Anomaly Detection and Classification from CCTV Camera Feed

Aakash	2021002
Lakshay Chauhan	2021060
Shubham Sharma	2021099

BTP report submitted in fulfillment of the requirements
for the Degree of B.Tech. in Computer Science & Engineering
on April 30, 2024.

BTP Track
Engineering

BTP Advisor
Dr. Sujay Deb

Indraprastha Institute of Information Technology
New Delhi

Student's Declaration

I hereby declare that the work presented in the report entitled **Anomaly Detection and Classification from CCTV Camera Feed** submitted by me for the fulfillment of the requirements for the degree of *Bachelor of Technology* in *Computer Science & Engineering* at Indraprastha Institute of Information Technology, Delhi, is an authentic record of my work carried out under guidance of **Dr. Sujay Deb**. Due acknowledgements have been given in the report to all material used. This work has not been submitted anywhere else for the reward of any other degree.

Aakash	**IIITD, April 30, 2024**
Lakshay Chauhan	**IIITD, April 30, 2024**
Shubham Sharma	**IIITD, April 30, 2024**

Certificate

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

Dr. Sujay Deb	**IIITD, April 30, 2024**

Abstract

The recent rise in crime and accident rates has highlighted the difficulty in reporting these events to authorities in a timely manner. Continuous human monitoring of surveillance cameras is impractical, leading to the necessity for an automated process. Our proposed solution leverages CCTV feeds to pinpoint the exact frame and segment of the recording where anomalies occur, aiding in the swift determination of whether these anomalies are out of the ordinary or suspicious. By integrating convolutional neural networks (CNN) and recurrent neural networks (RNN), our goal is to accurately predict the nature of these anomalies in video footage. Due to hardware constraints, we experimented with several models, including Optical Flow, SNN, CNN, and LSTM. We combined these networks with Optical Flow to develop a model that demands minimal computational power while still delivering sufficiently high accuracy.

Keywords: Anomaly detection, Anomaly classification, CCTV camera, Neural networks, LRCN, Optical Flow, SNN, LSTM

Acknowledgments

We would like to express our sincere gratitude to all those who contributed to our project. First and foremost, We extend our deepest appreciation to our Advisor **Dr. Sujay Deb**, for their invaluable guidance, encouragement, and unwavering support throughout the process. Their expertise and insightful feedback have been instrumental in shaping this work. We are immensely thankful to **Indraprastha Institute of Information Technology Delhi (IIIT-Delhi)** for providing access to essential resources, data, and facilities crucial for the successful execution of this project. Their cooperation and assistance significantly enriched the quality of this report. Lastly, we would like to acknowledge all individuals, institutions, and sources whose work and contributions have been referenced in this report. Their insights and research have been pivotal in shaping the context and understanding of this study. We are grateful to all those mentioned above for their invaluable contributions, without which this report would not have been possible.

Work Distribution

The collaborative nature of this report incorporates a collective effort from all members, where each individual contributed comprehensively across all topics. The distribution of work was such that all members engaged equally and extensively in each section. The seamless integration of diverse perspectives and skills from each contributor underscores the collaborative spirit of this report. The unified effort was aimed at ensuring a comprehensive exploration of every topic, leveraging the collective expertise and insights of all authors. This approach was instrumental in presenting a well-rounded perspective that encapsulates the collaboration of our team.

Contents

Chapter 1

Introduction

Surveillance cameras are increasingly being used in public places e.g. streets, intersections, banks, shopping malls, etc. to increase public safety [4]. With the increase in the frequency of crimes and accidents, it is highly important to detect such kinds of anomalies and report them to the respective authorities. Generally, anomalous events rarely occur as compared to normal activities [4, 6], hence it is very difficult for a human to constantly monitor the CCTV footage and detect such anomalies and report it to the respective authorities [6]. Also it is very challenging to again find the cause of these anomalies when again looking at the long saved footage from CCTV. Therefore, to alleviate the waste of labor and time, an intelligent anomaly detection system is needed.

1.1 Motivation and Research Problem

The motivation for this project came from fire alarms. There is a separate fire alarm sensor which senses the fire and sets the fire buzzer ON. In our institute also there are such kinds of fire sensors and CCTV cameras are also installed. There would be no need for a fire sensor if we can detect fire directly from CCTV feed, hence such kind of fire detection camera can be cost effective as there would be no need to separately install fire sensors. We can directly detect fire from the camera and set the fire buzzer ON through Raspberry Pi which is attached with the camera on which the main anomaly detection algorithm would be running. We can also save the part of the video in which there was an anomaly so that when reviewing the cause of the anomaly, we don't have to go through the entire video clip or we don't have to skip to the exact timestamp where the anomaly happens.

We extend this idea of fire detection to detect other kinds of anomalies like Road accidents, fighting, shooting, explosions etc. If such kinds of anomalies can be detected accurately then it solves the problem of constant human monitoring. The goal of a practical anomaly detection system is to timely signal an activity that deviates from normal patterns and identify the time window of the occurring anomaly and save that particular video in a specific folder of the classified anomaly which can later be reviewed [4].

1.2 Overview

Video classification is not as trivial as classifying its constituent image frames. Context needs to be captured from sequence of frames rather than just a single one [6]. We are using CNN and RNN simultaneously to effectively predict the anomalies in the CCTV footage.We are using CNN to extract spatial features at a given timestamp in the input sequence (video) and then an LSTM to identify temporal relations between frames. For example, in case of backflip, the CNN will detect a fall or standing for some frame, now this information is passed to RNN which predicts it to be a backflip.

Chapter 2

Dataset Used

The **UCF Crime dataset** is utilized for anomaly detection and comprises 1900 extensive, unedited surveillance videos from real-world scenarios. It features 13 types of realistic anomalies such as Abuse, Arrest, Arson, Assault, Road Accident, Burglary, Explosion, Fighting, Robbery, Shooting, Stealing, Shoplifting, and Vandalism, in addition to normal footage.

