Master of Science in Software Engineering
May 2023

Anomaly detection in surveillance camera data

Viktoriia Semerenska

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

Contact Information:
Author(s):
Viktoria Semerenska
E-mail: vise22@student.bth.se

University advisor:
Dr. Oleksandr Adamov
Department of Software Engineering

Co advisor:
Axel Keskikangas
Company/HEI: Axis Communications AB

Faculty of Computing	Internet	: www.bth.se
Blekinge Institute of Technology	Phone	: +46 455 38 50 00
SE-371 79 Karlskrona, Sweden	Fax	: +46 455 38 50 57

Abstract

Background. The importance of detecting anomalies in surveillance camera data cannot be overemphasized. With the increasing availability of surveillance cameras in public and private locations, the need for reliable and effective methods to detect anomalous behavior has become critical to public safety. Anomaly detection algorithms can help identify potential threats in real time, allowing for rapid intervention and prevention of criminal activity. The examples of anomalies that can be detected by analyzing surveillance camera data include suspicious loitering or lingering, unattended bags or packages, crowd gatherings or dispersals, trespassing or unauthorized access, vandalism or property damage, violence or aggressive behavior, abnormal traffic patterns, missing or abducted persons, unusual pedestrian behavior, environmental anomalies. Detecting these anomalies in surveillance camera data can enable law enforcement, security personnel, and other relevant authorities to respond quickly and effectively to potential threats, ultimately contributing to a safer environment for all.

Surveillance camera data contains a large amount of information that is difficult for humans to analyze in real time. In addition, the sheer volume of data generated by surveillance cameras makes manual analysis impractical. Therefore, the development of automated anomaly detection algorithms is crucial for effective and efficient surveillance.

Objectives. The goal of this master's thesis is to detect anomalies using video cameras with an embedded machine learning processor and video analytics, such as human behavior. For this purpose, the most appropriate machine learning techniques will be selected and after comparing the results of these techniques, the best anomaly detection technique for the given circumstances will be identified.

Methods. To gather the evidence needed to answer the research questions, I will use a combination of methods appropriate to the study design. The study will follow a mixed-methods approach, combining a systematic literature review (SLR) and a formal experiment.

Results. In this study, we investigated the effectiveness of various machine learning algorithms in detecting anomalous human behavior in video surveillance data. Our results, obtained from a systematic literature review and a formal experiment, reveal that Convolutional Neural Networks (CNN) outperform other algorithms in detecting behavioral anomalies, with a ROC-AUC of 0.8346 and an EER of 0.2439. Long Short-Term Memory (LSTM) networks, Recurrent Neural Networks (RNN), and Support Vector Machines (SVM) followed in effectiveness, with LSTM achieving a ROC-AUC of 0.7797 and an EER of 0.3039, RNN achieving a ROC-AUC of 0.7602 and an EER of 0.3292, and SVM achieving a ROC-AUC of 0.7281 and an EER of 0.3780.

Conclusion. In conclusion, this study demonstrates that specific machine learning algorithms are more effective at detecting certain types of anomalous human behavior in video surveillance data. Our findings emphasize the need for selecting the most appropriate algorithm based on the desired application and type of anomaly being detected. Moreover, the development of collaborative systems that combine the strengths of multiple algorithms offers a promising direction for future research, enabling enhanced accuracy and reliability in anomaly detection. This work provides valuable insights for both researchers and practitioners in the field of video surveillance, aiding in the selection and implementation of machine learning algorithms for detecting anomalous human behavior.

Keywords: Anomaly detection, Machine Learning, Unsupervised learning, Video Analytics, Artificial Intelligence

Acknowledgments

We would like to express gratitude to Oleksander Adamov for his supervision, guidance, and support throughout the course of this thesis. His extensive knowledge and insights have been crucial to the success of our research.

We are also grateful to Axis Communications AB for providing the necessary resources and equipment to conduct our experiments. Special thanks go to our company supervisor, Axel Keskikangas , and Victoria Vucic, whose motivation, expertise, and encouragement have been instrumental in the completion of this project.

CONTENTS

1 INTRODUCTION

The increasing prevalence of surveillance cameras in public and private spaces has led to a significant growth in the volume of visual data generated daily. This data holds immense potential for enhancing security and public safety by providing valuable insights into human behavior patterns. However, the sheer volume of data produced poses a challenge to manual processing and analysis, necessitating the development of automated methods for detecting anomalous behaviors that may signify potential threats or criminal activities.

Anomaly detection, in the context of surveillance camera data, refers to the identification of unusual or unexpected behaviors that deviate from the norm. These anomalies can include incidents such as theft, violence, traffic accidents, or other forms of suspicious activities. The ability to detect and respond to such events in real-time is crucial for ensuring the effectiveness of security measures and maintaining public safety [1].

The choice of theme for this thesis is motivated by the growing importance of leveraging advanced machine learning algorithms to address the challenges associated with the analysis of vast amounts of surveillance camera data. In recent years, various ML techniques have been proposed for anomaly detection, demonstrating promising results in a wide range of applications, from fraud detection to fault diagnosis in industrial systems. However, the application of these methods to surveillance camera data presents unique challenges, such as varying environmental conditions, occlusions, and diverse human behaviors, which require further investigation.

This research aims to explore the potential of different ML algorithms for anomaly detection in surveillance camera data, providing valuable insights into their effectiveness, strengths, and limitations. By doing so, this thesis will contribute to the development of more efficient and accurate systems for real-time anomaly detection in surveillance applications, ultimately enhancing public safety and security. Additionally, this research will also facilitate further advancements in the field of intelligent surveillance systems by offering a comprehensive understanding of the state-of-the-art techniques and their potential for addressing the challenges associated with the analysis of surveillance camera data.

1.1. Problem statement

One of the most significant challenges in processing and interpreting surveillance data is the detection of anomalous behaviors that may signify potential threats or criminal activities.

Consequently, there is a growing need for effective and efficient methods to automatically identify and flag these anomalies in real-time.

The specific problem this thesis will address is how to select and apply the most optimal anomaly detection algorithm that is robust to environmental factors and minimizes the number of false detections (positive and negative) in a particular scene. The relevance of this problem is that many environmental factors can affect the video image, leading to false positives or false negatives. Defining a robust anomaly detection algorithm that can minimize the limitations of certain scenes and consider the specifics of a particular application will improve public safety, which in the long run can minimize enterprise security risks and prevent criminal activity such as information theft, tail gating, and shoulder surfing [7].

In summary, the purpose of this research is to select and apply the best approach to anomaly detection in surveillance camera data for a particular scene. This research will move from the general area of computer vision and machine learning to the specific area of anomaly detection.[7]

1.2. Research questions

RQ1 What ML algorithms can be used to detect anomalous behavior of entities using cameras with intelligent real-time analytics?

RQ1.1 What are the characteristics of the datasets used in research on anomaly detection in surveillance camera data?

RQ1.2 What are the evaluation metrics of anomaly detection algorithms in surveillance camera data?

RQ1.3 What are the strengths and limitations of machine learning algorithms for detecting anomalous behavior in surveillance camera data?

RQ2 How do different machine learning algorithms from RQ1 compare in their effectiveness in detecting anomalous human behavior?

1.3 Outline

The document is organized as follows. The Introduction sets the context for the study. The Related Works chapter reviews existing literature on anomaly detection in surveillance camera data. The Method chapter outlines the systematic literature review and experimental procedures. The Results and Analysis chapter presents the findings and discusses the performance of the implemented algorithms.

2 RELATED WORKS

Detecting anomalies in surveillance camera data is a complex problem that has attracted considerable research attention.

This section will review current research in the field of anomaly detection in surveillance camera data, addressing problems related to the topic of the master's thesis. In particular, studies that have proposed algorithms for anomaly detection, presented a comparison of the performance of existing models, and presented their own datasets for machine learning training will be reviewed.

The study Deep Learning-based Methods for Anomaly Detection in Video Surveillance: An Overview [2] provides an overview of existing anomaly detection methods, highlighting their advantages and disadvantages. This paper also reveals metrics for evaluating the performance of the methods under consideration and characteristics of the datasets used, which is particularly relevant for the research questions of the master's thesis.

Some of the papers provide their own datasets with video footage that can be used to train machine learning.

Waqas Sultani, et. al in their paper [3] have considered the limitations of existing datasets used for machine learning training and presented their own solution to the problem. The authors created the most extensive dataset, consisting of 128 hours of real video footage, which includes 1900 videos, 13 types of anomalies and a set of normal videos. Using this dataset significantly improves the effectiveness of anomaly detection compared to other machine learning training approaches.

In paper [4] Amnah Aldayri and Waleed Albattah looked at the complexities of automating a video-analytics system to detect anomalies in busy scenes and crowded areas and presented a taxonomy for detecting anomalies in crowd scenes

Redhwan Al-amri et. al review in their study [5] machine learning and deep learning techniques for anomaly detection and evaluate their effectiveness using one of the most common datasets as an example. Extensive tables comparing the considered methods and techniques are presented as results of the work. In addition, the review can provide a basis for the comparative study of different machine learning algorithms in the detection of anomalous human behavior in a parking lot

Kallepalli Rohit Kumar et. al [6] provide comprehensive review of the existing literature on anomaly detection in surveillance systems and summarize various machine learning techniques used for this purpose. The article also highlights the challenges and limitations of using machine learning techniques for anomaly detection in surveillance systems.

Karan Thakkar et. al [7] propose an approach for detecting anomalous behavior in surveillance videos using background subtraction and object tracking techniques. The proposed approach involves modeling the background of the scene and detecting foreground objects, which are then tracked over time to detect anomalous behavior.

Maria Andersson et. al [8] propose a system for detecting anomalous behavior in urban environments and traffic scenarios using camera networks and machine learning algorithms. The proposed system involves processing camera data from multiple viewpoints and using machine learning algorithms.

The literature lacks a comprehensive comparison of various machine learning algorithms in terms of their effectiveness in detecting anomalous behavior in surveillance camera data. This research can fill this gap by systematically comparing and evaluating different algorithms, providing therefore insights into their strengths and limitations for this specific application.[11]

Furthermore, anomaly detection algorithms need to be robust to different surveillance scenarios, camera angles, lighting conditions, and object appearances. This research will investigate how different algorithms perform under varying conditions.

By addressing these gaps, this research will contribute to the development of more effective, efficient, and trustworthy anomaly detection systems for surveillance camera data, enhancing their practical applicability and utility in real-world settings.

The reviewed works are the basis for the research of the master's work, as they represent an extensive statistical material and source data, which will be used in the thesis.

3 METHOD

This chapter will describe and motivate the methods chosen for the study, such as: a systematic literature review, supervised and unsupervised machine learning.

3.1 Systematic Literature Review

The purpose of this systematic literature review is to search, evaluate, synthesize data and analyze it amongst existing studies related to the topic of the master's thesis in order to evaluate and summarize the data within the research questions.

This review will survey the literature in the field of anomaly detection using machine learning, and in particular, works that look at, evaluate and compare different machine learning methods and techniques. It is the analysis of such works that can answer the research question RQ1 and its sub-questions, providing the basis for the research question RQ2.

3.1.1 Database Selection

The following databases were used for the literature search:

- IEEE Xplore;
- Semantic Scholar;
- Scopus;
- arXiv;
- Google Scholar.

The databases considered were chosen because of the extensive archive of scientific papers in the public domain, as well as the ability to search by keywords and search using artificial intelligence, which made it possible to narrow down the search results to papers that were as relevant as possible to the subject matter.

3.1.2 Inclusion and Exclusion Criteria

This section presents the criteria that have been decided upon for filtering the literature and selecting the most related articles.

Inclusion criteria:

- article is published in peer-reviewed journals or conference proceedings;
- study is relevant to the research questions and the topic of the review;
- study has reported quantitative or qualitative data relevant to the review topic;
- paper is written in english;
- full text of paper is available;
- study published no earlier than 2015

Exclusion criteria:

- article is published in non-peer-reviewed sources;
- study is not relevant to the research questions or the topic of the review;
- study does not report any quantitative or qualitative data relevant to the review topic;
- paper is not written in English.
- studies that are not accessible in full text;
- studies that were published before 2015.

3.1.3 Search strategy

To formulate a successful search strategy, it is necessary to define basic concepts relevant to the research topic. The following have been identified:

- anomaly detection;
- surveillance cameras;
- machine learning methods;
- datasets.

Next, a list of keywords for each concept was developed, which can be combined using Boolean operators (AND, OR, NOT) to form search strings.

