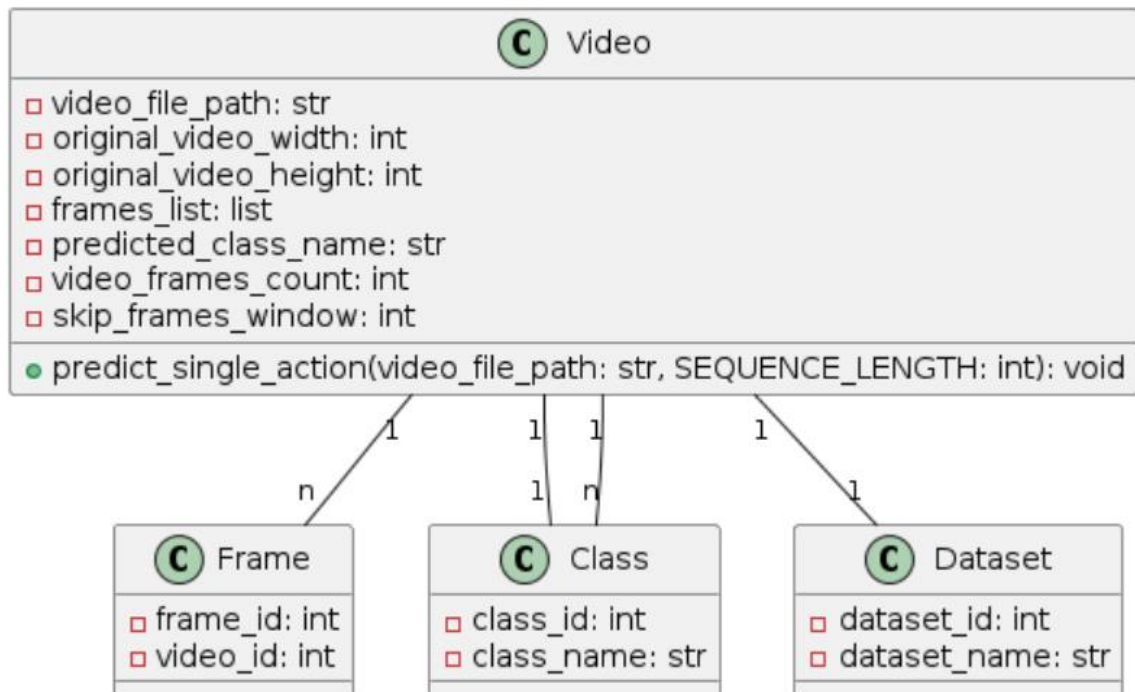
This pseudo-code provides a high-level overview of the project's algorithm and implementation steps. The actual implementation will require specific libraries, functions, and configurations tailored to the chosen deep learning framework and deployment environment.

## 2.5 Flow graph of the Minor Project Problem


### - Model creation flow Graph



### Model Prediction flow Graph



## 3.6 Screen shots of the various stages of the Project

✓ 18s  `pip install tensorflow opencv-python`

*Fig 1: Installing nessessary libraries ( Tenserflow )*


✓ 7s `[2] pip install PyDrive`

*Fig 2: Installing nessessary libraries ( Pydrive )*

✓ 11s `[6] !pip install pafy youtube-dl moviepy`

*Fig 3: Installing youtube-dl moviepy*

✓ 25s `[3] from google.colab import drive  
drive.mount('/content/drive')`

 Mounted at /content/drive

✓ 28s `[4] #file_path = '/content/drive/MyDrive/Minor__Data/minor.zip'  
!cp "/content/drive/MyDrive/Minor__Data/minor.zip" "/content/"`

*Fig 4: Drive mounting and extracting data*

✓ 26s `[5] !unzip minor`

*Fig 5:Unzipping the data*

```
✓ [7] import os
0s    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
```

*Fig 6:Importing necessary libraries*

```

✓ [10] # Create a Matplotlib figure and specify the size of the figure.
38 plt.figure(figsize = (20, 20))

# Get the names of all classes/categories in UCF50.
all_classes_names = os.listdir('minor')

# 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'minor/{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'minor/{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')