We are currently addressing the following categories of anomalies: Arson, Explosion, Fighting, Road Accidents, Theft, and Vandalism.

2.1 Reason for choosing this dataset

The decision to train our model on this particular dataset stems from the potential benefits of using actual CCTV footage. Given that such footage often has lower quality and is captured from specific angles and heights, training on this data can enhance the model's accuracy in detecting anomalies. Conversely, using a high-quality dataset from arbitrary cameras could result in reduced precision when applied to the typical footage from real-world CCTV cameras.

Chapter 3

Research Approach

3.1 Optical Flow

Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera. It is 2D vector field where each vector is a displacement vector showing the movement of points from first frame to second.

3.1.1 Why Optical Flow?

Incorporating optical flow in a CNNLSTM model can enhance prediction in various tasks, particularly those involving sequential data or video analysis.

1. Optical flow is less sensitive to changes in appearance such as variations in lighting conditions or object texture compared to raw pixel values. By focusing on motion information rather than static appearance, the model becomes more robust to such changes, leading to better generalization and performance on unseen data.

2. LSTM are capable of capturing long-term dependencies in sequential data. When combined with optical flow, which provides information about motion across frames, the model can effectively learn and exploit long-term temporal dependencies in video sequences, enabling more accurate predictions over extended periods.

3.1.2 How Optical flow?

1. We have utilized the Farneback method which is implemented in OpenCV to compute optical flow between consecutive frames of video data. This method estimates the motion vector of each pixel from one frame to the next, capturing the directional movement of objects in the scene. [8]

2. The CNN processes the optical flow features extracted from each frame, while the LSTM layer captures temporal dependencies across multiple frames, enabling the model to learn sequential patterns and dynamics within the video data.

3.2 Spiking Neural Network

Spiking Neural Networks (SNNs) are a type of artificial neural network inspired by the biological neurons in the brain. Unlike traditional artificial neural networks (ANNs) like Convolutional Neural Networks (CNNs), which operate on continuous-valued signals, SNNs communicate through discrete, asynchronous events called spikes. These spikes represent the firing of neurons in response to input stimuli, mimicking the behavior of neurons in the brain.

3.2.1 Why replace CNN with SNN?

- SNNs are inherently event-driven, meaning they only generate spikes when there is a change in input stimuli. This allows for highly efficient processing, as computations are only performed when necessary, leading to potentially lower power consumption and faster inference times compared to traditional neural networks.
- SNNs naturally incorporate temporal information into their computations through the timing of spikes. This makes them well-suited for tasks involving sequential or time-varying data, such as speech recognition, video analysis, and time series prediction.

3.3 How we incorporate these technologies?

We have devised five distinct architectures for the task of video anomaly detection. Some architectures solely utilize optical flow (OF), while others employ SNN, and a few combine both approaches. Additionally, we have created a baseline architecture to facilitate the comparison of their results and performance. The architectures are as follows:

1. **CNNLSTM**: This *baseline* architecture employs convolutional layers to capture the visual features of a video segment, followed by an LSTM layer to process the temporal information.
2. **OF-CNNLSTM**: This model preprocesses visual data by combining a motion matrix with visual features, then applies CNN layers and concludes with an LSTM layer.
3. **OF-CNN**: Similar to the OF-CNNLSTM, but it omits the final LSTM layer.
4. **OF-SNN**: This architecture replaces the CNN with an SNN for more computationally efficient inference compared to traditional ANNs.
5. **OF-SNNLSTM**: Building on the OF-SNN, this model adds an LSTM layer to handle sequential data.