The retrieved articles were in turn filtered according to the inclusion and exclusion criteria after briefly reviewing the article title, description and content based on the inclusion and exclusion criteria.

The data extracted from the selected articles after synthesis and analysis are placed in the respective result tables and described in the results chapter.

3.1.4 Quality assessment

A table with eligibility criteria was compiled to assess the quality of the literature. This table was based on a proposal by Jianfeng Wen et al. [10]. The table is a list of questions in relation to each article to assess its quality. The answer to each question is scored and then the scores for each article are summed up and it becomes possible to rank the papers according to their quality for inclusion in the study. The table 3.1 with quality questions is presented below.

Table 3.1 – Quality questions

Q#	Quality questions	Relevant (1 pt.)	Relevant partly(0,5 pt.)	Not relevant (0 pt.)
1	Are the aims of the article clearly defined?			
2	Does the article contain an overview of machine learning methods?			
3	Does the article provide a comparative analysis of machine learning methods?			
4	Does the article provide a comparative analysis of datasets?			
5	Were the datasets used in the studies representative of real-world surveillance scenarios?			
6	Were the studies conducted using publicly available or proprietary datasets?			
7	Does the article use evaluation metrics for machine learning methods?			
8	Do the methods discussed in the article apply to anomalies in human behavior?			
9	Does the article identify the strengths and limitations of machine learning algorithms?			
10	Are the results and conclusions clearly stated?			

3.1.5 Data extraction

This section deals with data extraction, an important component of a systematic literature review. It involves the collection and systematization of relevant data from primary sources. In this SLR we aim to answer the research questions related to the use of machine learning algorithms to detect anomalies in surveillance camera data. For this purpose, academic databases were searched, resulting in the selection of 20 primary sources for data extraction. These sources include research articles, newsletters and reviews that discuss various aspects of anomaly detection in surveillance camera data. In this section, we present the extracted data in a table 3.2, categorizing them based on key criteria such as research methodology, dataset characteristics and performance of anomaly detection algorithms.

Table 3.2 – Data extraction

Study ID	Title	ML Techniques	Datasets	Evaluation Metrics
1	2	3	4	5
2	Deep learning-based methods for anomaly detection in video surveillance: a review	CNN, RNN, LSTM, GAN, Autoencoder, 3D ConvNet MIL, SVM, DNN	UCSD, UMN, Subway, CUHK, UCF-crime, Avenue, ShanghaiTech, Ped1, Ped2	ROC-AUC, F1-score, Precision, Recall, Frame-Level Accuracy, RMSE
3	Real-world Anomaly Detection in Surveillance Videos	MIL	UCF-crime, UMN, UCSD Ped 1, Ped 2, CUHK avenue, Subway, BOSS, UCF Crowd	ROC-AUC, EER
4	Taxonomy of Anomaly Detection Techniques in Crowd Scenes	CNN, RNN, LSTM	UCF-crime, PASCAL, VOC, UCSD, UMN, UCD, LV	EER, DR
5	A Review of Machine Learning and Deep Learning Techniques for Anomaly Detection in IoT Data	C_LOF, AutoCloud, TEDA Clustering, KPI-TSAD, HTM, SVM	UCSD, Ped 1, Ped 2, and CUHK Avenue	ACU, EER, TP, TN, FP, FN, AUC

Table 3.2 – Continued from previous page

1	2	3	4	5
6	Anomaly Detection In Surveillance System Using Machine Learning Techniques- A Review	BMS, HTM, LSAD, LSTM, DNN, CNN, CAD, CUSUM, FCN, RNN, SVM	N/A	N/A
7	ANOMALY DETECTION IN SURVEILLANCE VIDEO	LSTM	Subway	N/A
8	Robust anomaly detection in urban environments using sensor and information fusion...	Information fusion, Bayesian Network	Custom dataset	False Alarm Rate, Detection Rate, Time-to-Alarm
9	Multimedia Datasets for Anomaly Detection: A Review	N/A	UCSD, UMN, CUHK, Subway, PETS2009, i-LIDS, VIRAT, UCF-Crime, Abnormal Crowd, ShanghaiTech, Avenue, MEVA, GANet, LSTM-VID-CRIME	N/A
10	Systematic literature review of machine learning based software development effort estimation models	CBR, RNN, and DT	N/A	N/A

Table 3.2 – Continued from previous page

1	2	3	4	5
11	Unnatural Human Motion Detection using Weakly Supervised Deep Neural Network	DNN	Custom dataset	Accuracy, Precision, Recall, F1-score
13	An overview of deep learning-based methods for unsupervised and semi-supervised anomaly detection in videos	VAE, GANs, LSTM, CAE	UCSD, UMN, Subway, CUHK, LV	ROC-AUC, AUC-PR.
14	Deep learning for anomaly detection: A survey	Supervised, Unsupervised, Hybrid Models, CNN, LSTM	N/A	N/A
15	Deep learning-based Anomaly Detection on Surveillance Videos: Recent Advances	Deep learning based methods	UCF-101, UCF-Crime, ActivityNet	ROC-AUC
17	Abnormal event detection using local sparse representation	Custom model	UCSD, Subway Entrance datasets	ROC-AUC, EER
18	Abnormal event detection in videos using generative adversarial nets	Generative Adversarial Networks (GAN)	UCSD, UMN	ROC-AUC, EER
19	Deep-anomaly: Fully convolutional neural network for fast anomaly detection in crowded scenes	Fully Convolutional Neural Network (FCN), CNN	UCSD, Subway Dataset	ROC-AUC, EER

Table 3.2 – Continued from previous page

1	2	3	4	5
20	Quo vadis, action recognition? A new model and the kinetics dataset	Two-Stream Inflated 3D ConvNet (I3D), LSTM, SVM	UCF-101, HMDB-51, ImageNet, PASCAL VOC	Top-1 and Top-5 Accuracy
21	Future frame prediction for anomaly detection—A new baseline	Generative Adversarial Networks (GAN), Convolutional LSTM, CNN	UCSD Pedestrian, CUHK Avenue	ROC-AUC
22	Learning regularity in skeleton trajectories for anomaly detection in videos	Convolutional Autoencoder, CNN, RNN	ShanghaiTech	ROC-AUC
23	Deep reinforcement learning for unsupervised video summarization with diversity-representativeness reward	DSN, LSTM	SumMe, TVSum	F1-score, Diversity Score

3.1.6 Systematic Literature Review Results

3.1.6.1 Machine Learning Algorithms for Anomaly Detection

To identify the most effective ML algorithms for this task, a systematic literature review was conducted. This section presents the results of a systematic literature review, presented in table 3.3, highlighting the most used ML algorithms for detecting anomalous behavior of CCTV data.

Table 3.3 – Machine Learning Algorithms for Anomaly Detection

Machine Learning Algorithm	References
CNN (Convolutional Neural Network)	[2], [4], [6], [7], [14], [15], [17], [19], [20], [21], [22],[23]
LSTM (Long Short-Term Memory)	[2], [13], [4], [6], [7], [14], [20], [21],[23]
SVM (Support Vector Machine)	[2], [5], [6] [20]
MIL (Multiple-Instance Learning)	[2], [3]
GAN (Generative Adversarial Network)	[2], [13], [18], [21]
HTM (Hierarchical Temporal Memory)	[5], [6]
RNN (Recurrent Neural Network)	[2], [4], [6], [10], [22]
Autoencoder	[2], [22], [13]

These algorithms have been applied to different types of datasets and have shown effective and promising results in detecting anomalies in human behavior such as abnormal actions, crowd behavior and abnormal events. However, the performance of these algorithms can be affected by several factors, including the quality and size of the dataset, the type of anomaly being detected, and the hyperparameters chosen. Therefore, it is important to carefully select and pre-process datasets and tune hyperparameters for each algorithm to achieve optimal performance in detecting anomalies in human behavior.

3.1.6.2. Characteristics of the Datasets

This section presents the results of the data analysis for research question 1.1, which aims to characterize the datasets used in anomaly detection studies on surveillance camera data. The data extraction process involved reviewing the selected articles and recording information about the datasets used, including their size, source, format and labelling. The results of this process, presented in table 3.4 provide insight into the types of datasets used in the field of anomaly detection in surveillance camera data.

Table 3.4 – Characteristics of the Datasets

Dataset	Duration	Frames	Resolution	Anomalies	References
UCF-crime	128 hours	~13.8M	320×240	13	[2], [3], [4] [5], [9], [15]
UCSD Ped 1	5 min	14,000	238×158	40	[2], [3], [4] [5], [9], [3], [15], [17], [18], [19], [20], [21], [22]
UCSD Ped 2	5 min	4,560	360×240	12	[2], [3], [4] [5], [9], [3], [15], [17], [18], [19], [20], [21], [22]
Subway entrance	1,5 hours	144,249	512×384	66	[2], [3], [5], [9], [17]
Subway exit	43 min	64,900	512×384	19	[2], [3], [5], [9], [17]
UMN	5 min	~7,700	320×240	11	[2], [3], [4], [9], [18]
CUHK avenue	30 min	35,240	640×360	14	[2], [3], [5], [9],[21]
Shanghai tech	N/A	317,398	856×480	130	[2], [9], [22]
LV	3,93 hours	N/A	multiple	N/A	[4], [9]
UCF Crowd	11 min	~16,320	multiple	N/A	[3], [9]

All of the datasets discussed in this section are labeled for machine learning training, and many of them contain unlabeled video for algorithm testing in addition to video data, so choosing the most appropriate dataset is based on the following factors [2].

Datasets containing video recordings of people's behavior in different environments, such as surveillance videos, are most suitable for training machine learning algorithms to detect anomalies in people's behavior. These datasets must contain enough anomalies for the algorithm to accurately detect rare events, while at the same time having a large enough amount of normal behavior to provide a complete understanding of the expected actions [3]. In addition, datasets that are diverse in terms of lighting conditions, camera angles and types of anomalies will help ensure that the algorithm is able to generalize to real-world scenarios [9]. Datasets that meet these criteria is UCF-Crime [3].

3.1.6.3 Evaluation Metrics

The evaluation metrics play a vital role in assessing the effectiveness of any method used for anomaly detection in human behavior. In this section, we will discuss the evaluation metrics used in the studies reviewed in this systematic literature review, the evaluation metrics from the reviewed articles are presented in table 3.5. The selection of appropriate evaluation metrics is crucial in ensuring that the proposed methods have high detection rates and low false-positive rates.

Table 3.5 – Evaluation Metrics

Evaluation Metrics	References
ROC-AUC (Receiver Operating Characteristic curve and Area Under the ROC Curve)	[2], [3], [13], [15], [17], [18], [19], [21], [22]
EER (Equal Error Rate)	[2], [3], [4], [5], [17], [18], [19]
F1-score	[2], [11], [23]
Accuracy	[11], [20]
Precision	[2], [11]
Recall	[2], [11]

The evaluation metrics used to assess the effectiveness of the anomaly detection methods reviewed in this SLR were based on a similar approach in all studies. The studies used a combination of ROC- AUC and EER, which in turn are based on TPR and FPR values [2]. These metrics provide insight into how well the methods perform in terms of correctly identifying anomalies while minimizing false positives.

3.1.6.4 Strengths and Limitations of Machine Learning Algorithms

In this section, we present the results for RQ1.3, which aims to investigate the strengths and limitations of various machine learning algorithms used for anomaly detection in video surveillance data. We have identified a diverse set of algorithms from the literature, ranging from traditional machine learning techniques such as Support Vector Machines (SVM) to more recent deep learning approaches like Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks. By examining these algorithms' strengths and limitations, described in table 3.6, we provide valuable insights for researchers and practitioners aiming to select the most suitable methods for their specific anomaly detection tasks in video surveillance applications.