```



*Fig 7: Displaying Frames from randomly selected videos*

```

✓ [11] # Specify the height and width to which each video frame will be resized in our dataset.
0s      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 = "minor"

      # Specify the list containing the names of the classes used for training. Feel free to choose any set of classes.
      CLASSES_LIST = ["walking", "fights", "running"]

```

```

✓ ▶ def frames_extraction(video_path):
0s    ...

    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

```

*Fig 8: Resizing the dataset and frame extraction*

```

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

```

```

✓ [14] # Create the dataset.
3m features, labels, video_files_paths = create_dataset()

```

```

↔ Extracting Data of Class: walking
   Extracting Data of Class: fights
   Extracting Data of Class: running

```

*Fig 9: Creating the required dataset*

```

✓ [16] # Split the Data into Train ( 75% ) and Test Set ( 25% ).
0s 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

```

*Fig 10: Splitting the dataset for training and testing*



```

✓ 0s def create_LRCN_model():
    """
    This function will construct the required LRCN model.
    Returns:
        model: It is the required constructed LRCN model.
    """

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

    # Define the Model Architecture.
    #####

    model.add(TimeDistributed(Conv2D(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

```

```
✓ [18] model = create_LRCN_model()
```

⇒ Model: "sequential"

Layer (type)	Output Shape	Param #
time_distributed (TimeDistributed)	(None, 30, 64, 64, 32)	896
time_distributed_1 (TimeDistributed)	(None, 30, 16, 16, 32)	0
time_distributed_2 (TimeDistributed)	(None, 30, 16, 16, 64)	18496
time_distributed_3 (TimeDistributed)	(None, 30, 4, 4, 64)	0
time_distributed_4 (TimeDistributed)	(None, 30, 4, 4, 128)	73856
time_distributed_5 (TimeDistributed)	(None, 30, 2, 2, 128)	0
time_distributed_6 (TimeDistributed)	(None, 30, 2, 2, 256)	131328
time_distributed_7 (TimeDistributed)	(None, 30, 1, 1, 256)	0
time_distributed_8 (TimeDistributed)	(None, 30, 256)	0
lstm (LSTM)	(None, 32)	36992
dense (Dense)	(None, 3)	99

=====  
Total params: 261667 (1022.14 KB)

```
# 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, callbacks = [early_stopping_callback])

Epoch 1/70
42/42 [=====] - 27s 536ms/step - loss: 0.7531 - accuracy: 0.5833 - val_loss: 0.4312 - val_accuracy: 0.7193
Epoch 2/70
42/42 [=====] - 27s 643ms/step - loss: 0.5878 - accuracy: 0.5952 - val_loss: 0.4807 - val_accuracy: 0.6842
Epoch 3/70
42/42 [=====] - 21s 513ms/step - loss: 0.5165 - accuracy: 0.6667 - val_loss: 0.4333 - val_accuracy: 0.6842
Epoch 4/70
42/42 [=====] - 22s 516ms/step - loss: 0.4729 - accuracy: 0.6190 - val_loss: 0.4302 - val_accuracy: 0.6842
Epoch 5/70
42/42 [=====] - 23s 534ms/step - loss: 0.4718 - accuracy: 0.6964 - val_loss: 0.4293 - val_accuracy: 0.6842
Epoch 6/70
42/42 [=====] - 21s 511ms/step - loss: 0.4761 - accuracy: 0.6607 - val_loss: 0.4184 - val_accuracy: 0.7193
```

Fig 11: Creating the LRCN Model

```
# Save your Model.
model.save("Minor_Project_LRCN_Model.h5")
```