Chapter 4

Model architectures

4.1 CNNLSTM

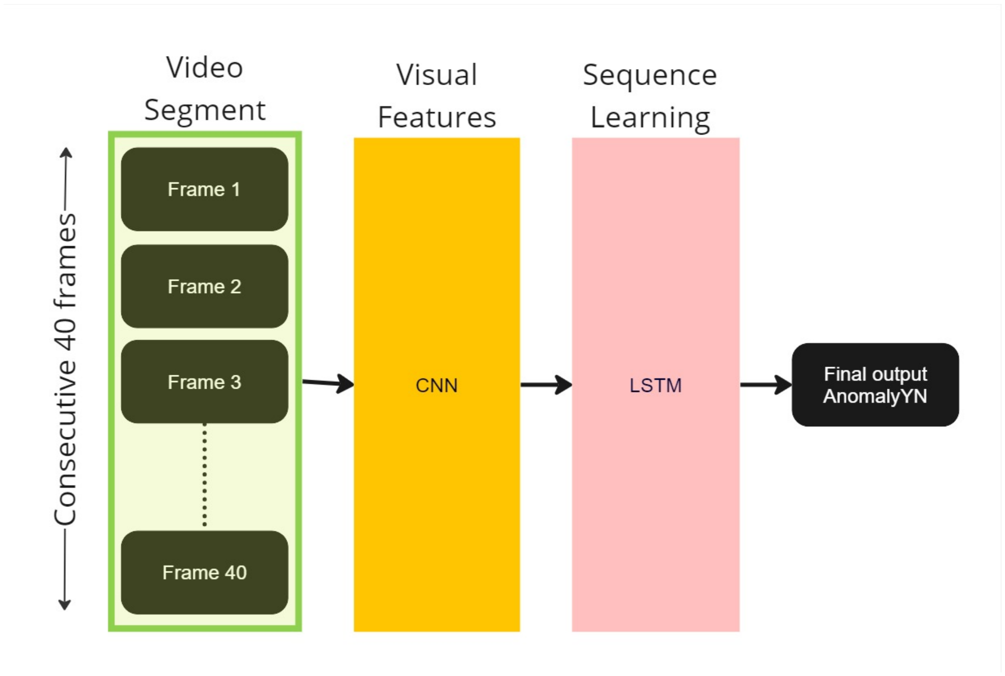


Figure 4.1: CNNLSTM model framework

In this architecture, we initially capture a video segment consisting of SEQUENCE_LENGTH consecutive frames from a video. This video segment is then preprocessed, including resizing the frames to fit a specific shape. This standardization allows for working with multiple videos of varying dimensions as they are all resized to a specific shape. Subsequently, the processed video segment is passed to the CNN model. The model first applies a convolutional layer to the segment, followed by max pooling to condense the segment and then a dropout layer to randomly deactivate some pixels in the segment frames. This step is crucial for mitigating model overfitting on the training data. By preventing the model from memorizing information specific to the training data, it minimizes the risk of poor performance on unseen data. These three layers are applied to the resulting segment three more times, each time with varying parameters. Finally, the segment is flattened before being passed to the LSTM layer. The LSTM layer examines the CNN output for each frame and provides a final assessment of whether the video segment is anomalous or not.

Layer (type)	Output Shape
Conv2D	(None, 92, 92, 16)
MaxPooling2D	(None, 23, 23, 16)
Dropout	(None, 23, 23, 16)
Conv2D	(None, 23, 23, 32)
MaxPooling2D	(None, 5, 5, 32)
Dropout	(None, 5, 5, 32)
Conv2D	(None, 5, 5, 64)
maxPooling2D	(None, 2, 2, 64)
Dropout	(None, 2, 2, 64)
Conv2D	(None, 2, 2, 64)
MaxPooling2D	(None, 1, 1, 64)
Flatten	(None, 64)
LSTM	(None, 2)
Dense	(None, 2)

Figure 4.2: CNNLSTM model architecture

Note: All layers, excluding the LSTM and the final Dense layer, have a Time Distributed wrapper around them, though this detail isn't depicted in the architecture for visual aesthetics. The Time Distributed wrapper is necessary due to the nature of video segment processing. Convolutional layers operate on images rather than videos, so wrapping them in the Time Distributed wrapper enables them to process each frame independently. The LSTM layer at the end aggregates these time-distributed outputs to produce a single score indicating the anomaly level of the video segment.

4.2 OF-CNNLSTM

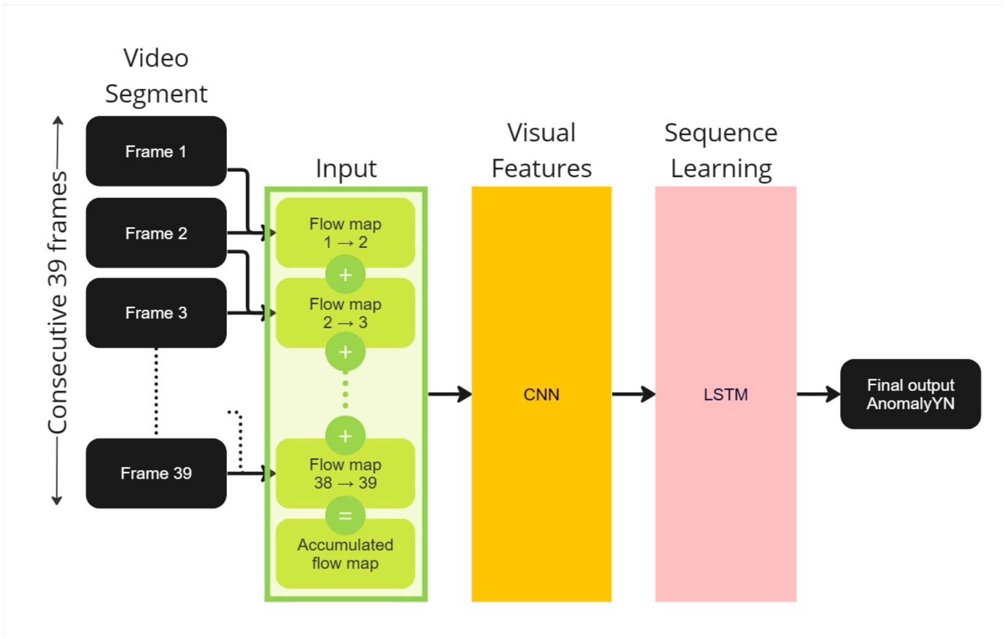


Figure 4.3: OF-CNNLSTM model framework

In this architecture, we initially utilize optical flow to capture motion maps between two consecutive frames. The resulting motion matrix is accumulated over time and appended to the resized frames, completing the input generation process. Subsequently, similar to the architecture depicted in 4.2, the input is passed through CNN layers. These layers capture visual information and progressively reduce the input's dimensionality, as demonstrated in architecture 4.4. Ultimately, the output is flattened at the penultimate layer and forwarded to an LSTM layer with 32 neurons. This LSTM layer outputs a probability indicating the likelihood of the segment being anomalous.

Layer (type)	**Output Shape**
Conv2D	(None, 128, 128, 16)
MaxPooling2D	(None, 32, 32, 16)
Dropout	(None, 32, 32, 16)
Conv2D	(None, 32, 32, 32)
MaxPooling2D	(None, 8, 8, 32)
Dropout	(None, 8, 8, 32)
Conv2D	(None, 8, 8, 64)
MaxPooling2D	(None, 4, 4, 64)
Dropout	(None, 4, 4, 64)
Conv2D	(None, 4, 4, 64)
MaxPooling2D	(None, 2, 2, 64)
Flatten	(40, None, 256)
LSTM	(None, 2)
Dense	(None, 2)

Figure 4.4: OF-CNNLSTM model architecture

Note: All layers, excluding the LSTM and the final Dense layer, have a Time Distributed wrapper around them, though this detail isn't depicted in the architecture for visual aesthetics. The Time Distributed wrapper is necessary due to the nature of video segment processing. Convolutional layers operate on images rather than videos, so wrapping them in the Time Distributed wrapper enables them to process each frame independently. The LSTM layer at the end aggregates these time-distributed outputs to produce a single score indicating the anomaly level of the video segment.