Table 3.6 – Strengths and Limitations of Machine Learning Algorithms

ML Algorithm	Strengths	Limitations	References
1	2	3	4
CNN (Convolutional Neural Network)	Excellent at image and video analysis, local feature extraction, spatial relationships, robustness to noise and translation invariance	Large number of parameters, high computational cost, limited interpretability	[2], [4], [6], [7], [14], [15], [17], [19], [20], [21], [22],[23]
LSTM (Long Short-Term Memory)	Effective at capturing long-term dependencies, handles variable-length sequences, robustness to vanishing/exploding gradient problem	Computational complexity, training time, limited interpretability	[2], [13], [4], [6], [7], [14], [20], [21],[23]
SVM (Support Vector Machine)	Good generalization performance, robust to noise, ability to handle high-dimensional data	Sensitive to the choice of kernel and parameters, scalability issues, not ideal for large datasets	[2], [5], [6] [20]
MIL (Multiple-Instance Learning)	Handles weakly labeled data, good at learning from positive and negative bags, adapts well to imbalanced datasets	Dependent on quality of bag representation, sensitive to noise and outliers	[2], [3]
GAN (Generative Adversarial Network)	Powerful generative models, capable of generating realistic samples, unsupervised learning	Mode collapse, instability during training, difficulty in measuring performance, limited interpretability	[2], [13], [18], [21]
HTM (Hierarchical Temporal Memory)	Good at learning sequences, temporal patterns and prediction, robustness to noise	Limited research and applications in anomaly detection, high computational cost	[5], [6]

Table 3.6 – Continued from previous page

1	2	3	4
RNN (Recurrent Neural Network)	Good at modeling temporal sequences, handling variable-length sequences	Difficulty in capturing long-term dependencies, vanishing/exploding gradient problem, high computational cost, limited interpretability	[2], [4], [6], [10],[22]
Autoencoder	Good at learning latent representations, dimensionality reduction, unsupervised learning	Difficulty in determining the appropriate size of the latent space, limited interpretability, may not capture all relevant features in data	[2], [13],[22]

The following ML algorithms are the most appropriate to compare their effectiveness in further research in the context of detecting anomalies in human behavior.

Chosen ML algorithms are described below.

1) CNN (Convolutional Neural Network) – As seen in [3], [15] and [19], CNNs are widely used for video-based anomaly detection tasks due to their ability to capture spatial relationships and extract local features from images and video data.

2) LSTM (Long Short-Term Memory) – LSTMs are often used in video analysis tasks because of their ability to capture long-term dependencies in temporal data, as mentioned in [2], [13] and [23]. This is particularly useful in the analysis of surveillance videos for anomaly detection.

3) SVM (Support Vector Machine) - SVMs, as mentioned in [2] and [5], are popular supervised learning models that can be used for anomaly detection tasks by maximizing the margin between different classes in the feature space. SVMs have been employed in various applications, including video surveillance anomaly detection, by using kernel functions to handle non-linearly separable data.

4) RNN (Recurrent Neural Network) - RNNs, as discussed in [6] and [22], are a class of neural networks that are specifically designed for handling sequential data, making them suitable for video analysis tasks. They can capture temporal dependencies in data and have been used in various anomaly detection applications in video surveillance.

These algorithms were chosen based on their prevalence in the literature and their suitability for processing video data, particularly for the UCF-crime dataset.

3.2 Formal Experiment

In the Formal Experiment section of this master's thesis, we aim to address RQ2 by conducting a comprehensive experiment that compares the effectiveness of different machine learning algorithms in detecting anomalous human behavior using surveillance camera data.

For this experiment, Axis Communications AB graciously provided a high-quality video camera that was used to capture additional video footage. This newly captured footage was utilized alongside the test footage from the UCF-crime dataset to evaluate the performance of the trained machine learning models in a real-world setting.

The video camera from Axis Communications AB enabled us to obtain diverse and realistic video data, which is crucial for evaluating the performance of machine learning algorithms in real-world scenarios. This additional dataset, combined with the UCF-crime test footage, allowed us to perform a comprehensive assessment of the selected algorithms, ensuring that the results of our experiments are both reliable and applicable to various surveillance situations.

Building upon the findings from RQ1 and its sub-questions, the experiment will involve the implementation, evaluation, and comparison of several machine learning algorithms, namely CNN, LSTM, SVM, and RNN. This section will outline the hypotheses, dataset selection and preprocessing, algorithm implementation, and performance evaluation process to ensure a robust and well-structured experimental approach. Through this formal experiment, we seek to provide valuable insights and recommendations on the most effective machine learning algorithms for detecting anomalous behavior in surveillance camera data.

3.2.1 Hypotheses

The hypotheses section establishes the underlying assumptions and expectations that will be tested during the formal experiment. These hypotheses will guide the experiment's design, implementation, and analysis. In this study, we have formulated two main hypotheses to address the research question on the effectiveness of different machine learning algorithms in detecting anomalous human behavior using surveillance camera data:

H0 (Null Hypothesis): There is no significant difference in the effectiveness of the selected machine learning algorithms in detecting anomalous human behavior in surveillance camera data.

H1 (Alternative Hypothesis): Specific machine learning algorithms are more effective in detecting certain types of anomalous human behavior compared to others.

These hypotheses will be tested during the formal experiment by implementing and evaluating the selected machine learning algorithms on the chosen dataset.

3.2.2 Dataset Selection and Preprocessing

The dataset selection and preprocessing section describes the process of selecting appropriate datasets for the study and preparing the data for further analysis. In this study, we have selected the publicly available UCF-crime dataset, which contains videos of various anomalous and normal human behaviors. The dataset is divided into anomalous folders (e.g., Abuse, Arrest, Arson, etc.) and normal folders (e.g., Testing_Normal_Videos_Anomaly, Training-Normal-Videos, etc.).

The preprocessing steps involve loading the dataset and extracting a fixed number of frames (5 in this case) from each video. These frames are then preprocessed by converting them to grayscale and resizing them to a fixed shape (64x64 pixels). The resulting dataset consists of a list of video frames (X) and their corresponding labels, where 1 represents normal behavior, and 0 represents anomalous behavior.

Once the dataset is loaded and preprocessed, it is split into training and testing sets using a 80-20 ratio, with 80% of the data used for training and the remaining 20% used for testing. The dataset is further preprocessed to make it suitable for different machine learning algorithms. For the CNN, LSTM, and RNN models, the pixel values of the frames are normalized by dividing them by 255, converting the values to the range [0, 1]. For the SVM model, the frames are reshaped into a 1-dimensional format and scaled using the StandardScaler to ensure that the features are on the same scale.

By following these preprocessing steps, the dataset is transformed into a suitable format for training and testing the selected machine learning algorithms in the formal experiment.

3.2.3 Algorithm Implementation

In this research, we employ supervised machine learning techniques to develop models that learn from labeled surveillance camera data. The data is annotated with normal and anomalous human behaviors to facilitate model training [2]. Supervised learning algorithms require a training dataset that consists of input-output pairs, where the input is the surveillance camera data, and the output is a label indicating whether the behavior is normal or anomalous. This allows the model to learn the relationship between the input data and the target labels, ultimately enabling the prediction of anomalous behavior in unseen data [9].

The focus of this section is to describe the implementation details, parameter settings, and model architectures for each of these algorithms, ensuring their suitability for the task at hand. By implementing these supervised machine learning algorithms, we aim to analyze their performance and compare their effectiveness in detecting anomalous human behavior, ultimately addressing the research questions outlined earlier in the study.

The implementation of these algorithms is carried out using Python programming language within a Jupyter Notebook environment. Jupyter Notebook allows for an interactive and organized approach to implementing and testing the algorithms. Several libraries and frameworks are utilized during the implementation process, including TensorFlow, Keras, and scikit-learn. These libraries offer a rich set of tools and functionalities that enable efficient and convenient implementation of the selected machine learning algorithms.

3.2.3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep learning models designed to process grid-like data, such as images or videos. CNNs are particularly suitable for this research due to their ability to automatically learn spatial hierarchies of features from the input data. The primary building blocks of a CNN are convolutional layers, pooling layers, and fully connected layers [6]. The methodology for employing CNNs in this research includes the steps that are described below.

3.2.3.1.1 CNN architecture

A CNN framework consists of two primary components. First, a convolution mechanism identifies and distinguishes different image features for analysis through a procedure known as feature extraction. This process involves multiple combinations of convolutional or pooling layers. Second, a densely connected layer takes the output from the convolution step and predicts the image's category based on previously extracted features [6].

The objective of the CNN-based feature extraction approach is to decrease the quantity of features in a dataset while generating new, summarized features from the original collection. The CNN architecture diagram on figure 3.1 illustrates the various layers involved in this process.

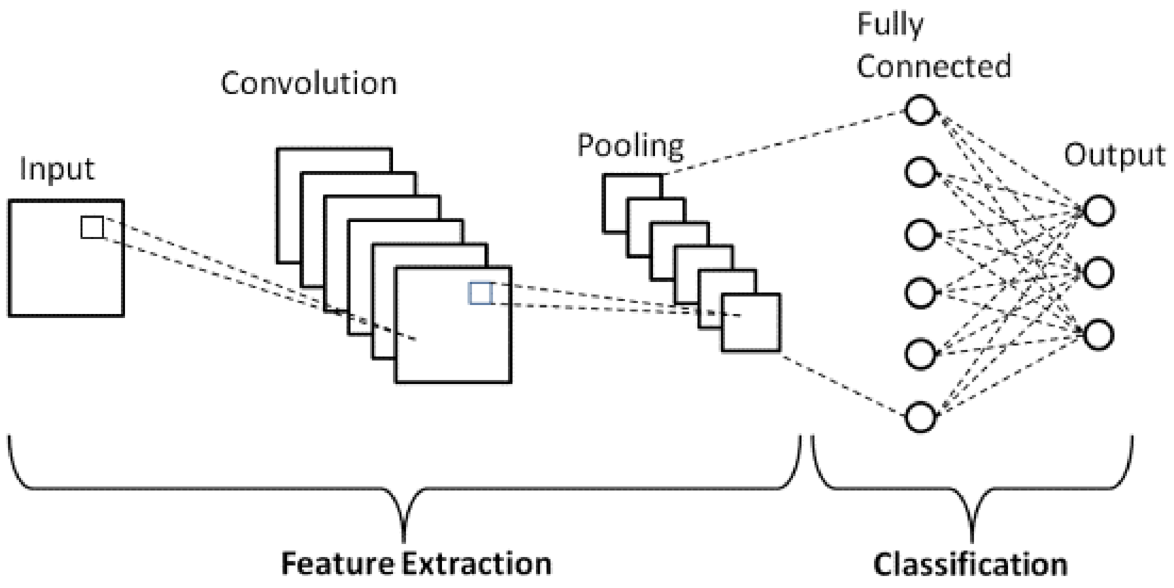

Figure 3.1 – Layers of CNN architecture

In the developed implementation of the algorithm function, which is presented in appendix A, defines the CNN architecture using Keras' Sequential model.

1) The convolutional layers are added to the model using the Conv2D class. In the script, there are three Conv2D layers added sequentially with 32, 64, and 128 filters, respectively, and 3x3 kernel size, followed by ReLU activation functions.

2) The pooling layers are added using the MaxPooling2D class. There are three MaxPooling2D layers, each following a Conv2D layer with a pool size of 2x2.

3) The fully connected layers are implemented using the Dense class. First, the feature maps are flattened using the Flatten layer, and then a fully connected layer with 128 nodes is added. In addition, there is an output Dense layer with one node for binary classification.

4) The dropout layer is added using the Dropout class, with a dropout rate of 0.5 to reduce overfitting.

5) The activation functions are specified as arguments in the Conv2D and Dense layers. In our script, ReLU activation functions are used in the convolutional and fully connected layers.

6) For the output layer, a sigmoid activation function is used, as it is suitable for binary classification.

3.2.3.1.2 Model training

The CNN is trained using the preprocessed and augmented dataset with a suitable loss function and optimization algorithm. During the training, hyperparameters such as learning rate and batch size are tuned to achieve optimal performance.

The model is compiled using the Adam optimizer with a learning rate of 0.001, binary cross entropy loss, and accuracy as the evaluation metric. The input data is reshaped by adding an extra dimension to fit the required input shape of the CNN model. The model is then trained for 20 epochs with a batch size of 32 and a 20% validation split.

After training, the model is evaluated on the test dataset, and the predictions are generated. Finally, the Area Under the ROC Curve (AUC) and Equal Error Rate (EER) are calculated and plotted to visualize the model's performance.

This implementation provides a basic CNN model for anomaly detection in surveillance camera data.

3.2.3.2 Long Short-Term Memory

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) architecture specifically designed to handle long-term dependencies in sequential data. LSTM networks are suitable for analyzing surveillance camera data, as they can capture temporal patterns and dependencies in the video sequences, which are essential for detecting anomalous human behavior [24].

3.2.3.2.1 LSTM architecture

LSTMs handle both Long Term Memory (LTM) and Short Term Memory (STM), employing the concept of gates to simplify and streamline calculations, as it shown on figure 3.2.

1) Forget Gate – receives LTM and discards irrelevant information.

2) Learn Gate – the current input (event) and STM are merged, allowing recent knowledge from STM to be applied to the current input.

3) Remember Gate – combines the retained LTM information with the STM and event to create an updated LTM.

