# Real Time Anomaly Detection in CCTV Surveillance

**MINOR PROJECT REPORT**

**Submitted in partial fulfilment of the requirements for the award of the degree**

**of**

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE & ENGINEERING**

**by**

| | | |
|---|---|---|
| **AVDHESH CHAUDHARY** | **AMAR SINHA** | **CHIRAG AGGARWAL** |
| **EN. No: 02315002719** | **EN. No: 04415002719** | **EN. No: 04615002719** |

**Guided by**

**Mr. Vikrant Shokeen**

**Assistant Professor,CSE**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**MAHARAJA SURAJMAL INSTITUTE OF TECHNOLOGY**

**(AFFILIATED TO GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY, DELHI)**

**DELHI – 110058**

**2019-23**

# CANDIDATE’S DECLARATION

It is hereby certified that the work which is being presented in the B. Tech Minor Project Report entitled **"Real Time Anomaly Detection in CCTV Surveillance"** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** and submitted in the **Department of Computer Science & Engineering of MAHARAJA SURAJMAL INSTITUTE OF TECHNOLOGY, New Delhi (Affiliated to Guru Gobind Singh Indraprastha University, Delhi)** is an authentic record of our own work carried out during a period from **September 2022 to December 2022** under the guidance of **Mr. Vikrant Shokeen, Assistant Professor ,CSE.**

The matter presented in the B. Tech Minor Project Report has not been submitted by me for the award of any other degree of this or any other Institute.

| | | |
|---|---|---|
| **AVDHESH CHAUDHARY** | **AMAR SINHA** | **CHIRAG AGGARWAL** |
| **EN. No: 02315002719** | **EN. No: 04415002719** | **EN. No: 04615002719** |

# CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge. He/She/They are permitted to appear in the External Minor Project Examination

| | |
|---|---|
| **(Mr. Vikrant Shokeen)** | **(Dr. Rinky Dwivedi)** |
| **Assistant Professor,CSE** | **HOD,CSE** |

# ACKNOWLEDGEMENT

We express our deep gratitude to **Mr. Vikrant Shokeen**, Assistant Professor, Department of Computer Science & Engineering for his valuable guidance and suggestion throughout my project work. We are thankful to Name of Project Coordinators, Project Coordinators, for their valuable guidance.

We would like to extend my sincere thanks to **Head of the Department, Dr. Rinky Dwivedi** for his time to time suggestions to complete my project work. I am also thankful to **Col(Dr.)Ranjit Singh, Principal** for providing me the facilities to carry out my project work.

| | | |
|---|---|---|
| **AVDHESH CHAUDHARY** | **AMAR SINHA** | **CHIRAG AGGARWAL** |
| **EN. No: 02315002719** | **EN. No: 04415002719** | **EN. No: 04615002719** |

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ABSTRACT

Nowadays, there has been a rise in the amount of disruptive and offensive activities that have been happening. Due to this, security has been given principal significance. Public places like shopping centres, avenues, banks, etc are increasingly being equipped with CCTVs to guarantee the security of individuals. Subsequently, this inconvenience is making a need to computerize this system with high accuracy. Since constant observation of these surveillance cameras by humans is a near-impossible task. It requires workforces and their constant attention to judge if the captured activities are anomalous or suspicious. Hence, this drawback is creating a need to automate this process with high accuracy. Moreover, there is a need to display which frame and which parts of the recording contain the uncommon activity which helps the quicker judgment of that unordinary action being unusual or suspicious. Therefore, to reduce the wastage of time and labour, we are utilizing deep learning algorithms for Automating Threat Recognition System. Its goal is to automatically identify signs of aggression and violence in real-time, which filters out irregularities from normal patterns. We intend to utilize different Deep Learning models (CNN and RNN) to identify and classify levels of high movement in the frame. From there, we can raise a detection alert for the situation of a threat, indicating the suspicious activities at an instance of time

# CHAPTER 1: INTRODUCTION

## 1.0 INTRODUCTION

Presently, there has been an increase in the number of offensive or disruptive activities that have been taking placethese days. Due to this, security has been given uttermost importance lately. Installation of CCTVs for constant monitoring of people and their interactions is a very common practice in most of the organizations and fields. For a developed country with a population of millions, every person is captured by a camera many times a day. A lot of videos are generated and stored for a certain time duration. Since constant monitoring of these surveillance videos by the authorities to judge if the events are suspicious or not is nearly an impossible task as it requires a workforce and their constant attention. Hence, we are creating a need to automate this process with high accuracy. Moreover, there is a need to show in which frame and which parts of it contain the unusual activity which aids the faster judgment of that unusual activity being abnormal or suspicious. This will help the concerned authorities to identify the main cause of the anomalies occurred meanwhile saving time and labour required in searching the recordings manually.

Anomaly Recognition System is defined as a real-time surveillance program designed to automatically detect and account for the signs of offensive or disruptive activities immediately. This work plan to use different Deep Learning models to detect and classify levels of high movement in the frame. In this work, videos are categorized into segments. From there, a detection alert is raised in the case of a threat, indicating the suspicious activities at an instance of time. In this work, the videos are classified into two categories: Threat (anomalous activities) and Safe (normal activities). Further, we recognize each of the 12 anomalous activities - Abuse, Burglar, Explosion, Shooting, Fighting, Shoplifting, Road Accidents, Arson, Robbery, Stealing, Assault, and Vandalism. These anomalies would provide better security to the individuals.

To solve the above-mentioned problem, deep learning techniques are used which would create phenomenal results in the detection of the activities and their categorization. Here, two Different Neural Networks: CNN [3] and RNN [4] have been used. CNN is the basic neural network that is being used primarily for extracting advanced feature maps from the available recordings. This extraction of high-level feature maps alleviates the complexity of the input. To apply the technique of transfer learning, we use InceptionV3- a pre-trained model. The inceptionV3, pre-trained is selected by keeping in view that the modern models used for object recognition consider loads of parameters and thus take an enormous amount of time in to completely train it. However, the approach of transfer learning would enhance this task by considering initially the previously learned model for some set of classified inputs e.g. ImageNet; which further can be re-trained based on the new weights assigned to various new classes. The output of CNN is fed to the RNN as input. RNN has one additional capability of predicting the next item in a sequence. Therefore, it essentially acts as a forecasting engine. Providing the sense to the captured sequence of actions/movements in the recordings is the motivation behind using this neural network in this work. This network is having an LSTM cell in the primary layer, trailed by some hidden layers with appropriate activation functions, and the output layer will give the final classification of the video into the 13 groups (12 anomalies and 1 normal). The output of this system is used to perform real-time surveillance on the CCTV cameras of different organisations to avoid and detect any suspicious activity. Hence, the time complexity is reduced to a great extent

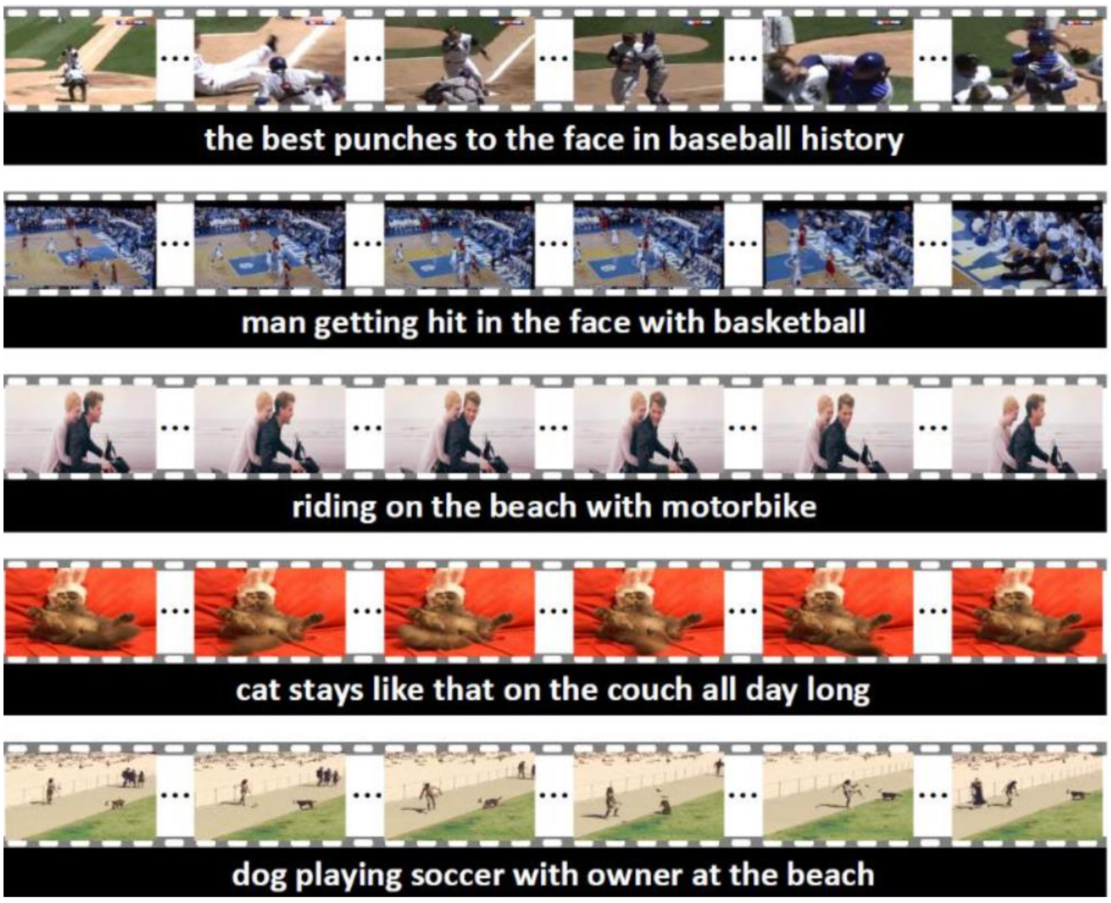

the best punches to the face in baseball history
man getting hit in the face with basketball
riding on the beach with motorbike
cat stays like that on the couch all day long
dog playing soccer with owner at the beach