4.3 OF-CNN

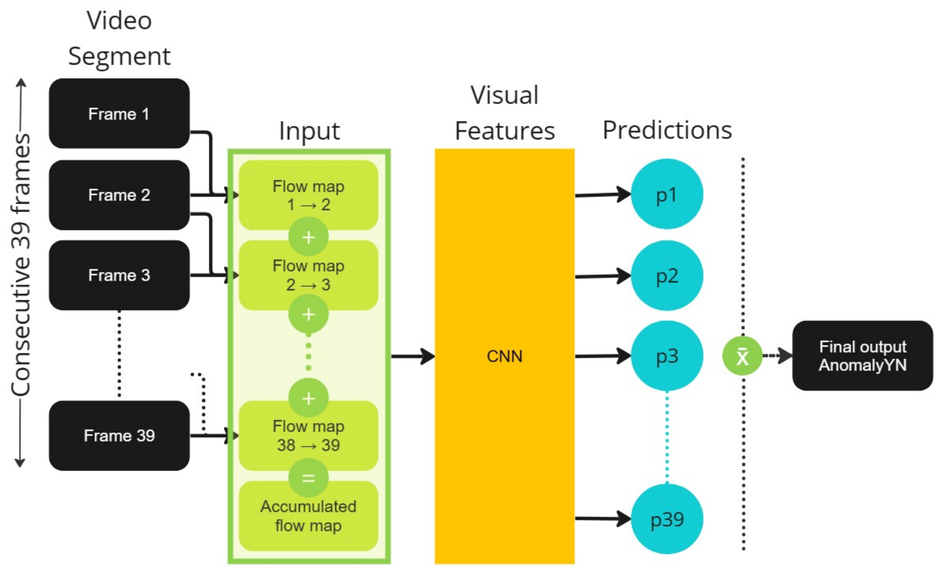


Figure 4.5: OF-CNN model framework

Similar to the architecture depicted in 4.2, optical flow is utilized to capture motion between two consecutive frames. The resulting motion matrix is accumulated over time and combined with the resized frames to complete the input generation process. Subsequently, this input is passed through CNN layers, which extract visual information and gradually reduce the input's dimensionality, as indicated in 4.2. The output of the CNN layers is flattened, and a probability is returned, indicating the likelihood of an anomaly occurring in the video segment. Since CNN operates on individual images rather than videos, each CNN output is independent of the others. Therefore, unlike the previous architecture, the outputs

Layer (type)	**Output Shape**
Conv2D	(None, 128, 128, 16)
Activation	(None, 128, 128, 16)
BatchNormalization	(None, 128, 128, 16)
AveragePooling2D	(None, 32, 32, 16)
Dropout	(None, 32, 32, 16)
Conv2D	(None, 32, 32, 32)
Activation	(None, 32, 32, 32)
BatchNormalization	(None, 32, 32, 32)
AveragePooling2D	(None, 8, 8, 32)
Dropout	(None, 8, 8, 32)
Conv2D	(None, 8, 8, 64)
Activation	(None, 8, 8, 64)
BatchNormalization	(None, 8, 8, 64)
AveragePooling2D	(None, 4, 4, 64)
Dropout	(None, 4, 4, 64)
Conv2D	(None, 4, 4, 64)
Activation	(None, 4, 4, 64)
BatchNormalization	(None, 4, 4, 64)
AveragePooling2D	(None, 2, 2, 64)
Dropout	(None, 2, 2, 64)
Flatten	(None, 256)
Dense	(None, 2)
Activation	(None, 2)

Figure 4.6: OF-CNN model architecture

Note: In contrast to previous architectures, this one separates the activation function from the Convolutional and Dense layers, positioning them immediately after the respective layer. This adjustment was essential for the SNN architectures 4.4, 4.5, as they build upon the CNN architecture. Additionally, the $MaxPooling2d$ operation used in previous architectures 4.1, 4.2 has been replaced with $AveragePooling2D$, which computes the average of the values inside the kernel rather than selecting the maximum.

are averaged to produce the final result.

4.4 OF-SNN

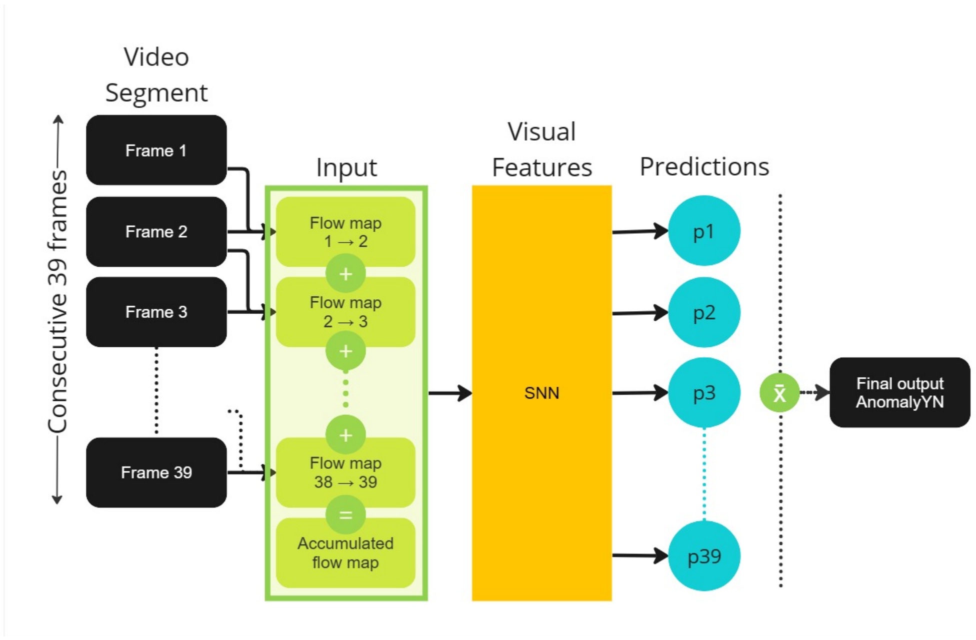


Figure 4.7: OF-SNN model framework

The core architecture of this model is the same as that shown in 4.3. The only difference is the utilization of SNN (spiking neural networks) instead of CNN. The philosophy behind this architecture is its suitability for operation on limited hardware machines. Hence, the use of SNNs, which have proven to be more efficient on neuromorphic hardware compared to CNNs.