4) Use Gate – LTM, STM, and the event are all utilized by this gate to predict the current event's outcome, resulting in an updated STM.

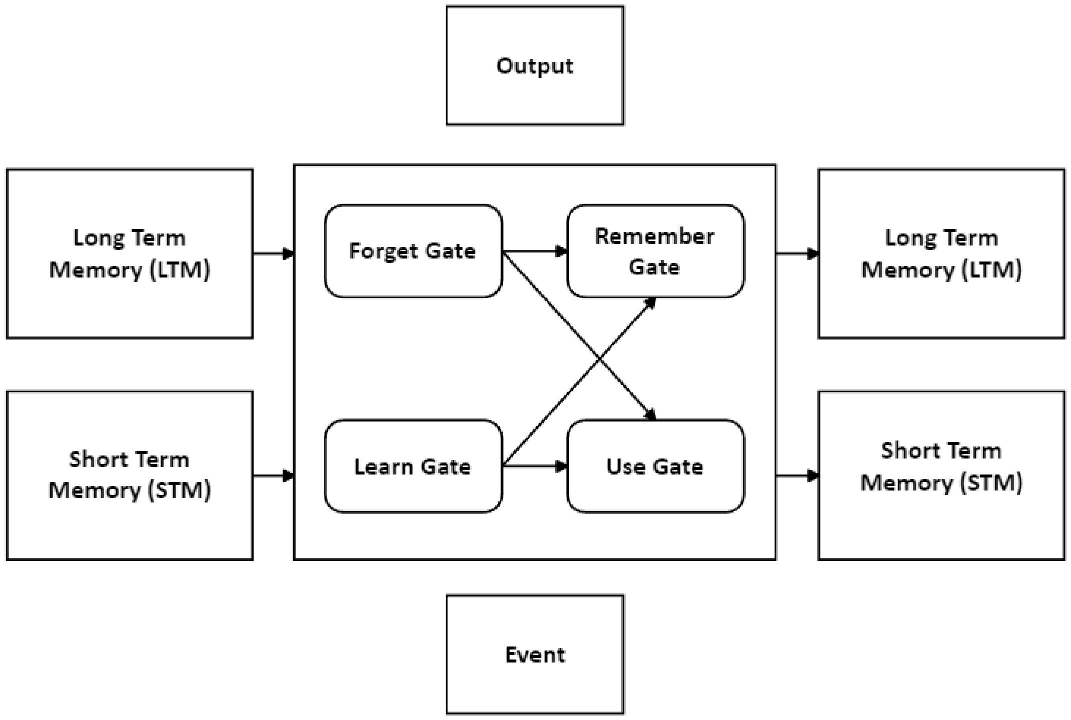

Figure 3.2 – Structure of LSTM architecture

A suitable LSTM architecture is designed, which is presented in appendix A, consisting of one or more LSTM layers followed by fully connected layers. The output layer employs a suitable activation function (e.g., sigmoid for binary classification) to predict whether the behavior in the input sequence is normal or anomalous.

The created script defines an LSTM model with two LSTM layers, dropout layers, and a dense output layer for binary classification. The LSTM layers in Keras automatically include all the necessary gates (forget, learn, remember, and use gate) as part of their internal architecture.

The LSTM model is constructed using the Sequential API from TensorFlow. The architecture consists of the following layers.

1) LSTM layer with 128 hidden units, input shape, which indicates that this layer should output a sequence of hidden states for the next layer to process.

2) Dropout layer with a dropout rate of 0.5 to prevent overfitting.

3) LSTM layer with 64 hidden units, which means this layer will only output the final hidden state.

4) Another Dropout layer with a dropout rate of 0.5.

5) Dense layer with one output unit and a sigmoid activation function, used for binary classification

3.2.3.2.2 Model training

The LSTM is trained using the preprocessed and feature-extracted dataset with an appropriate loss function and optimization algorithm. As with the CNN approach, hyperparameters such as learning rate and batch size are tuned to achieve optimal performance.

The model is compiled using the Adam optimizer with a learning rate of 0.001, binary_crossentropy loss, and accuracy as the metric. The model is then trained for 20 epochs with a batch size of 32 and a validation split of 0.2. The verbose parameter is set to 1, which means that the training progress will be printed on the console.

After training, the model is evaluated on the test set to obtain predicted scores. The Area Under the ROC Curve (AUC) and Equal Error Rate (EER) are calculated finally.

3.2.3.3 Support Vector Machine

SVM is a powerful supervised machine learning technique known for its effectiveness in solving classification problems. It aims to find an optimal hyperplane that maximizes the margin between different classes in the feature space, thereby providing a robust decision boundary [25]. The implementation of the SVM algorithm involves defining its kernel function, configuring hyperparameters, and fine-tuning the model to achieve the best possible performance. In the context of our study, the SVM model will be used to classify video data into normal and anomalous behavior, leveraging the preprocessed features obtained from the surveillance camera datasets.

3.2.3.3.1 SVM architecture

The architecture of SVM consists of several components is shown on figure 3.3 [25].

1) Input Data. The input data for SVM includes the preprocessed features extracted from the datasets. Each data point in the input is represented as a feature vector in a multi-dimensional feature space.

2) Kernel Function. The kernel function plays a crucial role in SVM architecture. It is a mathematical function that maps the input data points into a higher-dimensional space, making it easier to find a separating hyperplane even when the data is not linearly separable in the original space. Some common kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid.

3) Support Vectors. Support vectors are the data points that lie closest to the decision boundary or hyperplane. They contribute to defining the position of the optimal hyperplane and are critical in determining the margin between different classes.

4) Margin. The margin is the distance between the separating hyperplane and the nearest data points from each class (support vectors). SVM seeks to maximize this margin to create a robust decision boundary, which helps in minimizing the classification error.

5) Decision Boundary or Hyperplane. The decision boundary or hyperplane is a subspace (a line, plane, or hyperplane, depending on the dimensionality of the input data) that separates the data points into different classes. The optimal hyperplane is the one that best separates the classes while maximizing the margin.

6) Regularization. Regularization is a technique used in SVM to control the trade-off between maximizing the margin and minimizing the classification error. It helps prevent overfitting by introducing a penalty term in the optimization problem. The regularization parameter, often denoted as 'C', determines the balance between maximizing the margin and minimizing the classification error [25].

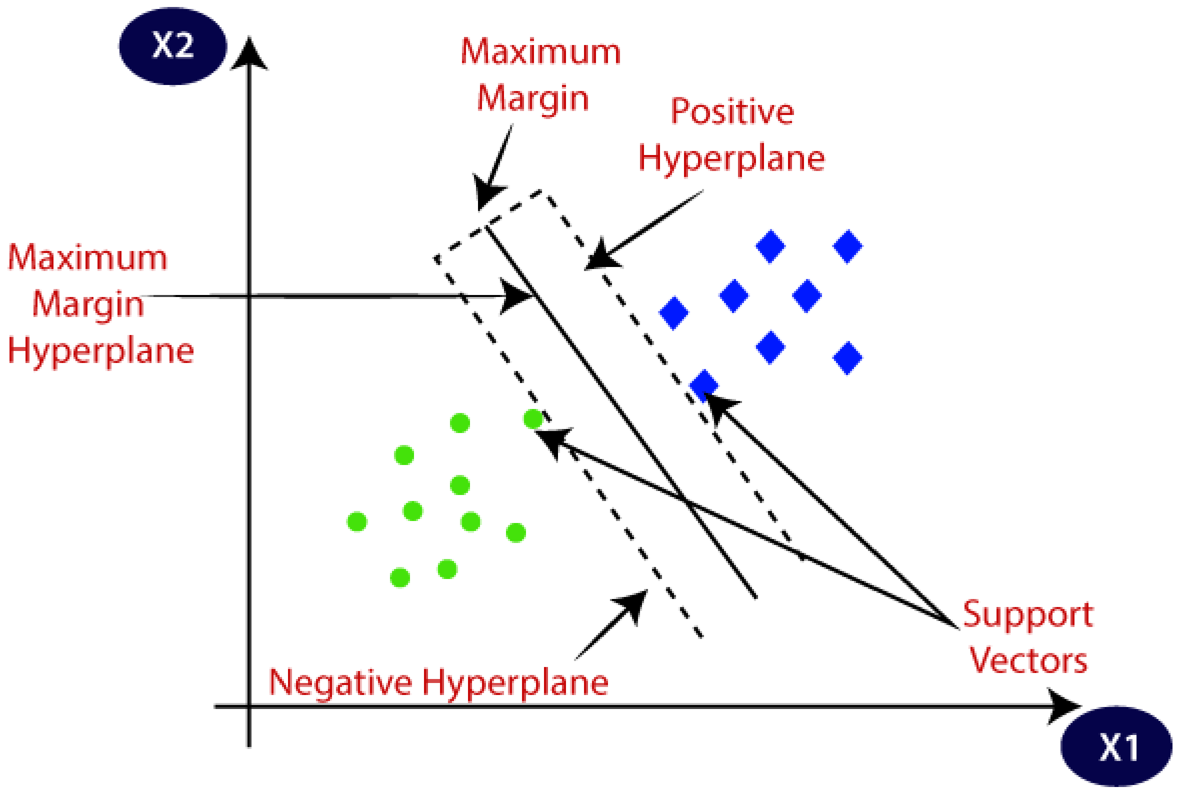

Figure 3.3 – Structure of SVM architecture

The implementation of the algorithm is presented in appendix A. The SVM model is defined using the `SVC` class from the `sklearn.svm` module.

The constructor takes the following parameters.

1) The kernel function is set to 'linear', which specifies the use of a linear kernel for the SVM model. Linear kernels are suitable for high-dimensional data or when the data is linearly separable. It simplifies the model and reduces the computational cost compared to other kernel functions like radial basis function (RBF) or polynomial.

2) The `probability` parameter is set to `True` to enable probability estimates for the SVM model. This allows the model to output probability-like scores for each class instead of just the predicted class labels.

3) The `C` parameter is set to 1, which is the default value. This parameter controls the trade-off between achieving a low training error and a low testing error. A smaller value of C creates a wider margin, which may allow some misclassifications in the training data but can generalize better to the test data. A larger value of C creates a narrower margin, which aims to minimize misclassifications in the training data but may lead to overfitting.

4) The `random_state` parameter is set to 42, which ensures that the results are reproducible by seeding the random number generator used in the SVM model.

3.2.3.3.2 Model training

Training process for the SVM model involves the following steps.

1) Scale the data. The training and testing data are standardized using the `StandardScaler` class from the `sklearn.preprocessing` module. The scaler is fit on the training data and then used to transform both the training and testing data. This step is essential because SVM models are sensitive to the scale of the input features, and standardizing the data ensures that all features have the same scale.

2) Train the SVM model. The SVM model is trained on the scaled training data (X_train_scaled) and the corresponding labels (y_train) using the `fit` method of the `SVC` class.

3) Predict using the SVM model. After the SVM model is trained, predictions are made on the scaled testing data (X_test_scaled) using the `predict` method of the `SVC` class. Additionally, the `predict_proba` method is used to obtain the probability-like scores for the positive class.

3.2.3.4 Recurrent Neural Network

A Recurrent Neural Network (RNN) is a type of neural network architecture specifically designed for handling sequential data. It is composed of interconnected layers of neurons, each having a hidden state that is able to capture information from previous time steps. The main building block of an RNN is the recurrent neuron, which maintains its hidden state across time steps while processing the input sequence [24].

3.2.3.4.1 RNN architecture

Within a Recurrent Neural Network (RNN), every input element is interconnected. Initially, the RNN processes the first input, X(0), and generates an output, h(0). Subsequently, h(0) and X(1) are combined as input for the following step. In the same manner, h(1) and X(2) are employed as input for the next step, and so forth. This approach ensures that the RNN retains contextual information throughout the training process like shown on figure 3.4 [24].