**Figure 1. This figure is from Show, attend and tell visualization**

## 1.1 Architecture

Anomaly Recognition System consists of a design composed of convolutional and recurrent neural networks.

• The first neural network is convolutional, which has been utilized to obtain the high-level feature maps of the images. This will reduce the intricacy of the input for the second neural net. We are utilizing a pre-trained model called inceptionV3 created by Google. This model applies transfer learning [20] as a widely used object identification models. This has several parameters and can require a large period of time to train completely. Henceforth, Transfer learning utilizes a previously learned model that simplifies loads of this work. The model has learned for various classes like ImageNet which is then re-trained for the weights of new classes.

• The second neural network is utilized as a recurrent neural net to extract meaning from the chain of the actions portrayed in a fixed time duration. This model will be used to classify the segments of videos as a threat and safe.

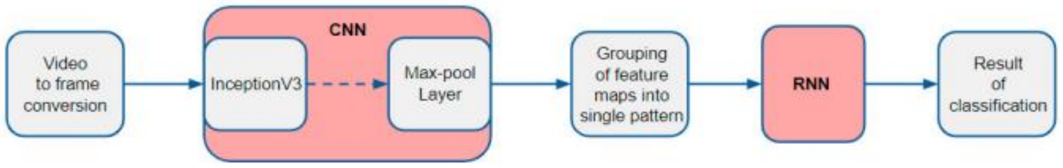
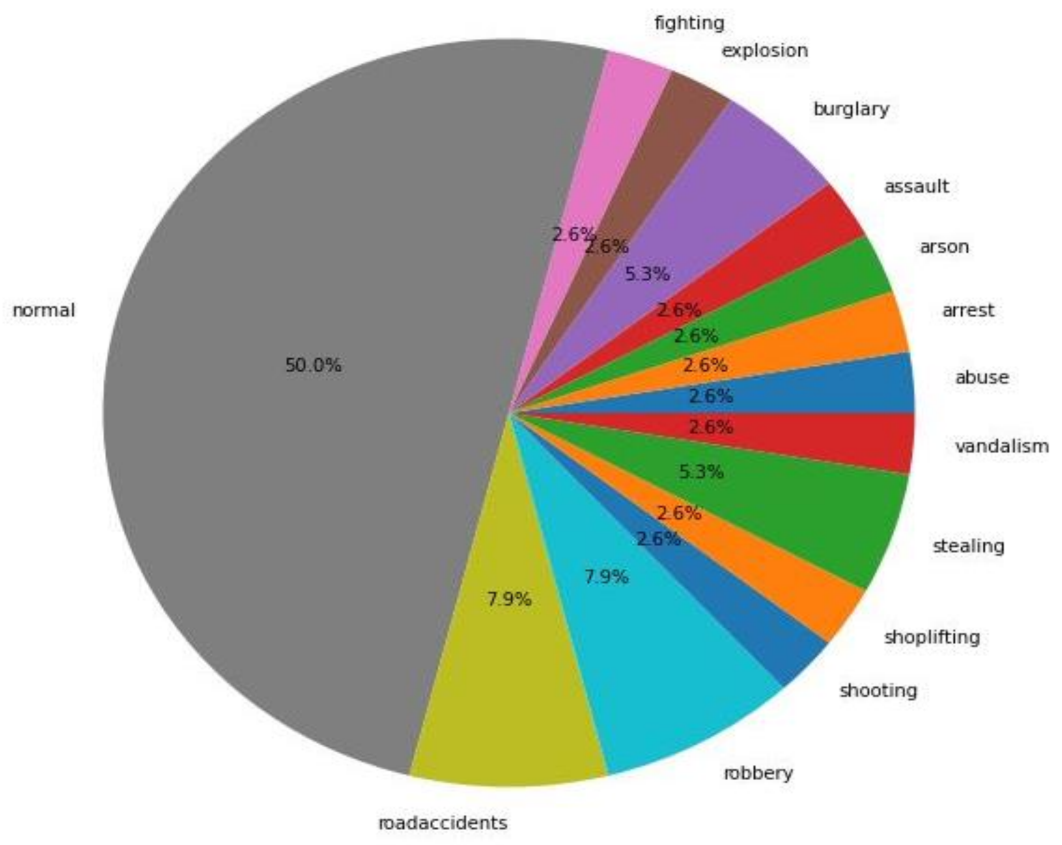

Video to frame conversion
CNN
InceptionV3
Max-pool Layer
Grouping of feature maps into single pattern
RNN
Result of classification

**Figure 2. Workflow of Anomaly Recognition System**

### 1.1.1 Dataset

Text queries have been used to web scrap these videos from websites like LiveLeak and Youtube with minor alterations (for example “gunfire”, “public shootout”) for each anomaly, individually. The text queries in different languages are used to increase the dataset. Recordings that are manually altered, hoax recordings, news collected, the non-CCTV camera captured, or captured by a portable recording camera and containing aggregation are expelled from the dataset. The recordings in which the anomaly isn't clear have been removed too. To guarantee the quality of this dataset, 10 trained annotators are utilized with each having varying degrees of expertise in computer vision. Following the above constraints, we have gathered 950 unaltered real-world surveillance videos containing anomalies. Moreover, 940 normal scenarios are also collected, making a total of 1800 videos in this dataset.

**Pie Chart 1. Share of Different Types of Training Videos**



## 1.1.2 Preprocessing Testing set

Unnecessary footage like advertisements, inactivities and looped frames have been manually trimmed off from each video to reduce the size of the dataset and hence increasing the processing speed. The videos are also flipped horizontally to double the video counts to lower the overfitting in the training process. This is eventually increasing the effectiveness of the training process of the model. The entire dataset has been divided in the ratio 3:1 as training and validation set during the training of our model. The normal videos are shuffled with the videos containing the anomalies. Finally, these videos are passed through the pre-trained InceptionV3 model for dimension reduction.

## 1.1.3 Video to Frame Conversion

Extracting frames from the captured CCTV recordings is the first step of this approach. The work extracts the frame after a fixed and small interval of time (say 1 sec). This extracted frame is then resized to the dimension 299x299 pixels which are