Fig 12: Saving the model

```

[22] def plot_metric(model_training_history, metric_name_1, metric_name_2, plot_name):
    ...
    This function will plot the metrics passed to it in a graph.
    Args:
        model_training_history: A history object containing a record of training and validation
                                loss values and metrics values at successive epochs
        metric_name_1:          The name of the first metric that needs to be plotted in the graph.
        metric_name_2:          The name of the second metric that needs to be plotted in the graph.
        plot_name:              The title of the graph.
    ...

    # Get metric values using metric names as identifiers.
    metric_value_1 = model_training_history.history[metric_name_1]
    metric_value_2 = model_training_history.history[metric_name_2]

    # Construct a range object which will be used as x-axis (horizontal plane) of the graph.
    epochs = range(len(metric_value_1))

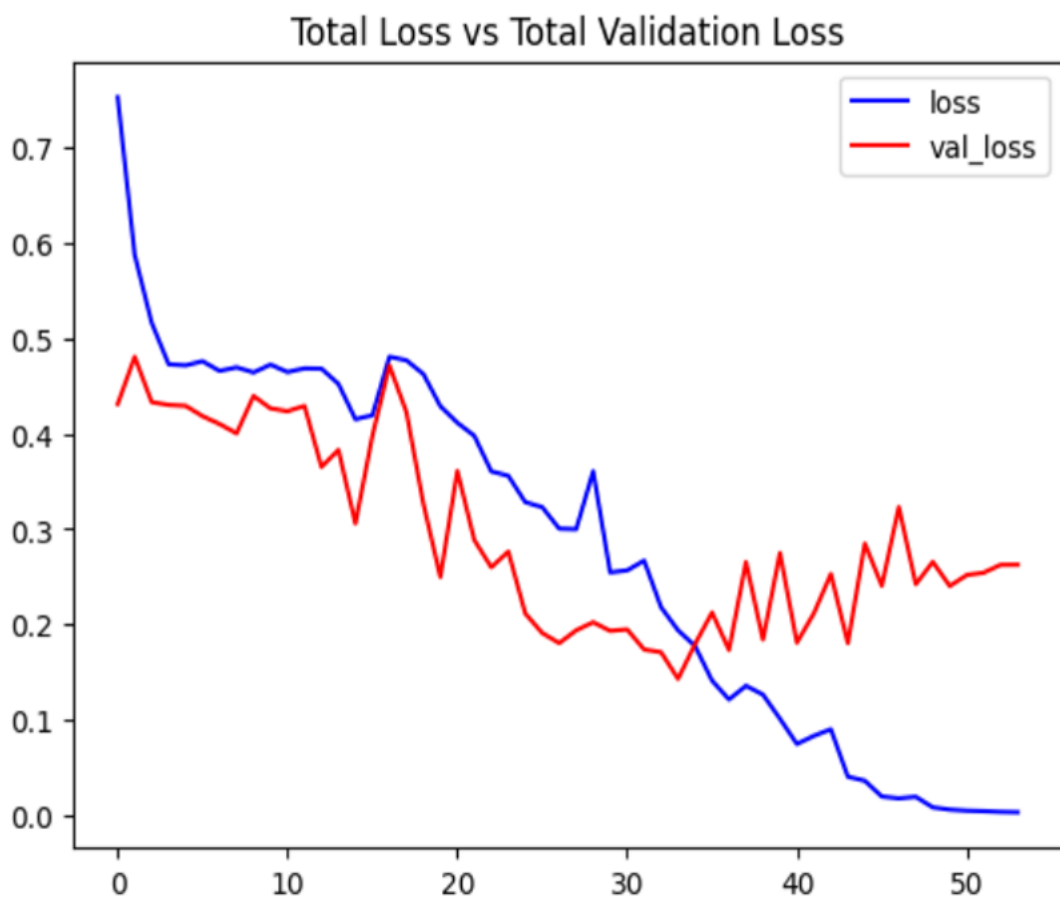
    # Plot the Graph.
    plt.plot(epochs, metric_value_1, 'blue', label = metric_name_1)
    plt.plot(epochs, metric_value_2, 'red', label = metric_name_2)

    # Add title to the plot.
    plt.title(str(plot_name))

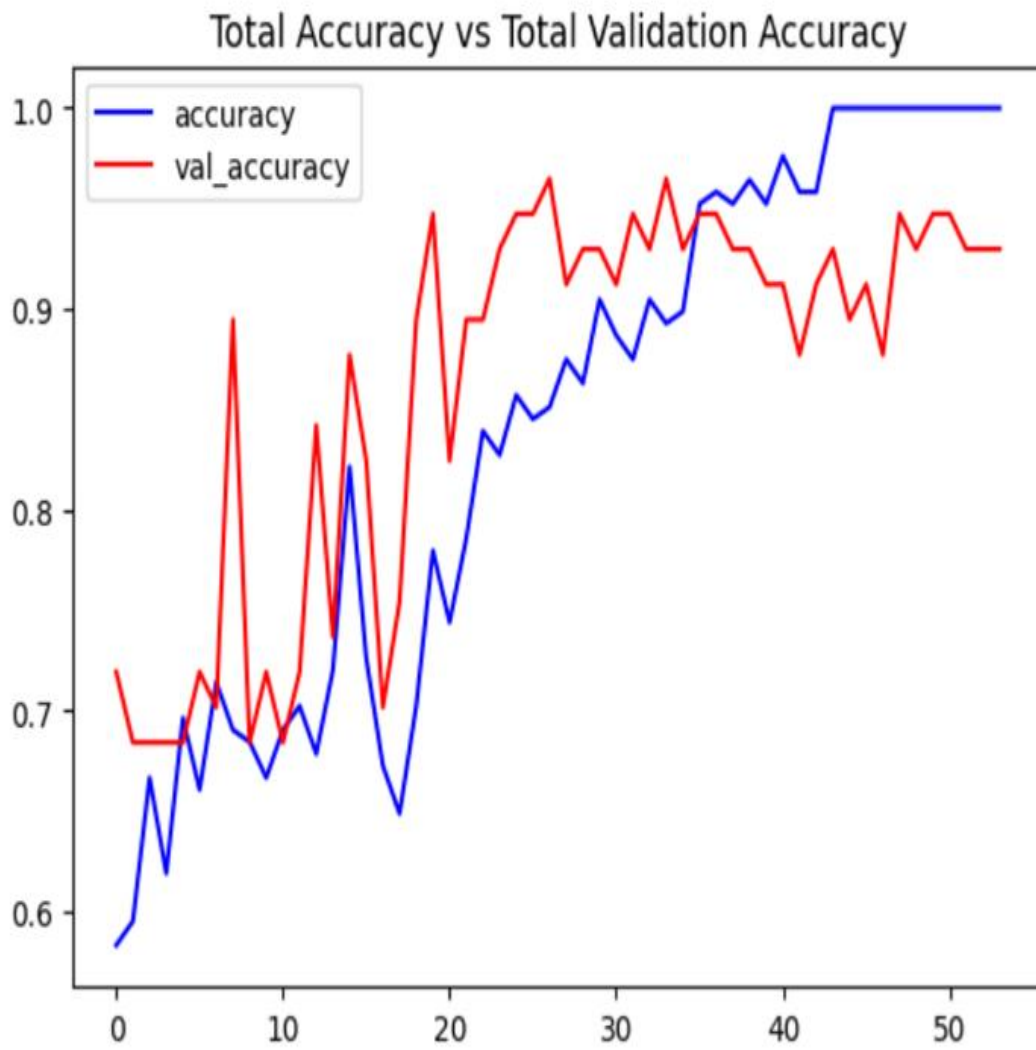
    # Add legend to the plot.
    plt.legend()

[23] plot_metric(model_training_history, 'loss', 'val_loss', 'Total Loss vs Total Validation Loss')