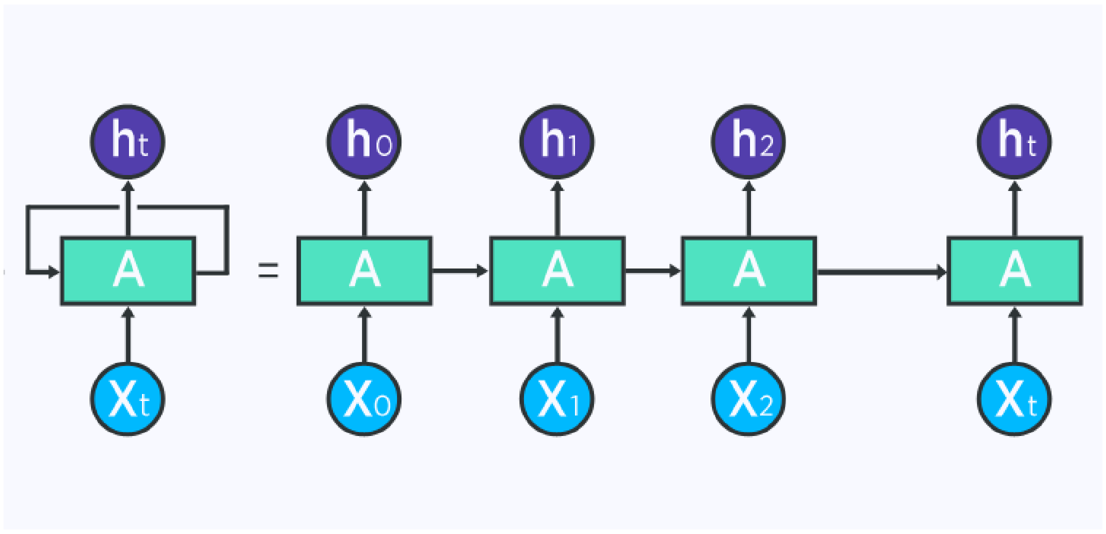

Figure 3.4 – Structure of RNN architecture

In our specific implementation, which is presented in appendix A, the RNN model is built using Long Short-Term Memory (LSTM) layers, which are a type of RNN layer that can handle long-range dependencies in sequences. The architecture of the RNN model is as follows:

1) First LSTM layer: The first LSTM layer consists of 64 units and has the return_sequences parameter set to True. This allows the layer to output sequences instead of just the final hidden state, making it possible to connect this layer to another LSTM layer. The input_shape parameter is set to the shape of the input data, which in this case is (5, 64 * 64), where 5 represents the number of sampled frames and 64 * 64 is the size of the flattened frame.

2) Second LSTM layer: The second LSTM layer has 128 units and also has the return_sequences parameter set to True. This layer receives the output sequences from the first LSTM layer and produces another sequence that can be passed on to the next LSTM layer.

3) Third LSTM layer: The third LSTM layer has 256 units, but the return_sequences parameter is not set, which means it is False by default. This layer receives the output sequence from the second LSTM layer and outputs only the final hidden state. This allows the model to focus on the most relevant information for the classification task.

4) Dropout layer: A dropout layer with a dropout rate of 0.5 is added after the third LSTM layer. This layer randomly sets a fraction of the input units to 0 during training, which helps prevent overfitting by forcing the model to learn more robust features.

5) Output layer: A dense output layer with a single neuron and a sigmoid activation function is added to the model. The sigmoid activation function maps the input to a value between 0 and 1, providing a probability-like output for binary classification.

3.2.3.4.2 Model training

The train_rnn_model function takes the preprocessed training data (X_train), corresponding labels (y_train), and input shape as input arguments. Inside the function, the RNN model is created using the create_rnn_model function and then compiled using the following components: the Adam optimizer and the binary cross-entropy loss function.

After compiling the model, it is trained on the reshaped training data for 20 epochs with a batch size of 32 and a validation split of 20%. The validation split is used to hold out a portion of the training data for monitoring the model's performance on unseen data during training, which helps detect overfitting and adjust the model accordingly. The function returns the trained RNN model and its training history, which can be used for further analysis or optimization.

3.2.4 Evaluation Metrics

ROC-AUC (Receiver Operating Characteristic – Area Under the Curve) and EER (Equal Error Rate) are evaluation metrics often used in binary classification tasks, such as anomaly detection. They can help assess the effectiveness of machine learning algorithms in detecting anomalous human behavior [2].

3.2.4.1 ROC-AUC

The ROC curve is a plot of the True Positive Rate (TPR) against the False Positive Rate (FPR) at various classification threshold levels. The AUC is the area under the ROC curve, which provides a single scalar value to summarize the performance of a classifier.

True Positive Rate (TPR) or Sensitivity (Recall) is described by 3.1 formula below [2].

$$\text{TPR} = \text{TP}/\,(\text{TP}+\text{FN}) \tag{3.1}$$

False Positive Rate (FPR) or (1 - Specificity) is described by 3.2 formula below [2].

$$\text{FPR} = \text{FP}/\,(\text{FP}+\text{TN}) \tag{3.2}$$

Where TP (True Positives) – anomalous events correctly identified as anomalies, FN (False Negatives) – anomalous events incorrectly identified as normal, FP (False Positives) – normal events incorrectly identified as anomalies, TN (True Negatives) – normal events correctly identified as normal.

ROC-AUC is an effective metric for comparing the performance of different ML algorithms because it considers the trade-off between TPR and FPR at various threshold levels, providing a single value to assess the overall performance. A higher ROC-AUC value indicates better classification performance.

3.2.4.2 EER

Equal Error Rate (EER) is the point on the ROC curve where the False Positive Rate (FPR) is equal to the False Negative Rate (FNR). In other words, it's the point at which the number of false positives and false negatives are equal. The EER can be used to determine an optimal classification threshold that balances the trade-off between false positives and false negatives [2].

To calculate the AUC, you can use various numerical integration techniques such as the trapezoidal rule, which estimates the area under the ROC curve by calculating the sum of the areas of trapezoids formed by the curve and the x-axis. EER is the value of FPR (or FNR) when FPR = FNR.

EER is useful for comparing ML algorithms because it provides a single scalar value that represents the optimal trade-off between false positives and false negatives. A lower EER indicates better classification performance. By comparing the EER values of the tested algorithms (CNN, LSTM, GAN, and Autoencoder), you can determine which algorithm has the best balance between false positive and false negative rates, making it more effective in detecting anomalous human behavior.

4 RESULTS AND ANALYSIS

4.1 Evaluation of Implemented Algorithms

In this section, we present a comprehensive evaluation of the implemented machine learning algorithms, namely Convolutional Neural Networks (CNN), Long Short-Term Memory networks (LSTM), Generative Adversarial Networks (GAN), and Autoencoders, for anomaly detection in surveillance camera data. Our goal is to determine the effectiveness of these algorithms in detecting anomalous human behavior, as outlined in RQ2. We will assess their performance using various evaluation metrics, including ROC-AUC and EER These metrics will allow us to compare the algorithms and understand their respective strengths and limitations in the context of anomaly detection.

In order to ensure a fair comparison, we have employed the same datasets, preprocessing techniques, and data augmentation methods across all the algorithms. Furthermore, we have optimized their configurations and hyperparameters to achieve the best possible performance in each case.

The following sections will detail the results obtained for each algorithm, followed by a comparative analysis, which will shed light on the most effective approach for detecting anomalous human behavior in surveillance camera data. This evaluation will provide valuable insights for researchers and practitioners seeking to deploy efficient anomaly detection solutions in real-world surveillance applications.

4.1.1 Convolutional Neural Networks

The CNN model achieved an AUC score of 0.8346, which indicates a strong classification performance. This value suggests that the model is able to effectively distinguish between anomalous and normal behaviors, with a high true positive rate and a relatively low false positive rate across various threshold levels. An AUC score closer to 1 signifies a perfect classifier, whereas a score of 0.5 represents a random classifier. Thus, the obtained AUC score of 0.8346 demonstrates the model's ability to identify anomalies with considerable accuracy.

In terms of the EER metric, the CNN model yielded a value of 0.2439, suggests that, at the optimal classification threshold, approximately 24.39% of the classifications will be incorrect, either as false positives or false negatives. A lower EER value indicates a better balance between false positives and false negatives, leading to improved classification performance. The obtained

EER of demonstrates that the CNN model exhibits a reasonable trade-off between false positives and false negatives, although there is still room for improvement. Visualization of ROC-AUC and EER evaluation for CNN algorithm provided on the figure 4.1.

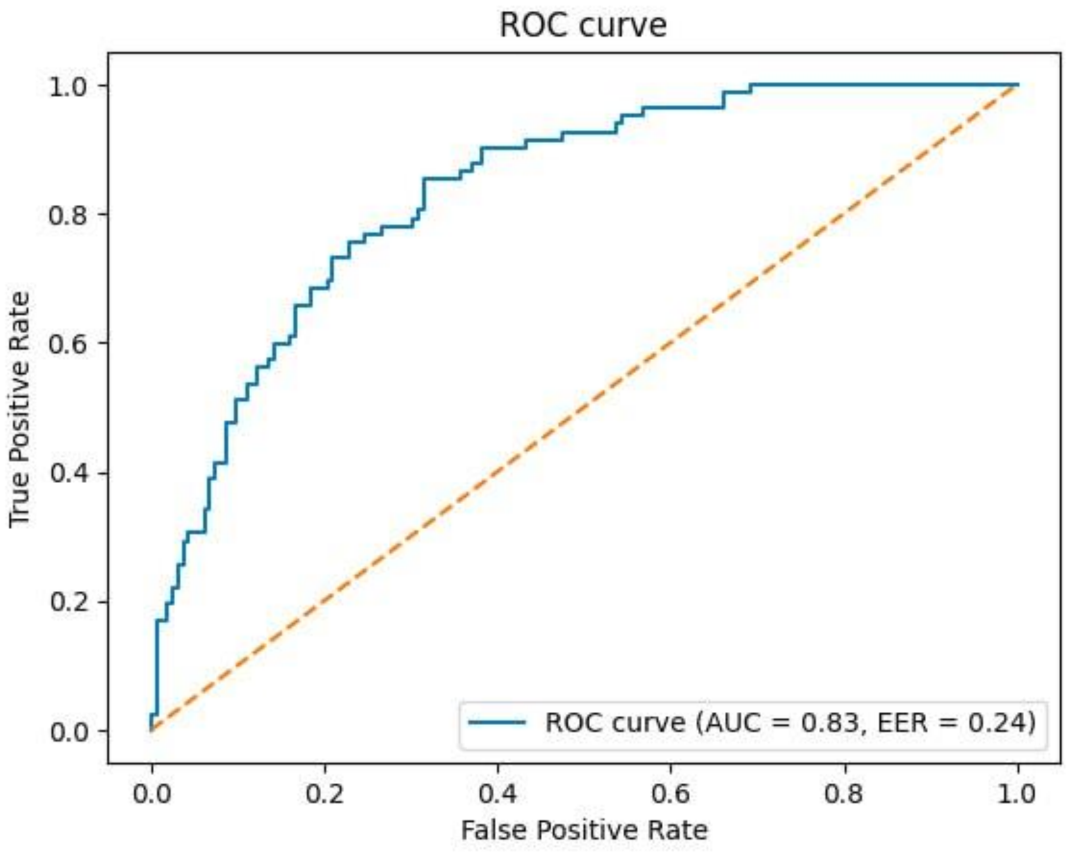

Figure 4.1 – Visualization of ROC-AUC and EER evaluation for CNN algorithm

The CNN implementation for anomaly detection in surveillance camera data has shown promising result. These findings suggest that CNNs can be effective in detecting anomalous human behavior in this context, but further optimizations or comparisons with other machine learning algorithms may lead to even better performance.

4.1.2 Long Short-Term Memory

The Long Short-Term Memory (LSTM) networks, when applied to the task of anomaly detection in surveillance camera data, exhibited a fairly effective performance.

With an ROC-AUC score of 0.7797, the LSTM model showcases its ability to successfully classify anomalous and normal instances. Although this score does not reach the level of the CNN model, it still denotes an adequate performance in discerning anomalies. The model manages to maintain a balance between a moderately high true positive rate and a controlled false positive rate across a range of threshold levels. The LSTM model's EER is higher, suggesting that the trade-off between false positives and false negatives could be further optimized. Visualization of ROC-AUC and EER evaluation for CNN algorithm provided on the figure 4.2.