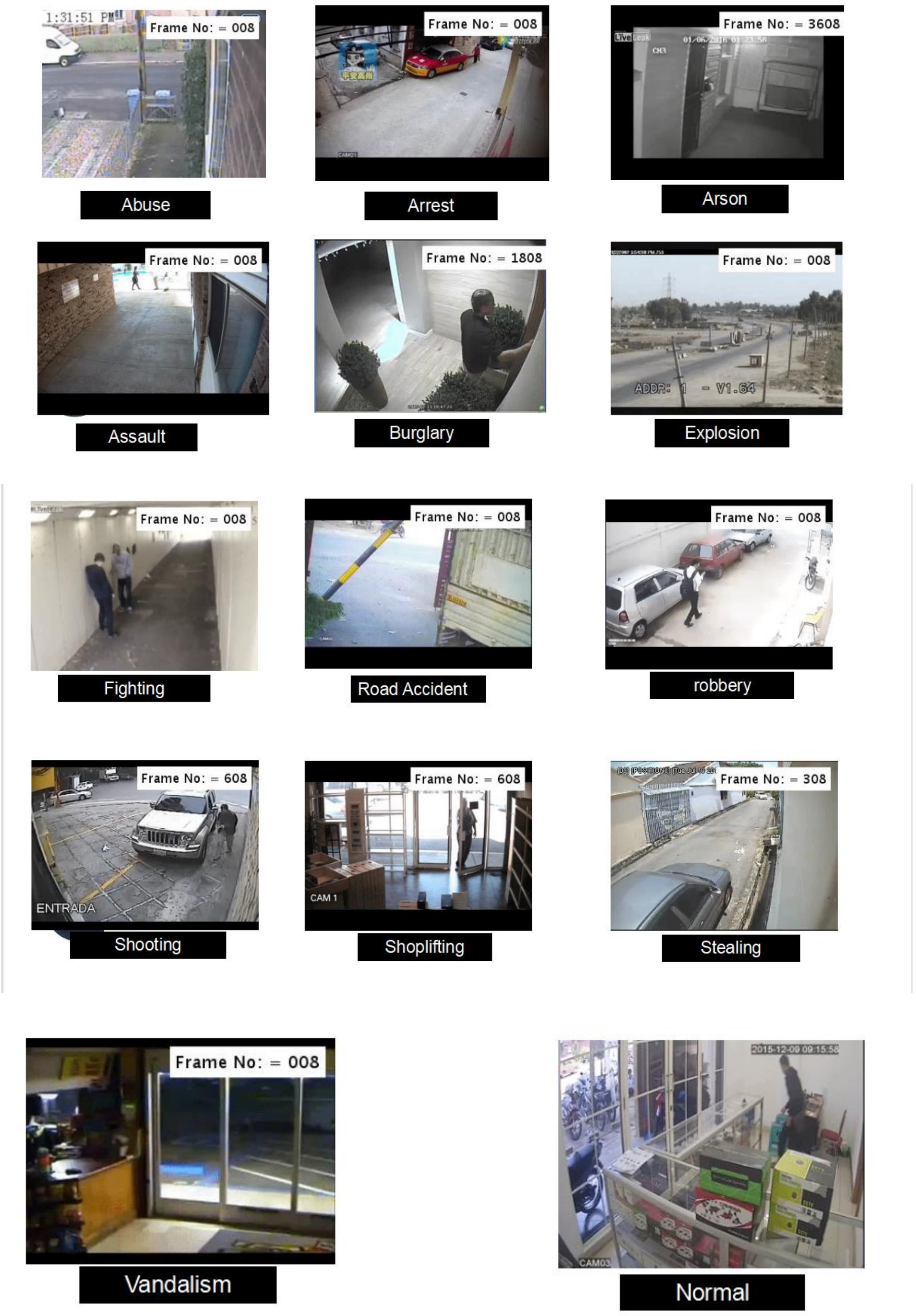


**Figure 3. Videos of 14 Malicious Activities**

the standard input dimensions for InceptionV3. The preprocess_input function is meant to adequate the resized image to the format that the model requires.

### 1.1.4 InceptionV3:

InceptionV3 is learned on the ImageNet dataset. It is a large dataset released in Visual Recognition competition. The model attempts to classify entire dataset into 1,000 categories, which is usually done in Computer Vision. The model concentrates general features from input pictures within the initial half. Later, it classifies those images based on the extracted features in the second half. The layers in the standard inception model are explained thoroughly

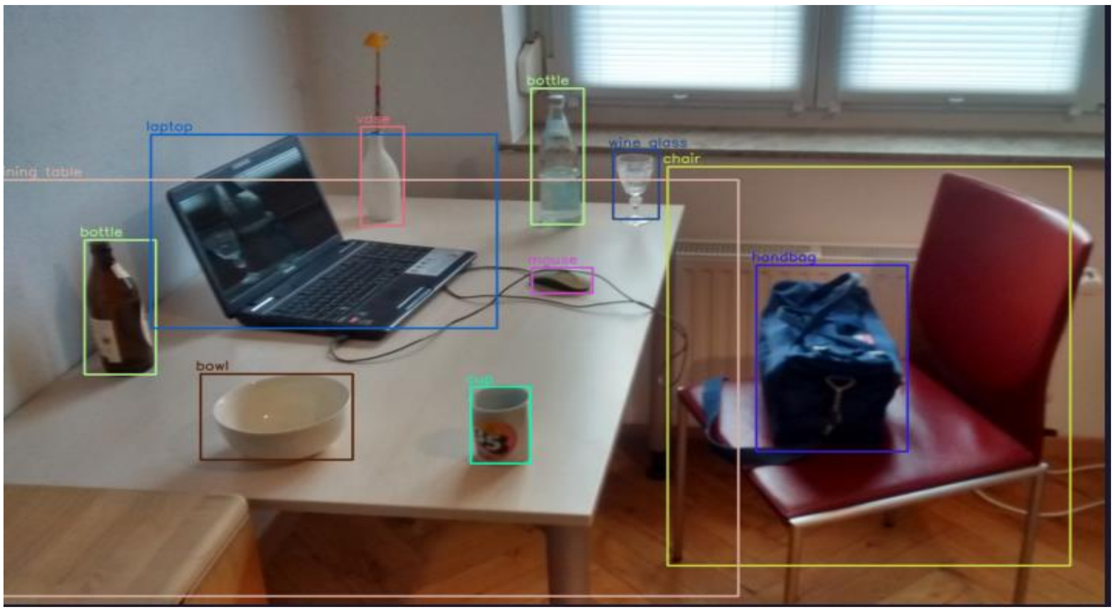


**Figure 3. Object detection using inception V3**

### 1.1.5 Convolution Neural Network

The transfer Learning is used to train our CNN with the already trained InceptionV3 model. In transfer learning, we employ the feature extraction part to the new model and re-train the classification part with our original dataset. Since we don't need to learn the feature extraction part (which is a very complex part of the model), the overall learning process requires less computational resources and training time. The output of the Inception model is passed to the input of the CNN which isn't the final classification model. Rather, the outcome of the last pooling layer is extracted which is a vector containing 2,048 features to feed as an input to RNN. The vector is referred to as a high-level feature map.

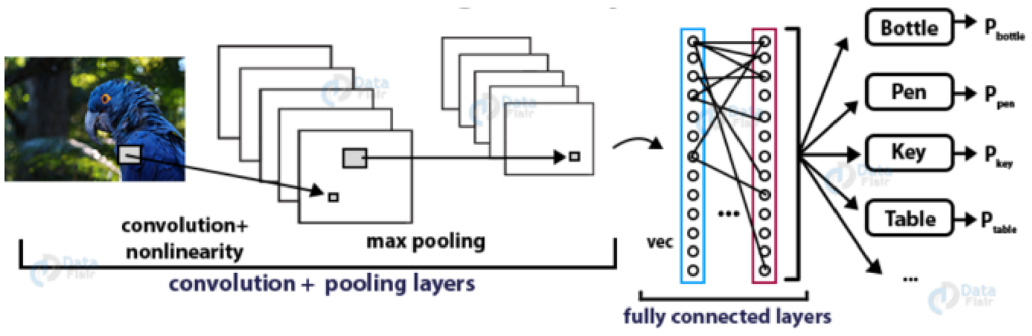


**Figure 3. A Working Of Deep CNN**

### 1.1.6 Grouping of feature maps into a single pattern

To give the framework a sense of the sequence, multiple prepossessed frames are considered. This chunk is then used to make the final classification. A chunk of these frames classifies a temporal segment of the video and can provide a sense of motion. For this, some feature maps are stored which are predicted by the inception model (CNN), generated in that fixed period of the video. Low-level features have been considered to generate a high-level feature map. These features are used for finding shapes and objects in computer images. This single combined feature map is then passed to the RNN. The reason to pass the feature map instead of the frame itself is to reduce the training complexity of the RNN

### 1.1.7 Recurrent Neural Network

The input of the second neural network is the concatenated collection of highlevel feature maps generated in the previous step. This network has an LSTM cell with 5,727 neurons in the primary layer. This layer is followed by 2 Hidden layers. The first hidden layer contains 1,024 neurons with Relu as the activation function while the following layer has 50 neurons with sigmoid as the activation function. The actual probabilistic classification of the framework is produced the final layer having thirteen neurons with softmax as the activation function

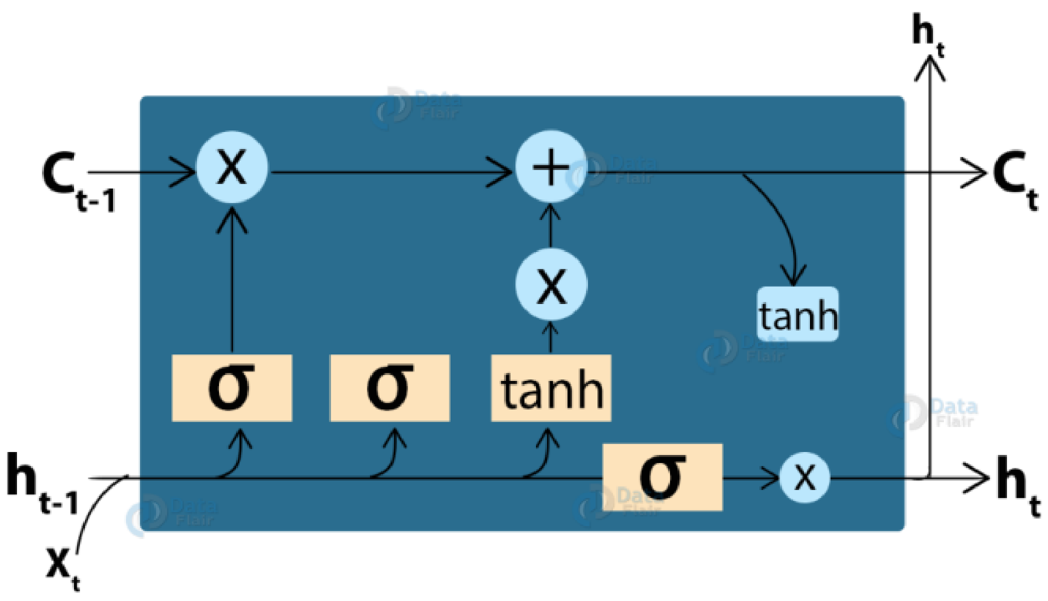


**Figure 3. A LSTM Cell Structure**

## 2.0 MOTIVATION

A lot of work has been done on "The Real Time CCTV Surveillance using Deep Learning Algorithms" with a wide range of different approaches. We went through a lot of Research papers to decide on this approach.