```



```
✓ [24] plot_metric(model_training_history, 'accuracy', 'val_accuracy', 'Total Accuracy vs Total Validation Accuracy')
```



*Fig 13: plotting the metrics of the model in the graph*

✓  
10s

```

# 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)

```

```

1/1 [=====] - 0s 47ms/step
1/1 [=====] - 0s 48ms/step
1/1 [=====] - 0s 51ms/step
1/1 [=====] - 0s 46ms/step
1/1 [=====] - 0s 47ms/step
1/1 [=====] - 0s 48ms/step
1/1 [=====] - 0s 47ms/step
1/1 [=====] - 0s 50ms/step
1/1 [=====] - 0s 56ms/step
1/1 [=====] - 0s 47ms/step
1/1 [=====] - 0s 49ms/step
1/1 [=====] - 0s 51ms/step
1/1 [=====] - 0s 50ms/step
1/1 [=====] - 0s 53ms/step
1/1 [=====] - 0s 46ms/step
1/1 [=====] - 0s 48ms/step
1/1 [=====] - 0s 48ms/step
1/1 [=====] - 0s 53ms/step
1/1 [=====] - 0s 46ms/step
1/1 [=====] - 0s 48ms/step
1/1 [=====] - 0s 48ms/step
1/1 [=====] - 0s 51ms/step
1/1 [=====] - 0s 46ms/step
1/1 [=====] - 0s 46ms/step
1/1 [=====] - 0s 45ms/step
1/1 [=====] - 0s 47ms/step
1/1 [=====] - 0s 58ms/step
1/1 [=====] - 0s 52ms/step
Accuracy = 84.0

```

Fig 14: Calculating the accuracy of the model

```
[28] from tensorflow.keras.models import load_model
```

```
[29] model = load_model('Minor_Project_LRCN_Model1.h5')
```

Fig 14: Loading the model

```

✓ 0s [30] def predict_single_action(video_file_path, SEQUENCE_LENGTH):
    """
    This function will perform single action recognition prediction on a video using the LRC
    Args:
    video_file_path: The path of the video stored in the disk on which the action recogniti
    SEQUENCE_LENGTH: The fixed number of frames of a video that can be passed to the model
    """

    # Initialize the VideoCapture object to read from the video file.
    video_reader = cv2.VideoCapture(video_file_path)

    # Get the width and height of the video.
    original_video_width = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))
    original_video_height = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # Declare a list to store video frames we will extract.
    frames_list = []

    # Initialize a variable to store the predicted action being performed in the video.
    predicted_class_name = ''

    # Get the number of frames in the video.
    video_frames_count = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))

    # Calculate the interval after which frames will be added to the list.
    skip_frames_window = max(int(video_frames_count/SEQUENCE_LENGTH),1)

    # Iterating the number of times equal to the fixed length of sequence.
    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)

```

```

✓ 1s [31] predict_single_action("/content/minor/fights/newfi1.avi",SEQUENCE_LENGTH)

```

```

➡ 1/1 [=====] - 1s 609ms/step
Action Predicted: fight
Confidence: 0.9994224309921265

```

```

✓ 0s [32] predict_single_action("/content/minor/running/person01_running_d1_uncomp.avi",SEQUENCE_LENGTH)

```

```

➡ 1/1 [=====] - 0s 49ms/step
Action Predicted: running
Confidence: 0.9962002635002136

```

```

✓ 1s [33] predict_single_action("/content/minor/walking/person01_walking_d1_uncomp.avi",SEQUENCE_LENGTH)

```

```

➡ 1/1 [=====] - 0s 48ms/step
Action Predicted: walking
Confidence: 0.9910353422164917