4.5 OF-SNNLSTM

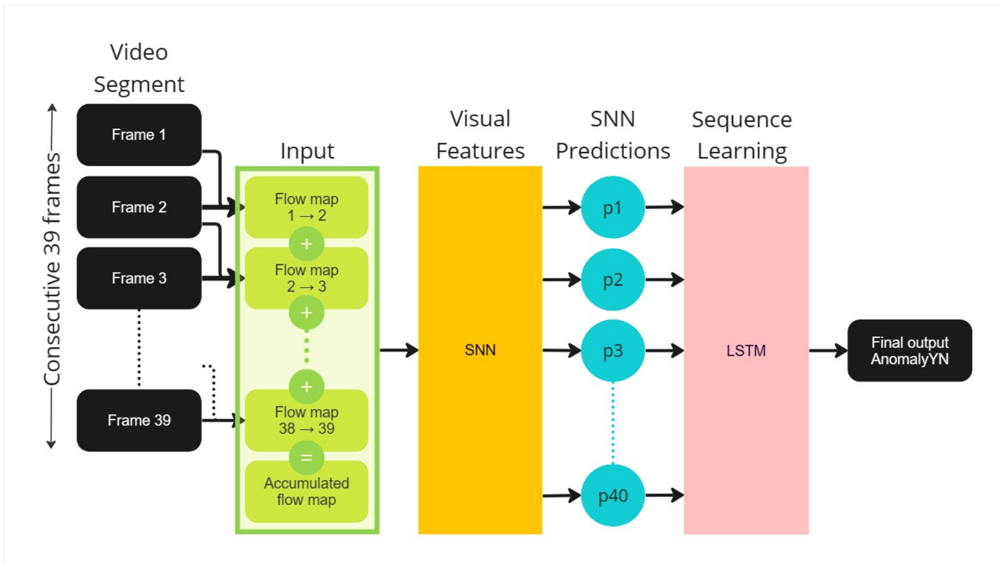


Figure 4.8: OF-SNNLSTM model framework

Layer (type)	Output Shape
LSTM	(None, 32)
Dense	(None, 2)

Table 4.1: LSTM architecture of the OF-SNNLSTM

The core architecture of this model remains the same as that depicted in 4.4. However, since the SNN serves as a replacement for the CNN model, it cannot be used in combination with LSTM. In contrast, architecture 4.2 employs CNN in conjunction with LSTM, enabling the backpropagation of losses from the LSTM layer to the convolutional layer for simultaneous fine-tuning. In this architecture, the use of LSTM alongside SNN is not feasible. We trained an SNN model with the same architecture as 4.4 separately from the structure outlined in 4.1. The output generated by the $OF - SNN$ model is then fed into the LSTM model for the final prediction.

Chapter 5

Results

Our model underwent a five-phase testing process to evaluate its accuracy and computational efficiency. Our goal was to achieve sufficiently high accuracy while maintaining low computational demands. For the purpose of creating an intelligent CCTV system, it's crucial that the model operates effectively on hardware with limited computational resources. We aimed for the anomaly detection and classification to take place at the camera level, ensuring that only videos identified as anomalous are transmitted to the server for storage.

5.1 Test on test dataset

Initially, we divided our dataset into training and testing sets with a ratio of 3 : 1. We trained and validated our models on the training dataset and assessed their performance on the testing dataset.

Model Architecture	Arson		Road Acc.		Explosion		Vandalism		Shooting	
	Loss	Acc.	Loss	Acc.	Loss	Acc.	Loss	Acc.	Loss	Acc.
CNNLSTM	0.236	0.898	0.502	0.743	0.298	0.897	0.532	0.796	0.603	0.714
OF-CNNLSTM	0.257	0.877	0.524	0.797	0.470	0.816	0.443	0.796	0.477	0.775
OF-CNN	0.054	0.978	0.038	0.973	0.055	0.978	0.058	0.976	0.040	0.984
OF-SNN	0.536	0.757	0.664	0.655	1.098	0.650	0.563	0.747	0.532	0.785
OF-SNNLSTM	0.637	0.673	0.548	0.756	0.633	0.673	0.632	0.673	0.630	0.673

Table 5.1: Loss and Accuracy of various model architectures

5.2 Test on youtube videos

Subsequently, testing was conducted on a laptop using random YouTube videos to assess the accuracy of our trained model. We randomly selected 12 videos each of arson and explosions from YouTube and submitted their video frames to our model for anomaly detection. Out of the 12 arson videos, our model successfully identified anomalies in 11 of them. A similar outcome was observed for the explosion videos. It's important to note that these selected videos were not part of the dataset used for training; only CCTV videos were considered for testing. The YouTube links provided in 5.1 belong exclusively to the arson and explosion classes.

Arson Videos	**Explosion Videos**
Arson_Video_1	Explosion_Video_1
Arson_Video_2	Explosion_Video_2
Arson_Video_3	Explosion_Video_3
Arson_Video_4	Explosion_Video_4
Arson_Video_5	Explosion_Video_5
Arson_Video_6	Explosion_Video_6
Arson_Video_7	Explosion_Video_7
Arson_Video_8	Explosion_Video_8
Arson_Video_9	**Explosion_Video_9**
Arson_Video_10	Explosion_Video_10
Arson_Video_11	Explosion_Video_11
	Explosion_Video_12

Note: The links in bold indicate instances where the model failed to detect the anomaly in the respective videos. Upon analyzing these videos, we concluded that the model's inability to detect anomalies may be attributed to sudden cuts within the videos. The videos in the dataset consist of continuous segments of CCTV footage without cuts, so the model may not be accustomed to detecting anomalies in videos with abrupt transitions.