Firstly, the work on "AI Image Caption Bot" - which takes an image and produces a caption for that image – using Supervised Learning opened the doorway for Computer Vision enthusiasts to apply the Neural Network approach to Videos and thus CCTV surveillance.

Some approaches to Action Recognition in Video Surveillance for Threat Detection include: i) Detecting a specific Anomalous Event, like a Traffic Accident. It has a lack of generalization.

ii) Sparse Coding Based Approaches, in this approach we assume initial frames of video contains normal events for building normal event dictionary and the anomaly is detected when there is an unexpected change in the frames. This system ends up giving too many false alarms and does not handle environment changes very well.

iii) Prediction Based Model, in this approach we use a future video frame prediction-based anomaly detection method. If a frame agrees with its prediction, it potentially corresponds to a normal event. Otherwise, it potentially corresponds to an anomalous event. But it faces constraint on intensity and gradient, optical flow gloss.

The approach we are going to use in this project is the Classification Based Approach and we will use a CNN and RNN to classify the video feed frames into different categories using Object detection. From the so far literature review, it has been observed that the maximum number of researches have designed methodologies for learning distribution of ordinary movements from the training done using available recordings.

Deep learning has resulted in being best for image classification and hence, is found suitable for video activity classification.

## 3.0 OBJECTIVE

The main objective of our project is to develop real time Anomaly detection in CCTV surveillance.

**Objectives Pointwise:**

1. To reduce probability of error in anomaly detection in CCTV surveillance.
2. To reduce time of finding the video segment in which anomalous activities happen by making the process real time.
3. To increase accuracy of automatic threat detection in CCTV surveillance.
4. To increase the reliability of the system by making it more generalized and training it on 12 anomalous activities videos data. 5. To increase operational efficiency

## 4.0 SUMMARY OF THE REPORT

CCTV surveillance systems are commonly used to ensure the safety and security of public and private spaces. However, manual monitoring of surveillance footage can be tedious and time-consuming, making it difficult to promptly identify and respond to potential threats. In this paper, we present a real-time threat detection system for CCTV surveillance that utilizes deep learning models to detect and classify levels of high movement in video frames. By treating videos as segments and defining anomalous (threatening) and normal (safe) segments, our system is able to continuously monitor surveillance footage in real-time and identify potential threats, such as abuse, burglaries, explosions, shootings, fighting, shoplifting, road accidents, arson, robbery, stealing, assault, and vandalism. To evaluate the performance of our system, we conducted extensive experiments on a large dataset of CCTV footage and achieved promising results. Our system has the potential to significantly improve the efficiency and effectiveness of CCTV surveillance, enabling faster response times and enhanced security for individuals.

In our approach, we consider normal and anomalous videos as bags and video segments as instances in multiple instance learning (MIL), and automatically learn a deep anomaly ranking model that predicts high anomaly scores for anomalous video segments. Furthermore, we introduce sparsity and temporal smoothness constraints in the ranking loss function to better localize anomaly during training. We also introduce a new large-scale first of its kind dataset of 128 hours of videos. It consists of 1900 long and untrimmed real-world surveillance videos, with 13 realistic anomalies such as fighting, road accident, burglary, robbery, etc. as well as normal activities. This dataset can be used for two tasks. First, general anomaly detection considering all anomalies in one group and all normal activities in another group. Second, for recognizing each of 13 anomalous activities. Our experimental results show that our MIL method for anomaly detection achieves significant improvement on anomaly detection performance as compared to the state-of-the-art approaches. We provide the results of several recent deep learning baselines on anomalous activity recognition. The low recognition performance of these baselines reveals that our dataset is very challenging and opens more opportunities for future work.

# CHAPTER 2: PROJECT DESIGN & IMPLEMENTATION

## 2.1 Data Sources

The Dataset Below is Divided in to Previous and Current Dataset

## 2.1. Previous datasets

We briefly review the existing video anomaly detection datasets in this section. The UMN dataset [2] consists of five different staged videos, where people walk around and after some time start running in different directions. The anomaly is characterized by only running action. UCSD Ped1 and Ped2 datasets [27] contain 70 and 28 surveillance videos, respectively. Those videos are captured at only one location. The anomalies in the videos are simple and do not reflect realistic anomalies in video surveillance, e.g. people walking across a walkway, non pedestrian entities (skater, biker and wheelchair) in the walkways. Avenue dataset [28] consists of 37 videos. Although it contains more anomalies, they are staged and captured at one location. Similar to [27], videos in this dataset are short and some of the anomalies are unrealistic (e.g. throwing paper). Subway Exit and Subway Entrance datasets [3] contain one long surveillance video each. The two videos capture simple anomalies such as walking in the wrong direction and skipping payment. BOSS [1] dataset is collected from a surveillance camera mounted in a train. It contains anomalies such as harassment, person with a disease, panic situation, as well as 6482 normal videos. All anomalies are performed by actors. Abnormal Crowd [31] introduced a crowd anomaly dataset which contains 31 videos with crowded scenes only. Overall, the previous datasets for video anomaly detection are small in terms of the number of videos or the length of the video. Variations in abnormalities are also limited. In addition, some anomalies are not realistic.

## 2.2. Our datasets

Due to the limitations of previous datasets, we construct a new large-scale dataset to evaluate our method. It consists of long untrimmed surveillance videos which cover 13 realworld anomalies, including Abuse, Arrest, Arson, Assault, Accident, Burglary, Explosion, Fighting, Robbery, Shooting, Stealing, Shoplifting, and Vandalism. These anomalies are selected because they have a significant impact on public safety. We compare our dataset with previous anomaly detection datasets in Table 1.

**Video collection.** To ensure the quality of our dataset, we train ten annotators (having different levels of computer vision expertise) to collect the dataset. We search videos on YouTube and LiveLeak 1 using text search queries (with slight variations e.g. “car crash”, “road accident”) of each anomaly. In order to retrieve as many videos as possible, we also use text queries in different languages (e.g. French, Russian, Chinese, etc.) for each anomaly, thanks to Google translator. We remove videos which fall into any of the following conditions: manually edited, prank videos, not captured by CCTV cameras, taking from news, captured using a hand-held camera, and containing compilation. We also discard videos in which the anomaly is not clear. With the above video pruning constraints, 950 unedited real-world surveillance videos with clear anomalies are collected. Using the same constraints, 950 normal videos are gathered, leading to a total of 1900 videos in our dataset. In Figure 2, we show four frames of an example video from each anomaly.

**Annotation.** For our anomaly detection method, only video-level labels are required for training. However, in order to evaluate its performance on testing videos, we need to know the temporal annotations, i.e. the start and ending frames of the anomalous event in each testing anomalous video. To this end, we assign the same videos to multiple annotators to label the temporal extent of each anomaly. The final temporal annotations are obtained by averaging annotations of different annotators. The complete dataset is finalized after intense efforts of several months.