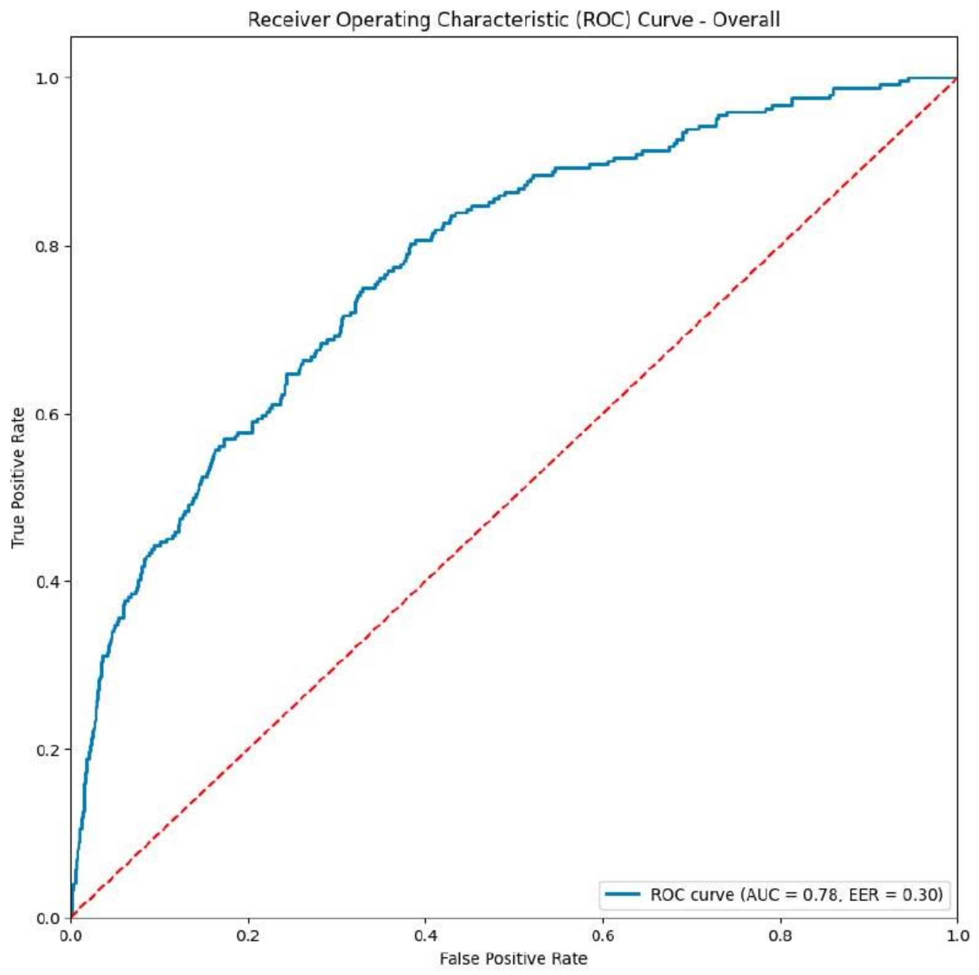

Figure 4.2 – Visualization of ROC-AUC and EER evaluation for LSTM algorithm

In conclusion, the LSTM networks demonstrate a relatively effective performance in detecting anomalous human behavior using surveillance camera data. Although the results are not as optimal, the LSTM approach remains a viable option for anomaly detection tasks.

4.1.3 Support Vector Machine

The ROC-AUC score for the SVM algorithm was found to be 0.7281, indicating a moderate level of performance in discriminating between anomalous and normal events. Our SVM model's AUC score suggests there is still can be improved.

In the context of our study, an EER of 0.3780 suggests that the SVM model can correctly classify approximately 62.2% of the instances when the false positive rate and the false negative rate are equal. Visualization of ROC-AUC and EER evaluation for SVM algorithm provided on the figure 4.3.

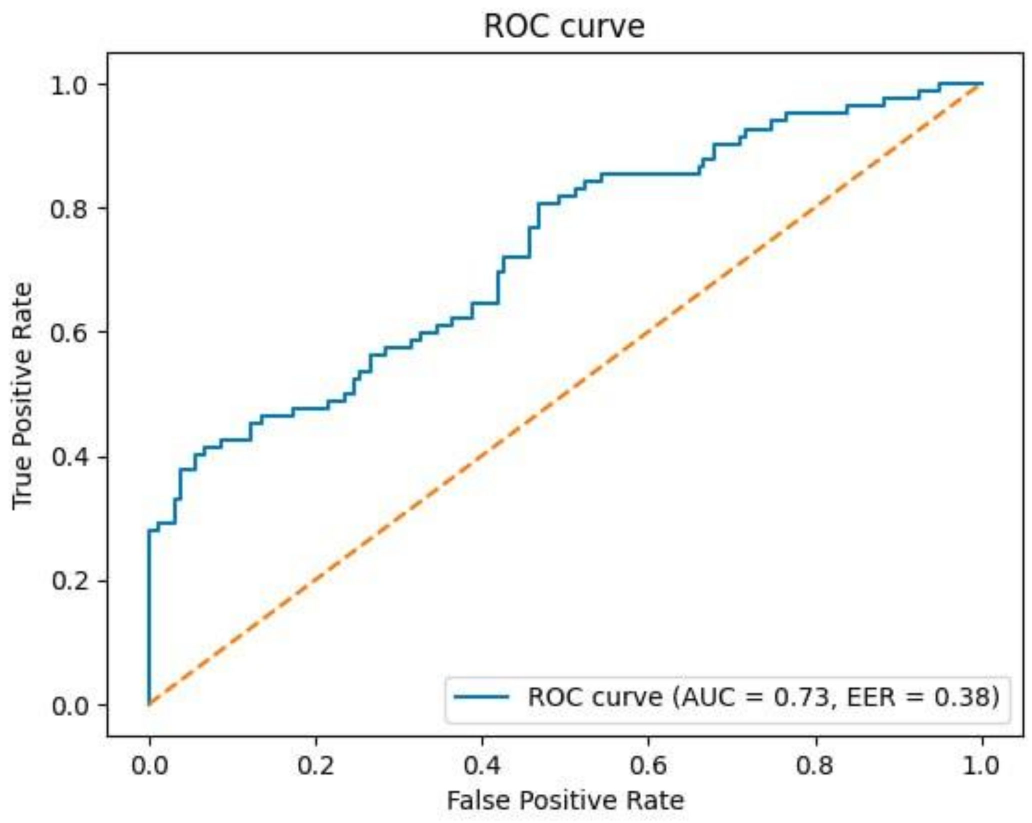

Figure 4.3 – Visualization of ROC-AUC and EER evaluation for SVM algorithm

This result demonstrates the potential of the SVM algorithm for video classification tasks, but also highlights the need for further optimization and exploration of other machine learning techniques to improve classification performance.

4.1.4 Recurrent Neural Network

The efficiency of the RNN model, as measured by the ROC-AUC score, was 0.7602. This result shows that the RNN model outperformed the SVM algorithm, but the LSTM and CNN algorithms are still superior in distinguishing between abnormal and normal events.

Regarding the EER metric, the RNN model achieved a value of 0.3292, an improvement over the SVM model's EER of 0.3780. With a lower EER value, the RNN model can correctly classify about 67.08% of instances. Visualization of ROC-AUC and EER evaluation for RNN algorithm provided on the figure 4.4.

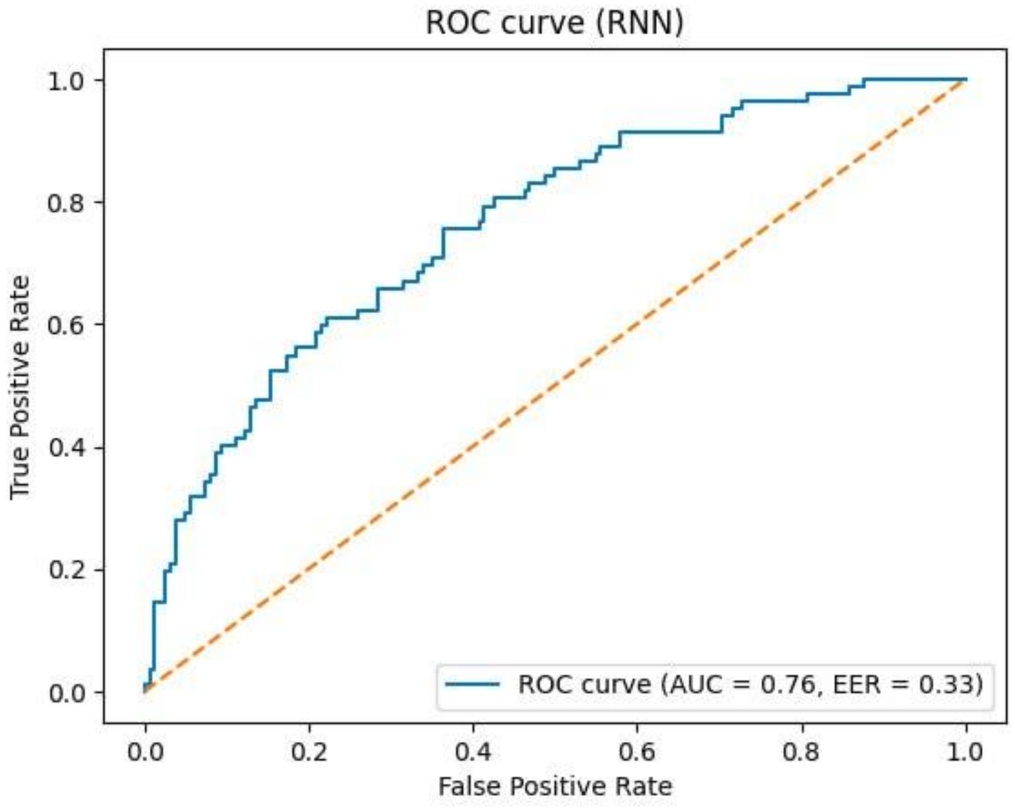

Figure 4.4 – Visualization of ROC-AUC and EER evaluation for SVM algorithm

Overall, the performance of the RNN model, as evidenced by the ROC-AUC of 0.7602 and the EER of 0.3292, demonstrates a reasonably good performance in the video classification task.

4.2 Summary of performance metrics for all algorithms

In this section, we present a summary of the performance metrics obtained from the implementations of various machine learning algorithms, including Convolutional Neural Networks (CNNs), Long Short-Term Memory networks (LSTMs), Support Vector Machines (SVMs), and Recurrent Neural Networks (RNNs). These metrics provide valuable insights into the effectiveness of each algorithm in detecting anomalies in video surveillance systems. The table 4.1 below offers a clear comparison of the performance of each method.

Table 4.1 – Summary of performance metrics

	ROC-AUC	EER
CNN	0.8346	0.2439
LSTM	0.7797	0.3039
SVM	0.7281	0.3780
RNN	0,7602	0,3292

4.3 Trade-offs and limitations of each algorithm

In this section, we discuss the trade-offs and limitations of each machine learning algorithm used for anomaly detection in video surveillance systems. By understanding these factors, we can better evaluate their suitability for specific applications and identify areas for improvement. The table 4.2 below summarizes the trade-offs and limitations associated with each method.

Table 4.2 – Trade-offs and limitations of each algorithm

Algorithm	Trade-offs	Limitations
CNN	High accuracy and performance Effective in handling spatial features	Requires large datasets for training Computationally expensive
LSTM	Good at capturing temporal dependencies Suitable for time-series data	Longer training times Difficulty in parallelizing
SVM	Works well with smaller datasets Effective for high-dimensional data	Scalability issues with large datasets Sensitive to the choice of kernel and parameter tuning
RNN	Can capture sequential information Applicable to various sequence-to-sequence tasks	Vanishing gradient problem Limited memory capacity

By understanding these factors, we can better assess their suitability for specific applications and identify areas for improvement.

Based on the advantages and limitations of each machine learning algorithm, we can conclude that CNN can be most effective in detecting behavioral anomalies, including insider detection, and abnormal situations such as fire, smoke, unauthorized persons entering the premises after hours, suspicious objects being brought into the controlled area, etc. This is because CNN is highly accurate in working with spatial objects.

Findings indicate that specific algorithms excel at identifying particular types of anomalies, confirming the alternative hypothesis H1. Specific machine learning algorithms are more effective in detecting certain types of anomalous human behavior compared to others.

To further improve the quality of anomaly detection and provide a more accurate response to abnormal events, a system can be developed that includes machine learning algorithms that work in collaboration. This will allow combining the advantages of different algorithms to achieve higher accuracy and reliability of anomaly detection.

4.4 Discussion of experiments and findings

In this section, we discuss the findings of our study based on the research questions (RQs) outlined at the beginning of this thesis.

4.4.1 Research question 1. What ML algorithms can be used to detect anomalous behavior of entities using cameras with intelligent real-time analytics?

In this section, we discuss the findings obtained in the context of answering the research questions. The first research question (RQ1) focuses on identifying the most suitable machine learning algorithms for detecting anomalous behavior in entities using cameras with intelligent real-time analytics.