```

*Fig 15: Predicting Single Actions from the video*

```

def predict_on_video(video_file_path, output_file_path, SEQUENCE_LENGTH):
    """
    This function will perform action recognition on a video using the LRCN model.
    Args:
    video_file_path: The path of the video stored in the disk on which the action recognition is to be performed.
    output_file_path: The path where the output video with the predicted action being performed overlayed will be stored.
    SEQUENCE_LENGTH: The fixed number of frames of a video that can be passed to the model as one sequence.
    """

    # Initialize the VideoCapture object to read from the video file.
    video_reader = cv2.VideoCapture(video_file_path)

    # Get the width and height of the video.
    original_video_width = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))
    original_video_height = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # Initialize the VideoWriter Object to store the output video in the disk.
    video_writer = cv2.VideoWriter(output_file_path, cv2.VideoWriter_fourcc(*'DIVX'),
                                   video_reader.get(cv2.CAP_PROP_FPS), (original_video_width, original_video_height))

    # Declare a queue to store video frames.
    frames_queue = deque(maxlen = SEQUENCE_LENGTH)

    # Initialize a variable to store the predicted action being performed in the video.
    predicted_class_name = ''

    # Iterate until the video is accessed successfully.
    while video_reader.isOpened():

        # Read the frame.
        ok, frame = video_reader.read()

        # Check if frame is not read properly then break the loop.
        if not ok:
            break

        # Resize the Frame to fixed Dimensions.
        resized_frame = cv2.resize(frame, (IMAGE_HEIGHT, IMAGE_WIDTH))

        # Normalize the resized frame by dividing it with 255 so that each pixel value then lies between 0 and 1.
        normalized_frame = resized_frame / 255

        # Appending the pre-processed frame into the frames list.
        frames_queue.append(normalized_frame)

        # Check if the number of frames in the queue are equal to the fixed sequence length.
        if len(frames_queue) == SEQUENCE_LENGTH:

            # Pass the normalized frames to the model and get the predicted probabilities.
            predicted_labels_probabilities = model.predict(np.expand_dims(frames_queue, axis = 0))[0]

            # Get the index of class with highest probability.
            predicted_label = np.argmax(predicted_labels_probabilities)

            # Get the class name using the retrieved index.
            predicted_class_name = CLASSES_LIST[predicted_label]

        # Write predicted class name on top of the frame.
        cv2.putText(frame, predicted_class_name, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

        # Write The frame into the disk using the VideoWriter Object.
        video_writer.write(frame)

```



```

# Release the VideoCapture and VideoWriter objects.
video_reader.release()
video_writer.release()

```

```

[36] %time
predict_on_video("/content/drive/MyDrive/videoplayback.mp4", "Human-Activity-Prediction.avi", SEQUENCE_LENGTH)

```

```

1/1 [=====] - 0s 91ms/step
1/1 [=====] - 0s 78ms/step
1/1 [=====] - 0s 85ms/step
1/1 [=====] - 0s 84ms/step
1/1 [=====] - 0s 79ms/step
1/1 [=====] - 0s 87ms/step
1/1 [=====] - 0s 81ms/step
1/1 [=====] - 0s 82ms/step
1/1 [=====] - 0s 87ms/step
1/1 [=====] - 0s 82ms/step
1/1 [=====] - 0s 84ms/step
1/1 [=====] - 0s 81ms/step
1/1 [=====] - 0s 81ms/step
1/1 [=====] - 0s 220ms/step
1/1 [=====] - 0s 76ms/step
1/1 [=====] - 0s 92ms/step
1/1 [=====] - 0s 78ms/step

```

```

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

```

```

Moviepy - Building video __temp__.mp4.
Moviepy - Writing video __temp__.mp4