**Training and testing sets.** We divide our dataset into two parts: the training set consisting of 800 normal and 810 anomalous videos (details shown in Table 2) and the testing set including the remaining 150 normal and 140 anomalousvideos. Both training and testing sets contain all 13 anomalies at various temporal locations in the videos. Furthermore, some of the videos have multiple anomalies. The distribution of the training videos in terms of length (in minute) is shown in Figures 3. The number of frames and percentage of anomaly in each testing video are presented in Figures 4 and 5, respectively.

```
train_df = pd.read_csv("data/train.csv")
test_df = pd.read_csv("data/test.csv")
```

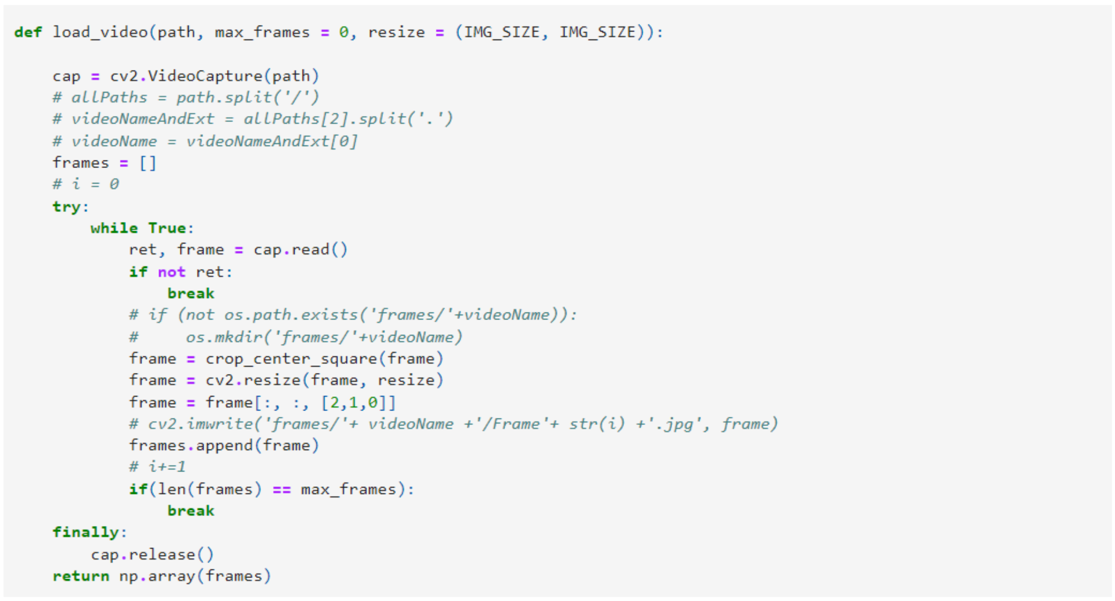
```
print(f"Total videos for training : {len(train_df)}")
print(f"Total videos for testing  : {len(test_df)}")

print("Training Dataframe : ")
print(train_df.head(10))

print("Testing Dataset    : ")
print(test_df.sample(10))
```

```
Total videos for training : 1520
Total videos for testing  : 380
Training Dataframe :
   Unnamed: 0    label                                         video_name
0         479   normal  ../input/real-time-anomaly-detection-in-cctv-s...
1          22    abuse  ../input/real-time-anomaly-detection-in-cctv-s...
2         633   normal  ../input/real-time-anomaly-detection-in-cctv-s...
3          47    abuse  ../input/real-time-anomaly-detection-in-cctv-s...
4          92   arrest  ../input/real-time-anomaly-detection-in-cctv-s...
5         414   normal  ../input/real-time-anomaly-detection-in-cctv-s...
6         106    arson  ../input/real-time-anomaly-detection-in-cctv-s...
7         185  assault  ../input/real-time-anomaly-detection-in-cctv-s...
8         965   normal  ../input/real-time-anomaly-detection-in-cctv-s...
9         481   normal  ../input/real-time-anomaly-detection-in-cctv-s...
```

## 2.2 Feed the videos to a Network

Extracting frames from the captured CCTV recordings is the first step of this approach. The work extracts the frame after a fixed and small interval of time (say 1 sec). This extracted frame is then resized to the dimension 299x299 pixels which are the standard input dimensions for InceptionV3. The preprocess_input function is meant to adequate the resized image to the format that the model requires.

```
IMG_SIZE = 299

def crop_center_square(frame):
    y, x = frame.shape[0:2]
    min_dim = min(y,x)
    # print(f"y : {y}      and      x : {x}")
    start_x = (x // 2) -  (min_dim // 2)
    start_y = (y // 2) -  (min_dim // 2)
    return frame[ start_y : start_y+min_dim, start_x : start_x + min_dim]
```



```python
def load_video(path, max_frames = 0, resize = (IMG_SIZE, IMG_SIZE)):

    cap = cv2.VideoCapture(path)
    # allPaths = path.split('/')
    # videoNameAndExt = allPaths[2].split('.')
    # videoName = videoNameAndExt[0]
    frames = []
    # i = 0
    try:
        while True:
            ret, frame = cap.read()
            if not ret:
                break
            # if (not os.path.exists('frames/'+videoName)):
            #     os.mkdir('frames/'+videoName)
            frame = crop_center_square(frame)
            frame = cv2.resize(frame, resize)
            frame = frame[:, :, [2,1,0]]
            # cv2.imwrite('frames/'+ videoName +'/Frame'+ str(i) +'.jpg', frame)
            frames.append(frame)
            # i+=1
            if(len(frames) == max_frames):
                break
    finally:
        cap.release()
    return np.array(frames)
```

## 2.3 Transfer Learning & Feature Extraction :

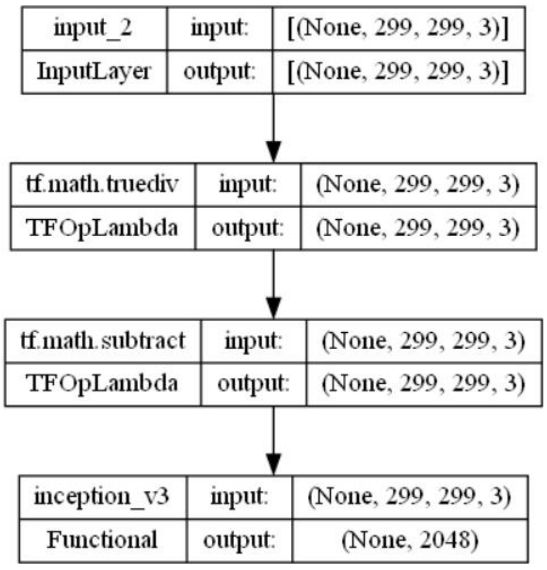
Images are nothing but input (X) to our model. As you may already know that any input to a model must be given in the form of a vector. We need to convert every image into a fixed sized vector which can then be fed as input to the neural network. For this purpose, we opt for transfer learning by using the Inception V3 model (Convolutional Neural Network).Inception V3 is not the first model coming from the ResNet family. The original model was called the  Inception V3 and was another milestone in the CV domain back in 2015. The main motivation behind this model was to avoid poor accuracy as the model went on to become deeper. Additionally, if you are familiar with Gradient Descent, you would have come across the Vanishing Gradient issue – the Inception V3 model aimed to tackle this issue as well. Inception V3 also follows a similar technique with just more layers) The code for this is as follows

```python
def build_feature_extractor():
    feature_extractor = keras.applications.InceptionV3(
        weights="imagenet",
        include_top=False,
        pooling="avg",
        input_shape=(IMG_SIZE, IMG_SIZE, 3),
    )
    preprocess_input = keras.applications.inception_v3.preprocess_input

    inputs = keras.Input((IMG_SIZE, IMG_SIZE, 3))
    preprocessed = preprocess_input(inputs)

    outputs = feature_extractor(preprocessed)
    return keras.Model(inputs, outputs, name="feature_extractor")


feature_extractor = build_feature_extractor()
```

```python
print(feature_extractor.summary())
```

```
Model: "feature_extractor"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 299, 299, 3)]     0         
                                                                 
 tf.math.truediv (TFOpLambda  (None, 299, 299, 3)      0         
 )                                                               
                                                                 
 tf.math.subtract (TFOpLambd  (None, 299, 299, 3)      0         
 a)                                                              
                                                                 
 inception_v3 (Functional)   (None, 2048)              21802784  
                                                                 
=================================================================
Total params: 21,802,784
Trainable params: 21,768,352
Non-trainable params: 34,432
_________________________________________________________________
None
```

```python
plot_model(feature_extractor, to_file='plots/model_plot.png', show_shapes=True, show_layer_names=True)
```



## 2.4 Label Encoding

StringLookup layer encode the class labels as integers.

```
label_processor = keras.layers.StringLookup(num_oov_indices=0, vocabulary=np.unique(train_df["label"]))
print(label_processor.get_vocabulary())

labels = train_df["label"].values
labels = label_processor(labels[..., None]).numpy()
labels
```

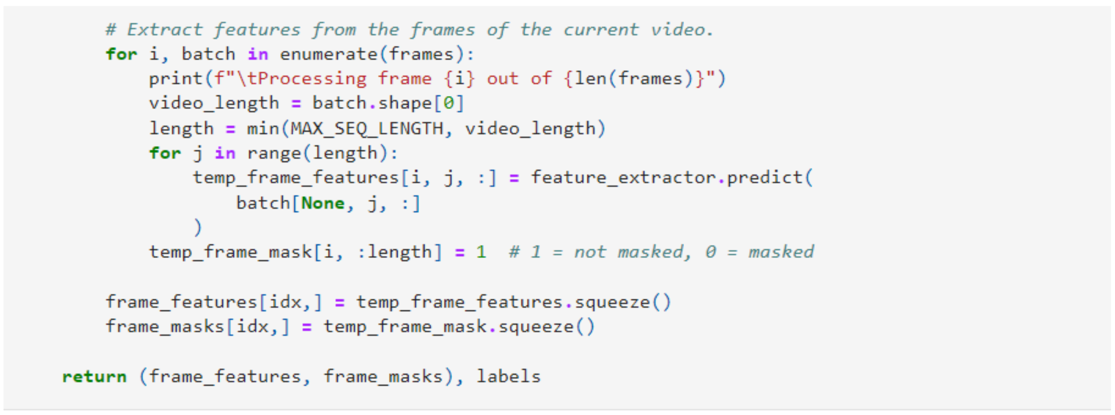
```
['abuse', 'arrest', 'arson', 'assault', 'burglary', 'explosion', 'fighting', 'normal', 'roadaccidents', 'robbery', 'shooting', 'shoplifting', 'stealin
g', 'vandalism']
```