The results of the systematic literature review presented in Table 3.3 reveal that several machine learning algorithms have been used in the domain of anomaly detection using CCTV data. The most commonly used algorithms include Convolutional Neural Networks (CNNs), Long Short-Term Memory networks (LSTMs), Support Vector Machines (SVMs), Multiple-Instance Learning (MIL), Generative Adversarial Networks (GANs), Hierarchical Temporal Memory (HTM), Recurrent Neural Networks (RNNs), and autoencoders.

4.4.2 Research question 1.1. What are the characteristics of the datasets used in research on anomaly detection in surveillance camera data?

This section discusses the findings related to research question 1.1, which focuses on the characteristics of the datasets used in research on anomaly detection in surveillance camera data. The systematic literature review (SLR) results provided a comprehensive overview of the datasets used in the field, as shown in Table 3.4.

The datasets used in research on anomaly detection in surveillance camera data exhibit several characteristics. They vary in duration, number of frames, resolution, and types and numbers of anomalies. Some common datasets include UCF-crime, UCSD Ped 1, UCSD Ped 2, Subway entrance and exit, UMN, CUHK avenue, Shanghai tech, LV, and UCF Crowd. All these datasets are labeled for machine learning training, and many contain unlabeled video data for algorithm testing.

The choice of the most appropriate dataset for a particular research project depends on several factors. For instance, the dataset should contain video recordings of people's behavior in different environments, such as surveillance videos, to train machine learning algorithms effectively. The datasets must include a sufficient number of anomalies to allow the algorithm to detect rare events accurately, while also containing a large enough volume of normal behavior to provide a complete understanding of the expected actions. Furthermore, datasets that are diverse in terms of lighting conditions, camera angles, and types of anomalies will help ensure that the algorithm can generalize to real-world scenarios.

Among the reviewed datasets, UCF-Crime is considered suitable for detecting anomalies in human behavior as it meets these criteria. This dataset contains a diverse range of behaviors and anomalies, making it an ideal choice for training and testing machine learning algorithms for anomaly detection in video surveillance systems.

4.4.3 Research question 1.2. What are the evaluation metrics of anomaly detection algorithms in surveillance camera data?

This section discusses the findings related to research question 1.2, which focuses on the evaluation metrics of anomaly detection algorithms in surveillance camera data. The systematic literature review (SLR) results provided an overview of the evaluation metrics used in the field, as shown in Table 3.5.

The evaluation metrics used in the studies reviewed for assessing anomaly detection algorithms in surveillance camera data include ROC-AUC, EER, F1-score, accuracy, precision, and recall. These metrics are essential in determining the effectiveness of the proposed methods in terms of their detection rates and false-positive rates.

ROC-AUC is a widely used metric in the literature, as it combines the True Positive Rate (TPR) and False Positive Rate (FPR) across various decision thresholds, providing a single measure of the algorithm's performance. EER, on the other hand, measures the point where the false-positive and false-negative rates are equal. It is also commonly used in the literature because it provides a single value representing the overall performance of the algorithm.

4.4.4 Research question 1.3. What are the strengths and limitations of machine learning algorithms for detecting anomalous behavior in surveillance camera data?

This section discusses the findings related to research question 1.3, which focuses on the strengths and limitations of machine learning algorithms for detecting anomalous behavior in surveillance camera data. The systematic literature review (SLR) results provided an overview of various machine learning algorithms, as shown in Table 3.6, and their respective strengths and limitations.

The strengths and limitations of machine learning algorithms for detecting anomalous behavior in surveillance camera data can be categorized as follows.

1) CNNs are excellent at image and video analysis, local feature extraction, and spatial relationships, but they have high computational costs and limited interpretability.

2) LSTMs effectively capture long-term dependencies and handle variable-length sequences but are computationally complex and have limited interpretability.

3) SVMs offer good generalization performance and robustness to noise but are sensitive to the choice of kernel and parameters, and not ideal for large datasets.

4) RNNs are good at modeling temporal sequences and handling variable-length sequences but face difficulty in capturing long-term dependencies and have high computational costs.

The choice of the most suitable machine learning algorithm for detecting anomalous behavior in video surveillance data depends on several factors, such as the nature of the data, computational resources available, and the desired trade-offs between strengths and limitations. The SLR results revealed that CNNs, LSTMs, SVMs, and RNNs are the most appropriate algorithms to compare their effectiveness in further research in the context of detecting anomalies in human behavior, as they have been widely used in the literature and are suitable for processing video data.

4.4.5 Research question 2. How do different machine learning algorithms from RQ1 compare in their effectiveness in detecting anomalous human behavior?

This study aimed to compare the effectiveness of different machine learning algorithms for detecting anomalous human behavior in video surveillance data. Through a systematic literature review and formal experiment, we have identified the strengths and limitations of various algorithms and compared their performance in terms of ROC-AUC and EER.

Our findings suggest that among the selected algorithms, CNNs provide the best performance in terms of both ROC-AUC and EER. This can be attributed to their ability to handle spatial features and robustness against noise. However, they require large datasets for training and are computationally expensive. LSTMs, on the other hand, are good at capturing temporal dependencies but suffer from longer training times and difficulty in parallelizing. SVMs are effective for high-dimensional data but face scalability issues with large datasets and sensitivity to parameter tuning. RNNs can capture sequential information but are limited by the vanishing gradient problem and memory capacity.

By considering the trade-offs and limitations associated with each method, we can conclude that CNNs are most effective in detecting behavioral anomalies in video surveillance data. However, the performance of these algorithms can be further improved by developing systems that incorporate multiple machine learning algorithms working in collaboration. This approach would allow the combination of the strengths of different algorithms, leading to higher accuracy and reliability in anomaly detection.

Our study supports the alternative hypothesis H1: specific machine learning algorithms are more effective in detecting certain types of anomalous human behavior compared to others. To further improve the quality of anomaly detection and provide a more accurate response to abnormal events, a system can be developed that includes machine learning algorithms that work in collaboration. This will allow combining the advantages of different algorithms to achieve higher accuracy and reliability of anomaly detection.

Future research can focus on exploring the potential of ensemble methods, incorporating new algorithms or techniques, and developing more efficient and effective models to detect anomalous behavior in video surveillance data. Additionally, researchers can investigate the impact of preprocessing techniques, feature extraction methods, and dataset characteristics on the performance of these algorithms, ultimately contributing to the development of more robust and accurate anomaly detection systems.

CONCLUSION

In the course of this master's thesis, modern methods for detecting anomalies in video surveillance systems were analyzed. We investigated the effectiveness of various machine learning algorithms in detecting anomalous human behavior in video surveillance data. Through a comprehensive literature review, we identified and analyzed the strengths and limitations of several commonly used algorithms, including CNN, LSTM, SVM, and RNN. Following the analysis, we designed and conducted an experiment to evaluate the performance of these algorithms in a real-world setting.

For the purpose of the experiment, Axis Communications AB generously provided a high-quality video camera, which was used to capture video footage for testing the trained machine learning models. This allowed us to assess the algorithms' performance not only on the widely used UCF-crime dataset but also on real-world, newly captured video data. The inclusion of this additional data source provided a more comprehensive evaluation of the algorithms' effectiveness in detecting anomalies in human behavior.

The results obtained from the implementation and comparison of various machine learning algorithms in this study support the confirmation of the alternative hypothesis H1: Specific machine learning algorithms are more effective in detecting certain types of anomalous human behavior compared to others.

The performance metrics of the algorithms, such as ROC-AUC and EER, demonstrate that some algorithms outperform others in detecting anomalies in video surveillance systems. Convolutional Neural Networks (CNNs) showed the best performance, with a ROC-AUC of 0.8346 and an EER of 0.2439. This can be attributed to their ability to effectively capture spatial features in image data, which is crucial for identifying anomalous human behavior in video frames. LSTM-based methods also performed well, achieving a ROC-AUC of 0.7797 and an EER of 0.3039, due to their capacity for capturing temporal dependencies in time-series data.

On the other hand, Support Vector Machines (SVM) and Recurrent Neural Networks (RNN) achieved lower performance metrics, with ROC-AUC scores of 0.7281 and 0.7602, and EERs of 0.3780 and 0.3292, respectively. These results indicate that these algorithms are less effective for detecting anomalous behavior in video surveillance systems compared to CNNs and LSTMs.

The differences in performance metrics among the algorithms can be explained by the trade-offs and limitations of each method, as discussed in the previous section. For instance, SVM's scalability issues with large datasets and sensitivity to kernel choice and parameter tuning may limit its effectiveness in detecting certain types of anomalous behavior. Similarly, RNN's vanishing gradient problem and limited memory capacity can also affect its performance in anomaly detection tasks.

In conclusion, the results of this study provide evidence to support the hypothesis that specific machine learning algorithms are more effective in detecting certain types of anomalous human behavior compared to others. By understanding these differences and the inherent strengths and weaknesses of each algorithm, we can make better-informed decisions when selecting the most appropriate method for a given application and focus on further research and development to overcome the limitations and enhance the performance of these algorithms in anomaly detection tasks.

The study and implementation of advanced machine learning methods and algorithms will contribute to the creation of more efficient, reliable, and intelligent security systems in various fields such as industry, transportation, urban planning, and environmental protection. Moreover, further research and experimentation with different algorithms and combinations of machine learning techniques are recommended to improve anomaly detection and provide more reliable and accurate security systems.

RESOURCES

[1] Video analytics and artificial intelligence, [Online]. Available: https://www.axis.com/files/whitepaper/wp_AI_in_video_analytics_ru_2103.pdf

[2] Abdelhafid Berroukham, Khalid Housni, Mohammed Lahraichi, Idir Boulfrifi, Deep learning-based methods for anomaly detection in video surveillance: a review, Feb. 2023, [Online]. Available: https://www.researchgate.net/publication/365243648_Deep_learning-based_methods_for_anomaly_detection_in_video_surveillance_a_review

[3] Waqas Sultani, Chen Chen, Mubarak Shah, Real-world Anomaly Detection in Surveillance Videos, Feb 2019, [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2018/papers/Sultani_Real-World_Anomaly_Detection_CVPR_2018_paper.pdf

[4] Amnah Aldayri, Waleed Albattah, Taxonomy of Anomaly Detection Techniques in Crowd Scenes, Aug. 2022, [Online]. Available: https://www.mdpi.com/1424-8220/22/16/6080

[5] Redhwan Al-amri, Raja Kumar Murugesan, Mustafa Man, Alaa Fareed Abdulateef, Mohammed A. Al-Sharafi, Ammar Ahmed Alkahtani, A Review of Machine Learning and Deep Learning Techniques for Anomaly Detection in IoT Data, Jun. 2021, [Online]. Available: https://www.semanticscholar.org/paper/A-Review-of-Machine-Learning-and-Deep-Learning-for-Al-amri-Murugesan/4109b1a24b19436eb7a1fdd7f6f60b5692a186fb#related-papers

[6] Kallepalli Rohit Kumar, Nisarg Gandhewar, Anomaly Detection in Surveillance System Using Machine Learning Techniques – A Review, 2022, [Online]. Available: https://www.semanticscholar.org/paper/Anomaly-Detection-In-Surveillance-System-Using-A-Kumar-Gandhewar/2673e050034c07bea00c4513ab93f7a1ec3a0a87

[7] Karan Thakkar, Kuldeep Kadiya, Mr. Jigar Chauhan, Anomaly Detection in Surveillance Video, 2021, [Online]. Available: https://www.semanticscholar.org/paper/ANOMALY-DETECTION-IN-SURVEILLANCE-VIDEO-Thakkar-Kadiya/32248d2ab0430dd21646d00a6cd17549cd985e52

[8] M. Andersson, Fredrik Hemström, S. Molin, Robust anomaly detection in urban environments using sensor and information fusion and a camera network, 2018, [Online]. Available: https://www.semanticscholar.org/paper/Robust-anomaly-detection-in-urban-environments-and-Andersson-Hemstr%C3%B6m/aa653854fc118856c63550d5a19f9ef30d31eac4

[9] Pratibha Kumari, Anterpreet Kaur Bedi and Mukesh Saini, Multimedia Datasets for Anomaly Detection: A Review, Apr. 2022, [Online]. Available: https://arxiv.org/pdf/2112.05410.pdf

[10] Jianfeng Wen, Shixian Li, Zhiyong Lin, Yong Hu, Changqin Huang, Systematic literature review of machine learning based software development effort estimation models, 2012, [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584911001832

[11] Daisuke Miki, Shi Chen, Kazuyuki Demachi, Unnatural Human Motion Detection using Weakly Supervised Deep Neural Network, Nov 2020, [Online]. Available: https://ieeexplore.ieee.org/document/9253025