```

Fig 16: Performing Action recognition using the LRCN Model

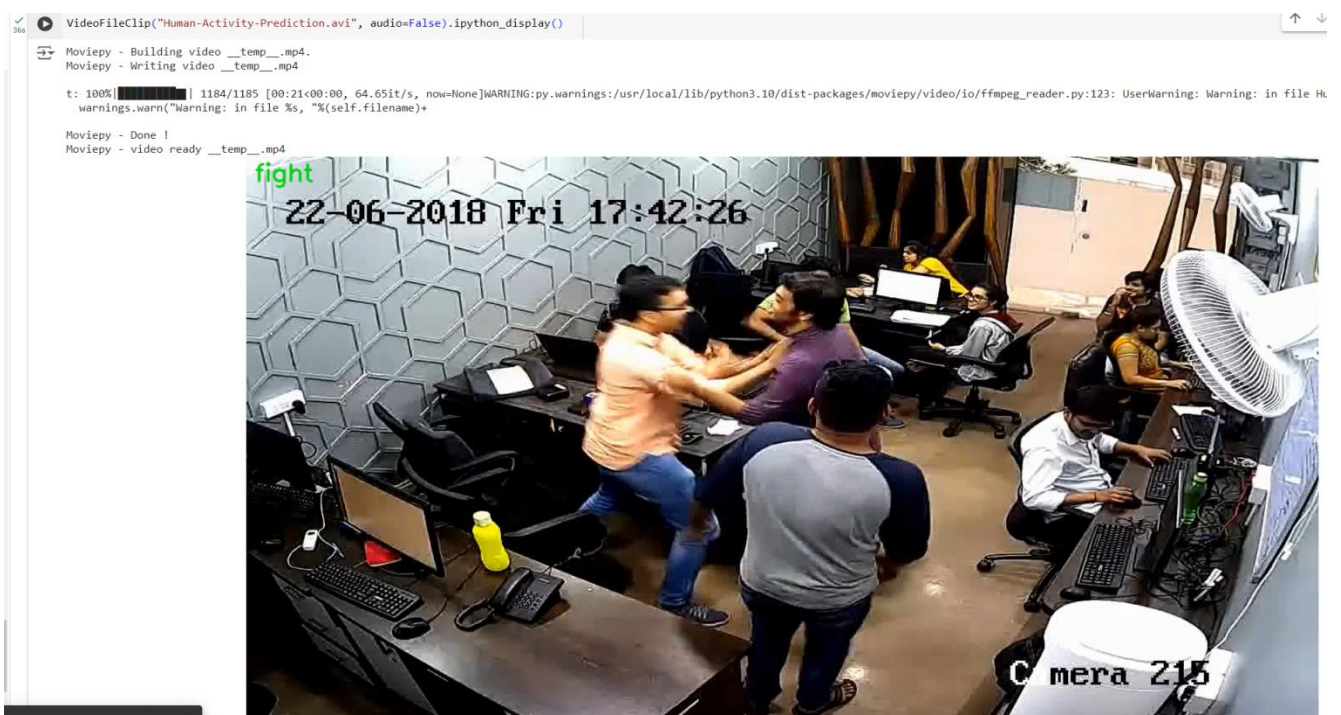


Fig 17: Model Recognises the Action (i.e) Fight in above video



## **Chapter 04: RESULTS**

### **4.1 Discussion on the Results Achieved**

The proposed model successfully achieves an 84% accuracy in detecting anomalous behavior such as fighting and normal activities like walking and running. This result is significant considering the challenges posed by real-time video surveillance, which involves large data volumes and the need for rapid processing. Performance was significantly improved by switching from the VGG-16 with LSTM to a specially designed Long-term Recurrent Convolutional Network (LRCN) architecture. With 12 layers as opposed to VGG-16's 19 layers, the bespoke LRCN model has shorter training and prediction timeframes, which makes it appropriate for real-time applications.

The reduction in frame size from 224x224 pixels to 64x64 pixels also contributed to more efficient data processing and storage, allowing the entire dataset to be loaded even on low-end devices. This change enabled the model to handle more data without sacrificing performance. Despite a slight drop in accuracy compared to the VGG-16 model (from 85% to 82%), the significant gains in speed and resource efficiency make the LRCN model more practical for real-life surveillance scenarios. Overall, the improvements in training and prediction time, coupled with the ability to process videos in near real-time, underscore the model's potential for practical deployment in various security-sensitive environments.

### **4.2 Application of the Minor Project**

The application of the minor project, "Suspicious Activity Detection from Surveillance Video," holds significant potential in enhancing security measures across various sectors and environments. One key application lies in bolstering safety protocols within educational institutions, particularly university campuses and academic facilities. By implementing the proposed surveillance system, administrators can effectively monitor student activities to prevent theft, vandalism, and other suspicious behaviors.

Furthermore, during examination periods, the system can play a crucial role in maintaining academic integrity by identifying and deterring any irregularities or cheating attempts within examination halls. This proactive approach helps ensure fair assessment processes and upholds the credibility of academic evaluations.