Figure 5.1: Youtube test video links for Arson and Explosion

5.3 Inference speed analysis

We then replicated the same testing procedure on the Raspberry Pi to evaluate its video processing capabilities. Initially, the device required 8 minutes to process a 42-second video using the CNNLSTM model, which proved inefficient. To address this, we reduced the overlapping factor between two video segments by discarding a greater number of frames from the buffer after each iteration and capturing more frames in advance, thereby reducing the processing time. Additionally, we then optimized our model by quantization which is to convert the weights from floating-point to integer format while maintaining their scales. Consequently, with the enhanced CNNLSTM model, the Raspberry Pi successfully processed a 42-second video in approximately 60 to 65 seconds.

Architecture	**Time Elapsed**	**Predictions #**	**Prediction Ratio**
CNNLSTM	88.253 secs	214	0.1689
OF-CNNLSTM	223.512 secs	307	0.2423
OF-CNN	316.944 secs	307	0.2423
OF-SNN	404.571 secs	307	0.2423
OF-SNNLSTM	436.110 secs	307	0.2423

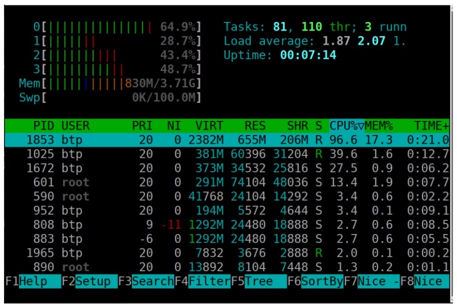
Table 5.2: Inference speed analysis on video of length 42.233 secs

Architecture	**Time Elapsed**	**Predictions #**	**Prediction Ratio**
CNNLSTM	133.861 secs	426	0.1755
OF-CNNLSTM	399.019 secs	598	0.2463
OF-CNN	588.782 secs	598	0.2463
OF-SNN	748.511 secs	598	0.2463
OF-SNNLSTM	802.453 secs	598	0.2463

Table 5.3: Inference speed analysis on video of length 80.933 secs

Architecture 4.1 utilizes $SEQUENCE\_LENGTH = 92$, whereas every other architecture employs $SEQUENCE\_LENGTH = 40$. Consequently, the number of predictions and prediction ratio remain consistent across all models except for $CNNLSTM$. With a smaller buffer window, the number of predictions would indeed increase.

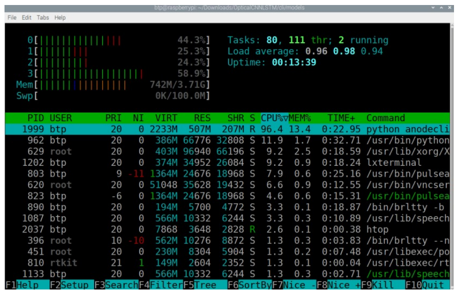
5.4 Resource consumption analysis



```
    0[||||||||||||||| 64.9%]   Tasks: 81, 110 thr; 3 runn
    1[||||||||        28.7%]   Load average: 1.87 2.07 1.
    2[|||||||||||     43.4%]   Uptime: 00:07:14
    3[||||||||||||    48.7%]
  Mem[|||||||||||830M/3.71G]
  Swp[            0K/100.0M]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+
 1853 btp        20   0 2382M  655M  206M R 96.6 17.3  0:21.0
 1025 btp        20   0  381M 60396 31204 R 39.6  1.6  0:12.7
 1672 btp        20   0  373M 34532 25816 S 27.5  0.9  0:06.2
  601 root       20   0  291M 74104 48036 S 13.4  1.9  0:07.7
  590 root       20   0 41768 24104 14292 S  3.4  0.6  0:02.2
  952 btp        20   0  194M  5572  4644 S  3.4  0.1  0:09.1
  808 btp         9 -11 1292M 24480 18888 S  2.7  0.6  0:08.5
  883 btp        -6   0 1292M 24480 18888 S  2.7  0.6  0:05.5
 1965 btp        20   0  7832  3676  2888 R  2.0  0.1  0:00.2
  890 root       20   0 13892  8104  7448 S  1.3  0.2  0:01.1
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice
```