## 2.5 CNN Model Processing

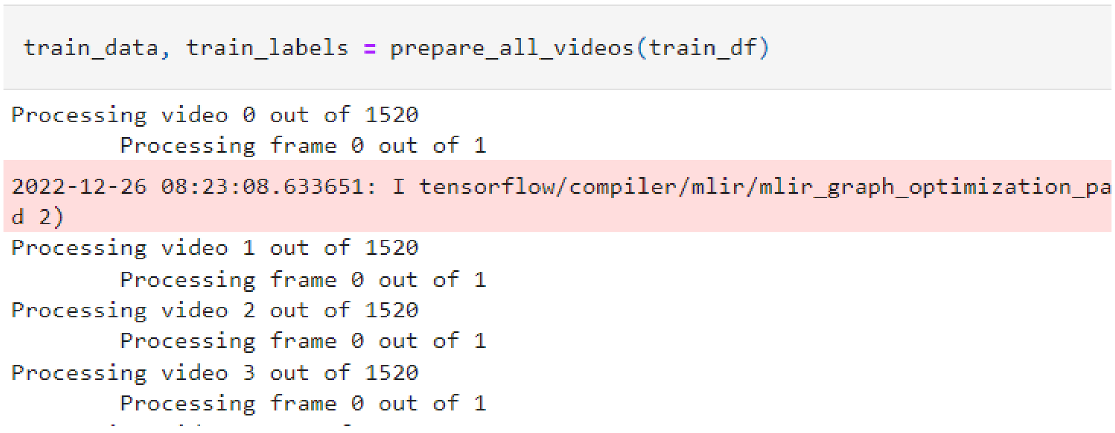
First, we run every frame from every video through Inception, saving the output from the final pool layer of the network. So we effectively chop off the top classification part of the network so that we end up with a 2,048-d vector of features that we can pass to our RNN.

```
#Define hyperparameters

IMG_SIZE = 299
BATCH_SIZE = 64
EPOCHS = 30

MAX_SEQ_LENGTH = 20
NUM_FEATURES = 2048
```

```
def prepare_all_videos(df):
    num_samples = len(df)
    video_paths = df["video_name"].values.tolist()

    ##take all classlabels from train_df column named 'label' and store in labels
    labels = df["label"].values

    #convert classlabels to label encoding
    labels = label_processor(labels[..., None]).numpy()

    # `frame_masks` and `frame_features` are what we will feed to our sequence model.
    # `frame_masks` will contain a bunch of booleans denoting if a timestep is
    # masked with padding or not.
    frame_masks = np.zeros(shape=(num_samples, MAX_SEQ_LENGTH), dtype="bool")
    frame_features = np.zeros(shape=(num_samples, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32")

    # For each video.
    for idx, path in enumerate(video_paths):
        # Gather all its frames and add a batch dimension.
        print(f"Processing video {idx} out of {num_samples}")

        frames = load_video(path)
        frames = frames[None, ...]
        # Initialize placeholders to store the masks and features of the current video.
        temp_frame_mask = np.zeros(shape=(1, MAX_SEQ_LENGTH,), dtype="bool")
        temp_frame_features = np.zeros(
            shape=(1, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32"
        )
```



```
        # Extract features from the frames of the current video.
        for i, batch in enumerate(frames):
            print(f"\tProcessing frame {i} out of {len(frames)}")
            video_length = batch.shape[0]
            length = min(MAX_SEQ_LENGTH, video_length)
            for j in range(length):
                temp_frame_features[i, j, :] = feature_extractor.predict(
                    batch[None, j, :]
                )
            temp_frame_mask[i, :length] = 1  # 1 = not masked, 0 = masked

        frame_features[idx,] = temp_frame_features.squeeze()
        frame_masks[idx,] = temp_frame_mask.squeeze()

    return (frame_features, frame_masks), labels
```



```
train_data, train_labels = prepare_all_videos(train_df)
```

```
Processing video 0 out of 1520
        Processing frame 0 out of 1
2022-12-26 08:23:08.633651: I tensorflow/compiler/mlir/mlir_graph_optimization_pa
d 2)
Processing video 1 out of 1520
        Processing frame 0 out of 1
Processing video 2 out of 1520
        Processing frame 0 out of 1
Processing video 3 out of 1520
        Processing frame 0 out of 1
```

## 2.6 The sequence model

we convert those extracted features into *sequences* of extracted features. If you recall from our constraints, we want to turn each video into a 40-frame sequence. So we stitch the sampled 40 frames together, save that to disk, and now we're ready to train different RNN models without needing to continuously pass our images through the CNN every time we read the same sample or train a new network architecture.

Now, we can feed this data to a sequence model consisting of recurrent layers like GRU.

```
# Utility for our sequence model.
def get_sequence_model():
    class_vocab = label_processor.get_vocabulary()

    frame_features_input = keras.Input((MAX_SEQ_LENGTH, NUM_FEATURES))
    mask_input = keras.Input((MAX_SEQ_LENGTH,), dtype="bool")

    # Refer to the following tutorial to understand the significance of using `mask`:
    # https://keras.io/api/layers/recurrent_layers/gru/
    x = keras.layers.GRU(16, return_sequences=True)(frame_features_input, mask=mask_input)
    x = keras.layers.GRU(8)(x)
    x = keras.layers.Dropout(0.4)(x)
    x = keras.layers.Dense(8, activation="relu")(x)
    output = keras.layers.Dense(len(class_vocab), activation="softmax")(x)

    rnn_model = keras.Model([frame_features_input, mask_input], output)

    rnn_model.compile(
        loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"]
    )
    return rnn_model
```

For the RNN, we use a single, 4096-wide LSTM layer, followed by a 1024 Dense layer, with some dropout in between. This relatively shallow network outperformed all variants where we tried multiple stacked LSTMs.

```
EPOCHS = 30
# Utility for running experiments.
def run_experiment():
    filepath = "./tmp/video_classifier"
    checkpoint = keras.callbacks.ModelCheckpoint(
        filepath, save_weights_only=True, save_best_only=True, verbose=1
    )

    seq_model = get_sequence_model()
    history = seq_model.fit(
        [train_data[0], train_data[1]],
        train_labels,
        validation_split=0.3,
        epochs=EPOCHS,
        callbacks=[checkpoint],
    )

    seq_model.load_weights(filepath)
    _, accuracy = seq_model.evaluate([test_data[0], test_data[1]], test_labels)
    print(f"Test accuracy: {round(accuracy * 100, 2)}%")

    return history, seq_model
```

```
_, sequence_model = run_experiment()
```

```
Epoch 1/10
13/13 [==============================] - 4s 101ms/step - loss: 1.5259 - accuracy: 0.3157 - val_lo
```

```
Epoch 00001: val_loss improved from inf to 1.47325, saving model to /tmp/video_classifier
Epoch 2/10
13/13 [==============================] - 0s 21ms/step - loss: 1.3087 - accuracy: 0.5880 - val_los
```

```
Epoch 00002: val_loss did not improve from 1.47325
Epoch 3/10
13/13 [==============================] - 0s 20ms/step - loss: 1.1532 - accuracy: 0.6795 - val_los
```

## 2.7 Prediction

```python
def prepare_single_video(frames):
    frames = frames[None, ...]
    frame_mask = np.zeros(shape=(1, MAX_SEQ_LENGTH,), dtype="bool")
    frame_features = np.zeros(shape=(1, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32")

    for i, batch in enumerate(frames):
        video_length = batch.shape[0]
        length = min(MAX_SEQ_LENGTH, video_length)
        for j in range(length):
            frame_features[i, j, :] = feature_extractor.predict(batch[None, j, :])
        frame_mask[i, :length] = 1  # 1 = not masked, 0 = masked

    return frame_features, frame_mask


def sequence_prediction(path):
    class_vocab = label_processor.get_vocabulary()

    frames = load_video(os.path.join(path))
    frame_features, frame_mask = prepare_single_video(frames)
    probabilities = sequence_model.predict([frame_features, frame_mask])[0]

    for i in np.argsort(probabilities)[::-1]:
        print(f"  {class_vocab[i]}: {probabilities[i] * 100:5.2f}%")
    return frames
```

```
Test video path: ../input/real-time-anomaly-detection-in-cctv-surveillance/data/normal/Normal_Videos667_x264.mp4
  normal: 10.74%
  stealing:  8.17%
  burglary:  7.91%
  robbery:  7.80%
  vandalism:  7.07%
  fighting:  7.04%
  roadaccidents:  6.92%
  arrest:  6.91%
  arson:  6.83%
  shooting:  6.35%
  explosion:  6.19%
  assault:  6.11%
  abuse:  6.05%
  shoplifting:  5.91%
```