Additionally, the system can be installed in vulnerable locations like parking lots, malls,

hospitals, and transportation hubs like bus and train stations to reduce the danger of terrorism, criminal activity, and public safety issues.

By continuously monitoring video feeds and promptly alerting security personnel to any suspicious events, the system contributes to proactive threat prevention and emergency response strategies.

Overall, the application of this minor project extends to diverse settings where enhanced surveillance and rapid response capabilities are essential for safeguarding assets, infrastructure, and public well-being.

### **4.3 Limitation of the Minor Project**

The proposed project's dependence on pre-existing datasets—such as the KTH dataset for running and walking detection and the Kaggle dataset for fighting detection—is one of its limitations. These datasets offer useful training examples, but they might not accurately capture the range of real-world situations seen in surveillance video.

As a result, the model's performance may be limited when applied to novel or unforeseen situations not covered by the training data.

Additionally, the proposed system focuses primarily on detecting specific classes of activities, such as fighting or normal behavior, within a controlled environment like a university campus. This narrow scope may restrict its applicability to broader surveillance contexts with more complex and varied behaviors, such as crowded public spaces or dynamic urban environments.

Furthermore, the system's reliance on CCTV footage assumes consistent and high-quality video feeds, which may not always be the case in real-world deployment scenarios. Factors like poor lighting conditions, occlusions, or camera malfunctions could degrade the system's performance and reliability, leading to potential false alarms or missed detections.

### **4.4 Future Work**

Moving forward, several avenues for enhancement and expansion exist to further improve the efficiency, accuracy, and applicability of the proposed suspicious activity detection system from surveillance video.

**1. Integration of Multimodal Data:** Incorporating additional sensors such as audio and thermal cameras can provide complementary information, enabling more robust detection of suspicious

activities. Integration with other surveillance technologies like drones or satellite imagery can extend monitoring capabilities beyond fixed locations.

**2. Enhanced Real-Time Analysis:** Further optimization of algorithms and hardware acceleration techniques can reduce processing time, enabling truly real-time detection and response to suspicious events. Utilizing edge computing and distributed processing can alleviate the computational burden on centralized systems.

**3. Continuous Model Refinement:** Continuous training and fine-tuning of the detection model with updated datasets can improve accuracy and adaptability to evolving threats and environmental conditions. Implementing techniques like transfer learning and online learning can facilitate ongoing model improvement.

**4. Behavioral Pattern Analysis:** Implementing advanced behavioral analytics algorithms can enable the system to recognize complex patterns of behavior, facilitating proactive threat detection and prediction. Incorporating anomaly detection techniques can identify deviations from normal behavior more effectively.

**5. Scalability and Deployment:** Designing the system for scalability to handle larger datasets and deployment across diverse environments is essential. Developing user-friendly interfaces and integration with existing security infrastructure can streamline deployment and adoption in various settings.

**6. Ethical and Privacy Considerations:** Addressing ethical concerns and privacy implications is crucial. Implementing robust data anonymization techniques and adhering to regulatory frameworks ensure the system respects individual privacy rights while maintaining security standards.

By addressing these areas in future research and development, the proposed system can evolve into a comprehensive and effective tool for enhancing security and safety in various public and private spaces.

## References

1. P. Bhagya Divya, S. Shalini, R. Deepa, and B. S. Reddy, "Inspection of suspicious human activity in the crowdsourced areas captured in surveillance cameras," \*International Research Journal of Engineering and Technology (IRJET)\*, vol. 4, no. 12, Dec. 2017.
2. J. Musale, A. Gavhane, L. Shaikh, P. Hagwane, and S. 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)\*, vol. 5, no. 12, Dec. 2017.
3. E. Scaria, A. A. T, and E. Isaac, "Suspicious activity detection in surveillance video using discriminative deep belief network," \*International Journal of Control Theory and Applications\*, vol. 10, no. 29, pp. 133-139, 2017.
4. D. J. Samuel R, F. E, G. Manogaran, V. G. N, T. Thanjaivadivel, J. S, and A. Ahilan, "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\*, vol. 151, pp. 35-42, 2019.