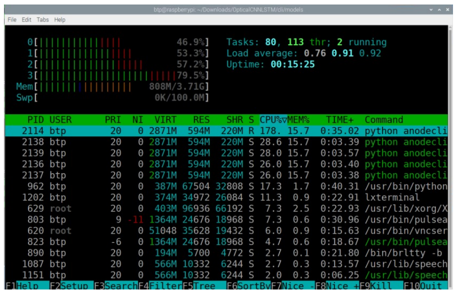
Figure 5.2: CNNLSTM consumption on RPI



```
    0[||||||||||||||||          44.3%]   Tasks: 80, 111 thr; 2 running
    1[||||||||||                25.3%]   Load average: 0.96 0.98 0.94
    2[|||||||||                 24.3%]   Uptime: 00:13:39
    3[||||||||||||||||||||||    58.9%]
  Mem[|||||||||||||||||     742M/3.71G]
  Swp[                       0K/100.0M]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+  Command
 1999 btp        20   0 2233M  507M  207M R 96.4 13.4  0:22.95 python anodecli
  962 btp        20   0  386M 66776 32808 S 11.9  1.7  0:32.71 /usr/bin/python
  629 root       20   0  403M 96940 66196 S  9.2  2.5  0:18.59 /usr/lib/xorg/X
 1202 btp        20   0  374M 34952 26084 S  9.2  0.9  0:18.24 lxterminal
  803 btp         9 -11 1364M 24676 18968 S  7.9  0.6  0:25.16 /usr/bin/pulsea
  620 root       20   0 51048 35628 19432 S  6.6  0.9  0:12.55 /usr/bin/vncser
  823 btp        -6   0 1364M 24676 18968 S  4.6  0.6  0:15.31 /usr/bin/pulsea
  890 btp        20   0  194M  5700  4772 S  3.3  0.1  0:18.87 /bin/brltty -b
 1087 btp        20   0  566M 10332  6244 S  3.3  0.3  0:10.89 /usr/lib/speech
 2037 btp        20   0  7868  3648  2828 R  2.6  0.1  0:00.38 htop
  396 root       10 -10  562M 10276  8872 S  1.3  0.3  0:03.83 /bin/brltty --n
  451 root       20   0  230M  8304  5904 S  1.3  0.2  0:07.48 /usr/libexec/po
  810 rtkit      21   1  149M  2604  2352 S  1.3  0.1  0:00.04 /usr/libexec/rt
 1133 btp        20   0  566M 10332  6244 S  1.3  0.3  0:02.71 /usr/lib/speech
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

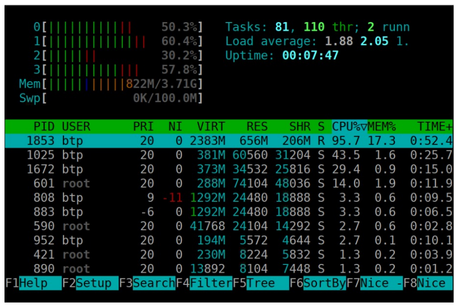
Figure 5.3: OF-CNNLSTM consumption on RPI



```
    0[||||||||||||||||          46.9%]   Tasks: 80, 113 thr; 2 running
    1[||||||||||||||||||        53.3%]   Load average: 0.76 0.91 0.92
    2[||||||||||||||||||||      57.2%]   Uptime: 00:15:25
    3[||||||||||||||||||||||||||79.5%]
  Mem[||||||||||||||||||    808M/3.71G]
  Swp[                       0K/100.0M]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+  Command
 2114 btp        20   0 2871M  594M  220M R 178. 15.7  0:35.02 python anodecli
 2138 btp        20   0 2871M  594M  220M S 28.6 15.7  0:03.39 python anodecli
 2139 btp        20   0 2871M  594M  220M S 28.0 15.7  0:03.57 python anodecli
 2136 btp        20   0 2871M  594M  220M S 26.0 15.7  0:03.40 python anodecli
 2137 btp        20   0 2871M  594M  220M S 26.0 15.7  0:03.38 python anodecli
  962 btp        20   0  387M 67504 32808 S 17.3  1.7  0:40.31 /usr/bin/python
 1202 btp        20   0  374M 34972 26084 S 11.3  0.9  0:22.91 lxterminal
  629 root       20   0  403M 96936 66192 S  7.3  2.5  0:22.93 /usr/lib/xorg/X
  803 btp         9 -11 1364M 24676 18968 S  7.3  0.6  0:30.96 /usr/bin/pulsea
  620 root       20   0 51048 35628 19432 S  6.0  0.9  0:15.63 /usr/bin/vncser
  823 btp        -6   0 1364M 24676 18968 S  4.7  0.6  0:18.67 /usr/bin/pulsea
  890 btp        20   0  194M  5700  4772 S  2.7  0.1  0:21.80 /bin/brltty -b
 1087 btp        20   0  566M 10332  6244 S  2.7  0.3  0:13.57 /usr/lib/speech
 1151 btp        20   0  566M 10332  6244 S  2.0  0.3  0:06.25 /usr/lib/speech
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

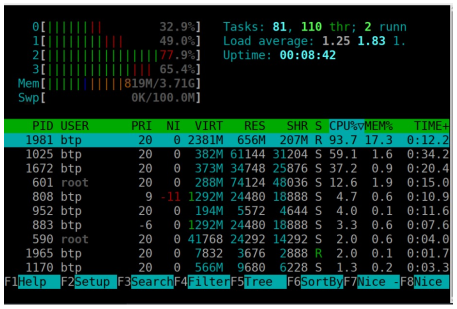
Figure 5.4: OF-CNN consumption on RPI



```
    0[||||||||||||||  50.3%]   Tasks: 81, 110 thr; 2 runn
    1[||||||||||||||| 60.4%]   Load average: 1.88 2.05 1.
    2[||||||||        30.2%]   Uptime: 00:07:47
    3[||||||||||||||  57.8%]
  Mem[|||||||||||822M/3.71G]
  Swp[            0K/100.0M]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+
 1853 btp        20   0 2383M  656M  206M R 95.7 17.3  0:52.4
 1025 btp        20   0  381M 60560 31204 S 43.5  1.6  0:25.7
 1672 btp        20   0  373M 34532 25816 S 29.4  0.9  0:15.0
  601 root       20   0  288M 74104 48036 S 14.0  1.9  0:11.9
  808 btp         9 -11 1292M 24480 18888 S  3.3  0.6  0:09.5
  883 btp        -6   0 1292M 24480 18888 S  3.3  0.6  0:06.5
  590 root       20   0 41768 24104 14292 S  2.7  0.6  0:02.8
  952 btp        20   0  194M  5572  4644 S  2.7  0.1  0:10.1
  421 root       20   0  230M  8224  5832 S  1.3  0.2  0:03.9
  890 root       20   0 13892  8104  7448 S  1.3  0.2  0:01.2
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice
```