# CHAPTER 3 : RESULT AND DISCUSSION

In this work, we have trained 6 variations of our approach by altering different parameters and refining the dataset. The output layer of the RNN in model 1 has two neurons which are used to classify the entire dataset into 2 categories i.e. threat and safe. The anomalies considered for this model are Abuse, Arrest, Assault, and Arson along with a set of normal videos. The videos used are untrimmed and contain several unwanted footages. There are 940 chunks of un-shuffled frames with each chuck of 30 frames extracted at an interval of 1 second. The optimiser and loss function used for training this model are Adam and mean_squared_error resp. Fig. 4. (a) and (b) shows that the model trained is overfitted and producing fluctuating output with poor testing and training accuracy

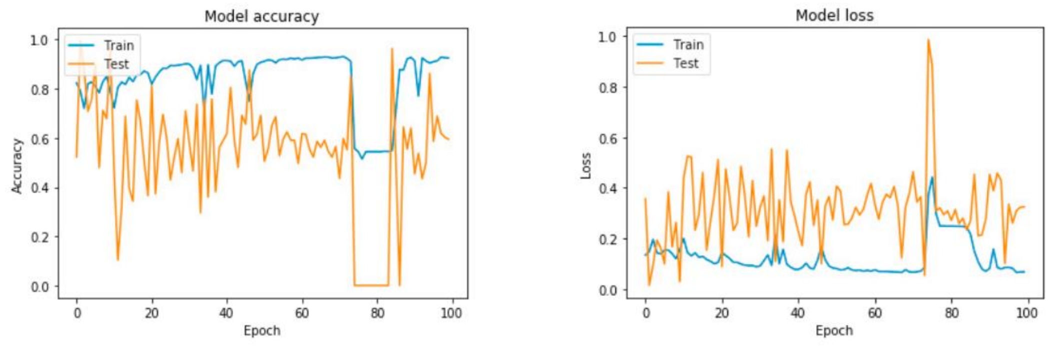


**Fig. 4. (a) Accuracy of Model 1; (b) Loss of Model 1**

To overcome the imperfections caused due to overfitting in model 1, the size of each chunk is reduced and the regularization [21] parameter is set to 0.01. The added term is considered to control the excessively fluctuating function. This prevents the coefficients to take extreme values. This forms the second model, Model 2. Fig. 5. (b) reflects that the loss function of model 2 is better tuned as compared to that of model 1. Moreover, overfitting is reduced to some extent as shown in Fig. 5. (a) producing a comparatively well-performing model.

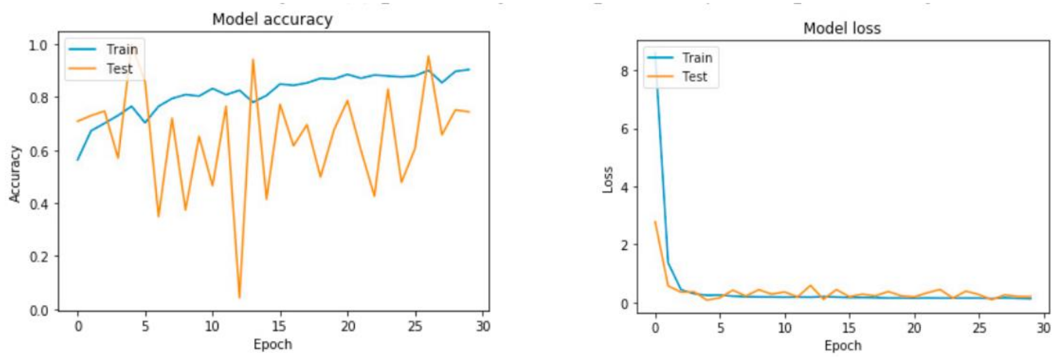


**Fig. 5. (a) Accuracy of Model 2; (b) Loss of Model 2**

Later, model 3 is designed to reduce overfitting to a considerable level. Along with regularization, the dataset is cleaned by manually trimming each video. Trimming is done to exclude the unwanted and useless footage from the videos that are causing improper training of the model. The dataset was shuffled this time which was not done in the case of model 1 and model 2. Moreover, the optimizer for model 3 is changed from Adam with the learning rate of 0.001 to SGD with the learning rate of 0.01.

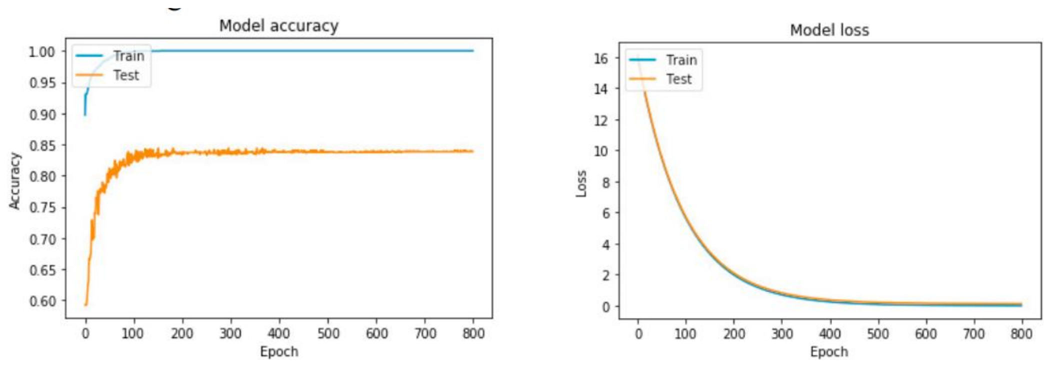


**Fig. 6. (a) Accuracy of Model 3; (b) Loss of Model 3**

Fig. 6. (a) shows that cleaning the dataset and changing the optimizer played an important role in reducing the overfitting in the trained model. But, the accuracy of the model is not as per expectations. Fig. 6. (b) presents a smooth curve for loss function with no fluctuation.

To enhance the accuracy of the model, eight more classes of anomalies; Road Accidents, Burglar, Explosion, Shooting, Fighting, Shoplifting, Robbery, Stealing, and Vandalism are added to the dataset. Also, the size of the chunks is decreased to 8. Model 4 is created to classify the segments of video into 13 different categories rather than 2 to give a better description of the anomaly detected by the algorithm.

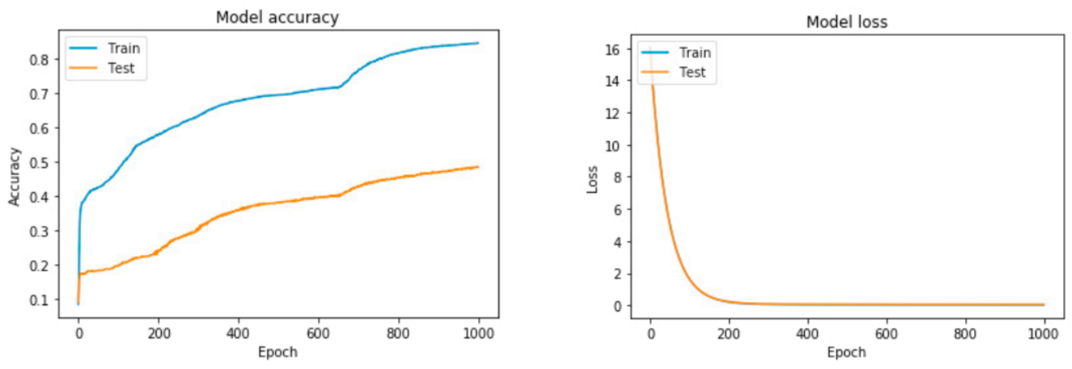


**Fig. 7. (a) Accuracy of Model 4; (b) Loss of Model 4**

Fig. 7. (a) and (b) shows that increasing the dataset alone, is not sufficient to improve the testing accuracy of the model.