Figure 5.5: OF-SNN consumption on RPI



```
    0[|||||||||       32.9%]   Tasks: 81, 110 thr; 2 runn
    1[|||||||||||     49.0%]   Load average: 1.25 1.83 1.
    2[||||||||||||||||77.9%]   Uptime: 00:08:42
    3[||||||||||||||| 65.4%]
  Mem[|||||||||||819M/3.71G]
  Swp[            0K/100.0M]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+
 1981 btp        20   0 2381M  656M  207M R 93.7 17.3  0:12.2
 1025 btp        20   0  382M 61144 31204 S 59.1  1.6  0:34.2
 1672 btp        20   0  373M 34748 25876 S 37.2  0.9  0:20.4
  601 root       20   0  288M 74124 48036 S 12.6  1.9  0:15.0
  808 btp         9 -11 1292M 24480 18888 S  4.7  0.6  0:10.9
  952 btp        20   0  194M  5572  4644 S  4.0  0.1  0:11.6
  883 btp        -6   0 1292M 24480 18888 S  3.3  0.6  0:07.6
  590 root       20   0 41768 24292 14292 S  2.0  0.6  0:04.0
 1965 btp        20   0  7832  3676  2888 R  2.0  0.1  0:01.7
 1170 btp        20   0  566M  9680  6228 S  1.3  0.2  0:03.3
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice
```

Figure 5.6: OF-SNNLSTM consumption on RPI

5.5 Final Testing

The final phase of testing involves using a real camera and a buzzer. Previously, video frames were sourced from external videos or YouTube; however, frames are now being captured directly from a USB video camera. The camera is utilized to record anomalous events or to capture footage of a laptop screen playing an anomalous video. The model then predicts anomalies in real-time. If an anomaly is detected, the buzzer is activated; if not, it remains off.

5.5.1 Observations

1. The model demonstrates optimal performance when viewed from specific angles, particularly when observed from a top-to-bottom perspective. We suspect that this phenomenon arises due to the limited variety of data in the dataset. A majority of the videos contained in the dataset are captured from a corner-top-left perspective, resembling a generic camera viewpoint.

2. The model architecture described in 4.1 exhibits poor performance under conditions of limited lighting. However, models incorporating optical flow do not encounter this limitation.

3. Actions such as placing a hand in front of the camera, camera movement or shaking, and changes in lighting trigger anomalies. This observation is promising, and we suspect it arises from the absence of such scenarios in the training dataset, rendering the model susceptible to them.

Chapter 6

Final Working

1. Anomalies are being detected by taking video frames from the USB camera attached to raspberry PI. The current raspberry Pi that we are using is raspberry Pi 4 Model B.

2. To consider an event as anomaly two thresholds have been taken. One on the basis of the consecutive number of video frames and other is on the basis of anomaly score. If we get anomaly score above 0.65 for 5 consecutive frames then we consider that 5 frames as anomaly. These threshold can be changed as per the need.

3. The model fills the frame buffer, which has a capacity of $SEQUENCE\_LENGTH$, with consecutive frames from the video stream. If the buffer reaches its capacity, it generates the optical flow map of the buffer and appends it to the buffer. Subsequently, the buffer is resized and sent to the model for inference. After inference, it deletes some leading frames determined by a $SHIFTING\_RATIO$ parameter. This process is then repeated.

4. If anomaly is predicted for a particular segment, the buzzer is set to on and a message is sent to the respective authorities about the anomaly.

5. The predicted anomalous video is sent to the server and stored under that anomaly directory. These video can later be watch by the authorites. Using that stored video we can again train our model accordingly to increase the accuracy furthermore. In case of false alarm the video then can be used to fine tune the model.

Chapter 7

Conclusion

Utilizing optical flow in training leads to improved accuracy for certain classes, although it also result in reduced accuracy for others. The variations in accuracy are relatively minor, but there is a notable rise in computational demand during real-time anomaly prediction. This surge in computation stems from the necessity to transform each frame into flow maps before they are processed for prediction.

To address the increase in computational demand, spiking neural networks (SNNs) can be employed. SNNs demand less computational power compared to traditional convolutional neural networks (CNNs), making them particularly beneficial in scenarios where hardware capabilities are limited, as in our situation. Integerating SNN with optical is not easy as of now, more research work is still need to be done on SNN and there is less resource available to understand how to implement SNN effectively with optical flow.

Bibliography

[1] Rohith, Muttineni Sai. 2022. “Keras EarlyStopping Callback to Train the Neural Networks Perfectly.” Medium. September 10, 2022. https://pub.towardsai.net/keras-earlystopping-callback-to-train-the-neural-networks-perfectly-2a3f865148f7.

[2] Data Science Stack Exchange. “Early Stopping on Validation Loss or on Accuracy?”, n.d. https://datascience.stackexchange.com/questions/37186/early-stopping-on-validation-loss-or-on-accuracy/49594#49594.

[3] Brownlee, Jason. 2019. “How to Configure the Learning Rate Hyperparameter When Training Deep Learning Neural Networks”. Machine Learning Mastery. June 4, 2019. https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/.

[4] Sultani, Waqas, Chen Chen, and Mubarak Shah. “Real-World Anomaly Detection in Surveillance Videos.” arXiv (Cornell University), January 12, 2018. https://arxiv.org/pdf/1801.04264.pdf

[5] Brownlee, Jason. 2019. “How to Configure the Learning Rate Hyperparameter When Training Deep Learning Neural Networks.” Machine Learning Mastery. June 4, 2019. https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

[6] “Anomaly Detection in Surveillance Videos — IEEE Conference Publication — IEEE Xplore.” n.d. Ieeexplore.ieee.org. Accessed November 29, 2023. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9001731

[7] “Dr. Chen Chen.” 2018. Ucf.edu. 2018. https://www.crcv.ucf.edu/chenchen/dataset.html

[8] ”Fast and accurate motion estimation using orientation tensors and parametric motion models,” Proceedings 15th International Conference on Pattern Recognition. ICPR-2000, Barcelona, Spain, 2000, pp. 135-139 vol.1, doi: 10.1109/ICPR.2000.905291. https://ieeexplore.ieee.org/abstract/document/905291