In model 5, the loss function is changed from mean_square_error to categorical_crossentropy. The mean squared error (MSE) or mean squared deviation of an estimator measures the average of the squares of the errors while categorical_crossentropy (CCE) is a softmax activation plus a Cross-Entropy loss. It sets up a classification problem between classes more than 2 for every class in C.

$$MSE = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

Where,yi to desired value and yi to the actual value obtained

$$CCE = \frac{1}{M}\sum_{p}^{M} -log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)$$

where, M: number of classes (Arson, Burglary, Shooting, etc), log: the natural log, s: is the CNN score for each positive class, 1/M: scaling factor to make the loss invariant and p: predicted probability observation o is of class c

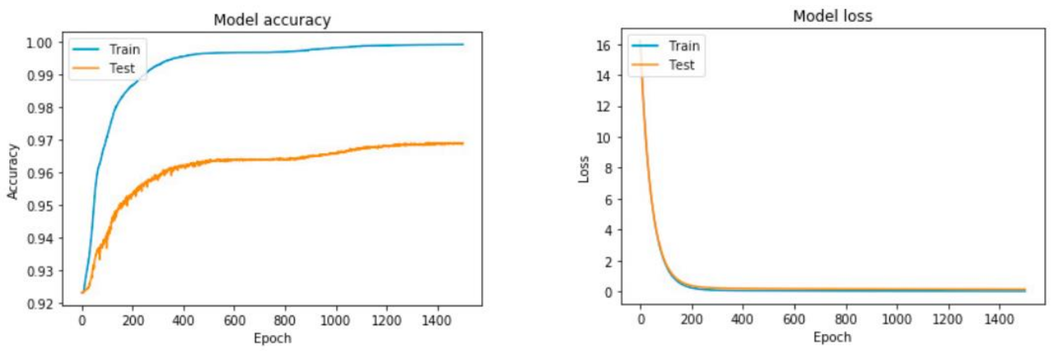


**Fig. 8. (a) Accuracy of Model 5; (b) Loss of Model 5**

Changing the error function has boosted the testing accuracy to a considerable amount and loss function is also converging faster as shown in Fig. 8. (a) and (b) resp.

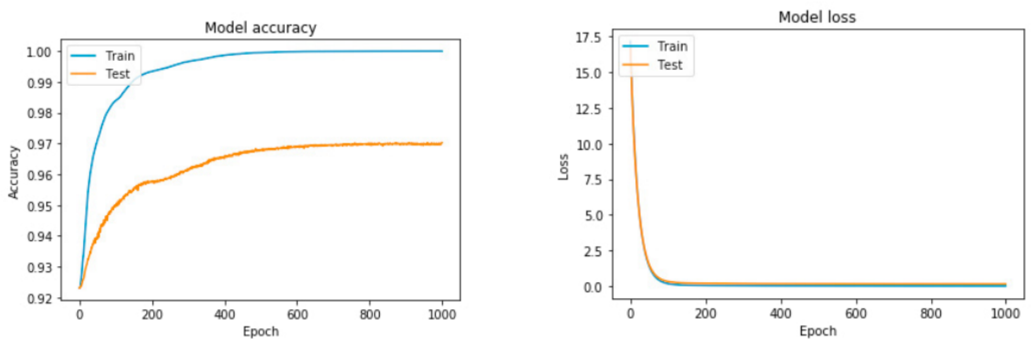


**Fig. 9. (a) Accuracy of Model 6; (b) Loss of Model 6**

Thus, all the changes integrated so far have a positive impact on the overall performance of the model.

Finally, the entire dataset is augmented by vertically flipping the dataset. Model 6 is trained on a dataset, double in size of the dataset used in model 5. This is done to optimise the validation accuracy of the model. Fig. 9. (a) shows that the training accuracy of the model increases logarithmically in accordance with the training curve. Fig. 9. (b) plots the loss function of the trained model.

**Table 1. Performance Comparison of all models**

| Models | train_loss | train_acc | val_loss | val_acc | Overfitting |
|---|---|---|---|---|---|
| Model 1 | 0.0652 | 0.9308 | 0.0142 | 0.9630 | Maximum |
| Model 2 | 0.1819 | 0.9033 | 0.0793 | 0.9896 | Considerable amount |
| Model 3 | 0.0062 | 1.0000 | 0.1333 | 0.8405 | Reduced |
| Model 4 | 0.0248 | 0.8458 | 0.0569 | 0.4850 | Reduced |
| Model 5 | 0.0148 | 0.9993 | 0.1341 | 0.9690 | Reduced |
| Model 6 | 0.0098 | 0.9999 | 0.1548 | 0.9723 | Least |

Model 6 has comparatively the best overall performance amongst all the six models that have been implemented during the course of this work as shown in Table 1. Table 1 present the results obtained by six models that have been explained so far. Furthermore, we have tested model 6 on the self-collected dataset to examine its application in realtime scenarios.

# Chapter 4: Conclusion

This work suggests an approach to spot variation from the norm in real-world CCTV recordings. The normal data alone may not be effective to distinguish abnormalities in these recordings. Therefore, to handle the complexity of these realistic anomalies, both normal and anomalous videos have been considered and hence, maximised the accuracy of the model. Furthermore, to prevent the efforts-requiring temporal annotations of abnormal sections in training recordings, a general model of anomaly detection has been learned utilizing two distinct neural networks with a poorly labelled dataset. A rarely processed large-scale anomaly dataset consisting of 12 real-world anomalies has been utilized for learning with the aim of validating the suggested approach. The experimental results obtained during the work conclude that our suggested anomaly detection approach performs significantly better than the previously used methods.

**Table 2. Details about the Optimized Model.**

| | Values |
|---|---|
| Categories Identified | Abuse, Burglar, Explosion, Shooting, Fighting, Shoplifting, Road Accidents, Arson, Robbery, Stealing, Assault, Vandalism, Normal |
| Chunk Size | 8 frames |
| Optimizer | Stochastic Gradient |
| Error Function | Categorical Cross-Entropy |
| Regularization | Regularizers.l2 (0.01) |
| Activation Functions | Relu, Sigmoid, Softmax |
| Augmentation | Horizontal Flip |

As specified in Table 2., this Threat Recognition Model classifies the anomalies into thirteen categories: Abuse, Burglar, Explosion, Shooting, Fighting, Shoplifting, Road Accidents, Arson, Robbery, Stealing, Assault, Vandalism, and Normal. The model concatenates 8 frames to form a chunk. The optimizer used in this model is Adam and the error function is categorical_crossentropy. The model uses three different types of activation function; Relu, Sigmoid, and Softmax. Moreover, to further increase the testing accuracy of the model the dataset has been doubled by flipping the videos horizontally. Hence, the overall accuracy of the model is 97.23% with reduced overfitting. Eventually, to implement this model in real-time, it is necessary to consider all the hardware constraints explained in this work. Hence, a proper implementation plan will reduce the computation power, optimise the use of resources and eventually reduce the overall cost of the system.

# REFRENCES

[1] J. Kooij, M. Liem, J. Krijnders, T. Andringa, and D. Gavrila. Multi-modal human aggression detection. Computer Vision and Image Understanding, 2016.

[2] S. Mohammadi, A. Perina, H. Kiani, and M. Vittorio. Angry crowds: Detecting violent events in videos. In ECCV, 2016.

[3] Convolutional Neural Network (CNN) in Keras, https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras329fbbadc5f5

[4] Recurrent Neural Networks (RNN) in Keras, https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-forsentiment-analysis-af410fd85b47

[5] W. Li, V. Mahadevan, and N. Vasconcelos. Anomaly detection and localization in crowded scenes. TPAMI, 2014.

[6] X. Cui, Q. Liu, M. Gao, and D. N. Metaxas. Abnormal detection using interaction energy potentials. In CVPR, 2011.

[7] T. Hospedales, S. Gong, and T. Xiang. A Markov clustering topic model for mining behaviour in the video. In ICCV, 2009.

[8] Y. Zhu, I. M. Nayak, and A. K. Roy-Chowdhury. Context-aware activity recognition and anomaly detection in video. In IEEE Journal of Selected Topics in Signal Processing, 2013.

[9] L. Kratz and K. Nishino. Anomaly detection in extremely crowded scenes using Spatio-temporal motion pattern models. In CVPR, 2009.

[10] How to Automate Surveillance Easily with Deep Learning, https://medium.com/nanonets/how-to-automate-surveillance-easily-with-deeplearning-4eb4fa0cd68d, 2018.

[11] C. Lu, J. Shi, and J. Jia. Abnormal event detection at 150 fps in Matlab. In ICCV, 2013.

[12] IR. Mehran, A. Oyama, and M. Shah. Abnormal crowd behaviour detection using the social force model. In CVPR, 2009.

[13] I. Saleemi, K. Shafique, and M. Shah. Probabilistic modelling of scene dynamics for applications in visual surveillance. TPAMI, 31(8):1472– 1485, 2009.

[14] B. Zhao, L. Fei-Fei, and E. P. Xing. Online detection of unusual events in videos via dynamic sparse coding. In CVPR, 2011.

[15] Unusual crowd activity dataset of the University of Minnesota. In http://mha.cs.umn.edu/movies/crowdactivity-all.avi.

[16] A. Adam, E. Rivlin, I. Shimshoni, and D. Reinitz. Robust real-time unusual event detection using multiple fixed-location monitors. TPAMI, 2008.

[17] Boss dataset, http://www.multitel.be/image/researchdevelopment/research-projects/boss.php.

[18] Waqas Sultani, Chen Chen, Mubarak Shah. Real-world anomaly detection in surveillance videos. In IEEE/CVF Conference, 2018.

[19] Data Augmentation, https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentationc26971dc8ced

[20] Transfer learning from pre-trained models, https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751, 2018.