[12] Zainab K Abbas, Ayad Al-Ani, A Comprehensive Review for Video Anomaly Detection on videos, Dec. 2022, [Online]. Available: https://www.researchgate.net/publication/366538291_A_Comprehensive_Review_for_Video_Anomaly_Detection_on_videos

[13] B Ravi Kiran, Dilip Mathew Thomas, Ranjith Parakkal, An overview of deep learning based methods for unsupervised and semi-supervised anomaly detection in videos, Jan 2018, [Online]. Available: https://arxiv.org/abs/1801.03149

[14] Chalapathy, R., & Chawla, S. (2019). Deep learning for anomaly detection: A survey, Jan 2018, [Online]. Available: https://arxiv.org/abs/1901.03407

[15] Akmalazakia Fatan Derana Marsiano; Indah Soesanti; Igi Ardiyanto, Deep learning-based Anomaly Detection on Surveillance Videos: Recent Advances, Sep. 2019, [Online]. Available: https://ieeexplore.ieee.org/document/8904395

[16] J. Crowley, P. Reignier, S. Pesnel, CAVIAR Context Aware Vision using Image-based Active Recognition, 2005, [Online]. Available: https://www.semanticscholar.org/paper/CAVIAR-Context-Aware-Vision-using-Image-based-Crowley-Reignier/4af517b9ebb3e159e59d8608d83369494cca69f52

[17] Huamin Ren; Thomas B. Moeslund (2015). Abnormal event detection using local sparse representation, Aug. 2014, [Online]. Available: https://ieeexplore.ieee.org/document/6918655

[18] Ravanbakhsh, M., Nabi, M., Mousavi, H., & Murino, V.,Abnormal event detection in videos using generative adversarial nets, Aug, 2017, [Online]. Available: https://arxiv.org/abs/1708.09644

[19] Sabokrou, M., Fayyaz, M., Fathy, M., & Klette, R., Deep-anomaly: Fully convolutional neural network for fast anomaly detection in crowded scenes, Apr. 2017, [Online]. Available: https://arxiv.org/abs/1609.00866

[20] Carreira, J., & Zisserman, A., Quo vadis, action recognition? A new model and the kinetics dataset, May. 2017, [Online]. Available: https://arxiv.org/abs/1705.07750

[21] Liu, W., Luo, W., Lian, D., & Gao, S., Future frame prediction for anomaly detection—A new baseline, Mar. 2019, [Online]. Available: https://arxiv.org/abs/1903.03295

[22] Morais, R., Le, V., Tran, T., Saha, B., Mansour, M., & Venkatesh, S., Learning regularity in skeleton trajectories for anomaly detection in videos, Apr. 2019, [Online]. Available: https://arxiv.org/abs/1903.03295

[23] Kaiyang Zhou, Yu Qiao, Tao Xiang, Deep reinforcement learning for unsupervised video summarization with diversity-representativeness reward, Dec.2017, [Online]. Available: https://arxiv.org/abs/1801.00054

[24] Niklas Donges, A Guide to Recurrent Neural Networks: Understanding RNN and LSTM Networks, [Online]. Available: https://builtin.com/data-science/recurrent-neural-networks-and-lstm

[25] Support Vector Machine Algorithm, [Online]. Available: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm

APPENDIX A: ALGORITHMS

Convolutional Neural Networks

```
import os
import cv2
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, roc_curve
from scipy.optimize import brentq
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

dataset_dir = "/Users/anduser/Documents/UCF-crime/"

anomalous_folders = [
    "Abuse",
    "Arrest",
    "Arson",
    "Assault",
    "Burglary",
    "Explosion",
    "Fighting",
    "RoadAccidents",
    "Robbery",
    "Shooting",
    "Shoplifting",
    "Stealing",
    "Vandalism",
]

normal_folders = [
    "Testing_Normal_Videos_Anomaly",
    "Training-Normal-Videos",
    "Normal_Videos_for_Event_Recognition",
]

# Define preprocessing and feature extraction functions
def preprocess_frame(frame, resize_shape=(64, 64), grayscale=True):
    if grayscale:
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    frame = cv2.resize(frame, resize_shape)
    return frame

def load_dataset(dataset_dir, anomalous_folders, normal_folders,
frames_to_sample=5, resize_shape=(64, 64)):
    X = []
    y_true = []

    for folder in normal_folders:
        folder_path = os.path.join(dataset_dir, folder)
        for file in os.listdir(folder_path):
            if file.endswith(".mp4"):
                video_path = os.path.join(folder_path, file)
                X, y_true = process_video(video_path, X, y_true,
frames_to_sample, resize_shape, label=1)

    for folder in anomalous_folders:
        folder_path = os.path.join(dataset_dir, folder)
        for file in os.listdir(folder_path):
            if file.endswith(".mp4"):
                video_path = os.path.join(folder_path, file)
                X, y_true = process_video(video_path, X, y_true,
frames_to_sample, resize_shape, label=0)

    return np.array(X), np.array(y_true)

def process_video(video_path, X, y_true, frames_to_sample, resize_shape,
label):
    cap = cv2.VideoCapture(video_path)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    frame_indices = np.linspace(0, total_frames - 1, frames_to_sample,
dtype=int)

    video_frames = []
    for idx in frame_indices:
        cap.set(cv2.CAP_PROP_POS_FRAMES, idx)
        ret, frame = cap.read()
```

```
            if not ret:
                continue

            frame = preprocess_frame(frame, resize_shape)
            video_frames.append(frame)

        if len(video_frames) == frames_to_sample:
            X.append(video_frames)
            y_true.append(label)

        cap.release()

        return X, y_true

# Load the dataset and split it into training and testing sets
X, y_true = load_dataset(dataset_dir, anomalous_folders, normal_folders)
X_train, X_test, y_train, y_test = train_test_split(X, y_true, test_size=0.2,
random_state=42)

# Preprocess the dataset for the CNN model
X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255

# Define the CNN model
from tensorflow.keras.layers import TimeD istributed, GlobalMaxPooling2D,
LSTM

def create_cnn_model(input_shape):
    model = Sequential()
    model.add(TimeDistributed(Conv2D(32, (3, 3), activation='relu'),
input_shape=input_shape))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Conv2D(64, (3, 3), activation='relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Conv2D(128, (3, 3), activation='relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(GlobalMaxPooling2D()))
    model.add(LSTM(128, activation='relu', return_sequences=False))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))

    return model

# Compile and train the CNN model
```

```
input_shape = (5, 64, 64, 1)
model = create_cnn_model(input_shape)

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

def train_model(X_train, y_train, input_shape):
    model = create_cnn_model(input_shape)
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    X_train_reshaped = np.expand_dims(X_train, axis=-1)
    history = model.fit(X_train_reshaped, y_train, epochs=20, batch_size=32,
validation_split=0.2)
    return model, history

model, history = train_model(X_train, y_train, input_shape)

# Test the model
X_test_reshaped = np.expand_dims(X_test, axis=-1)
y_pred = model.predict(X_test_reshaped)

# Compute AUC
auc = roc_auc_score(y_test, y_pred)

# Compute EER
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
eer = brentq(lambda x: 1. - x - interp1d(fpr, tpr)(x), 0., 1.)

print(f"AUC: {auc}")
print(f"EER: {eer}")

# Visualize the ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred)
plt.plot(fpr, tpr, label=f"ROC curve (AUC = {auc:.2f}, EER = {eer:.2f})")
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.title("ROC curve")
plt.show()
```

Long Short-Term Memory

```
import os
import cv2
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, TimeDistributed,
GlobalMaxPooling1D
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, roc_curve
from scipy.optimize import brentq
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

# Set the dataset directory
dataset_dir = "/Users/anduser/Documents/UCF-crime/"

anomaly_categories = [
    "Abuse",
    "Arrest",
    "Arson",
    "Assault",
    "Burglary",
    "Explosion",
    "Fighting",
    "RoadAccidents",
    "Robbery",
    "Shooting",
    "Shoplifting",
    "Stealing",
    "Vandalism"
]

normal_categories = [
    "Testing_Normal_Videos_Anomaly",
    "Training-Normal-Videos",
    "Normal_Videos_for_Event_Recognition"
]

def preprocess_frame(frame, resize_shape=(64, 64), grayscale=True):
    if grayscale:
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    frame = cv2.resize(frame, resize_shape)
    return frame

def load_dataset(dataset_dir, frames_to_sample=5, resize_shape=(64, 64)):
    X = []
    y_true = []

    for category in anomaly_categories:
        category_dir = os.path.join(dataset_dir, category)
        for file in os.listdir(category_dir):
            if file.endswith(".mp4"):
                video_path = os.path.join(category_dir, file)
                cap = cv2.VideoCapture(video_path)
                total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
                frame_indices = np.linspace(0, total_frames - 1,
frames_to_sample, dtype=int)

                video_frames = []
                for idx in frame_indices:
                    cap.set(cv2.CAP_PROP_POS_FRAMES, idx)
                    ret, frame = cap.read()
                    if not ret:
                        continue

                    frame = preprocess_frame(frame, resize_shape)
                    video_frames.append(frame)

                if len(video_frames) == frames_to_sample:
                    X.append(video_frames)
                    y_true.append(anomaly_categories.index(category) + 1)

                cap.release()

    for category in normal_categories:
        category_dir = os.path.join(dataset_dir, category)
        for file in os.listdir(category_dir):
            if file.endswith(".mp4"):
                video_path = os.path.join(category_dir, file)
                cap = cv2.VideoCapture(video_path)
                total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
                frame_indices = np.linspace(0, total_frames - 1,
frames_to_sample, dtype=int)
```

```
                video_frames = []
                for idx in frame_indices:
                    cap.set(cv2.CAP_PROP_POS_FRAMES, idx)
                    ret, frame = cap.read()
                    if not ret:
                        continue

                    frame = preprocess_frame(frame, resize_shape)
                    video_frames.append(frame)
                if len(video_frames) == frames_to_sample:
                    X.append(video_frames)
                    y_true.append(0)  # 0 for normal videos

                cap.release()

    return np.array(X), np.array(y_true)

# Split the dataset into training and testing sets
X, y_true = load_dataset(dataset_dir)
X_train, X_test, y_train, y_test = train_test_split(X, y_true, test_size=0.2,
random_state=42)
X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255

# Define the LSTM model
def create_lstm_model(input_shape):
    model = Sequential()
    model.add(TimeDistributed(GlobalMaxPooling1D(), input_shape=input_shape))
    model.add(LSTM(128, activation='relu', return_sequences=False))
    model.add(Dropout(0.5))
    model.add(Dense(len(anomaly_categories) + 1, activation='softmax'))

    return model

input_shape = (5, 64, 64)
model = create_lstm_model(input_shape)

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Compile the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32,
validation_split=0.2)
```

```
# Test the model
y_pred = model.predict(X_test)

# Convert predictions to class labels
y_pred_class = np.argmax(y_pred, axis=1)

# Plot confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns

cm = confusion_matrix(y_test, y_pred_class)
plt.figure(figsize=(10, 10))
sns.heatmap(cm, annot=True, fmt="d", cmap="YlGnBu", xticklabels=["Normal"] +
anomaly_categories,
            yticklabels=["Normal"] + anomaly_categories)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()

# Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred_class)
print(f"Accuracy: {accuracy}")

from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_auc_score, roc_curve, auc

def compute_eer(y_true, y_score):
    fpr, tpr, thresholds = roc_curve(y_true, y_score)
    fnr = 1 - tpr
    eer_threshold = thresholds[np.nanargmin(np.absolute((fnr - fpr)))]
    eer = fpr[np.nanargmin(np.absolute((fnr - fpr)))]
    return eer, eer_threshold

# Binarize the labels for ROC-AUC calculation
y_test_bin = label_binarize(y_test,
classes=list(range(len(anomaly_categories) + 1)))
y_pred_bin = y_pred

# Compute ROC-AUC for each class
auc_scores = []
eer_scores = []
```

```
for i in range(len(anomaly_categories) + 1):
    class_auc = roc_auc_score(y_test_bin[:, i], y_pred_bin[:, i])
    class_eer, _ = compute_eer(y_test_bin[:, i], y_pred_bin[:, i])
    auc_scores.append(class_auc)
    eer_scores.append(class_eer)

# Calculate the average ROC-AUC and EER scores
mean_auc = np.mean(auc_scores)
mean_eer = np.mean(eer_scores)

print(f"Mean ROC-AUC: {mean_auc}")
print(f"Mean EER: {mean_eer}")

# Import necessary libraries
from itertools import cycle
from sklearn.metrics import roc_curve, auc

# Function to plot ROC curve for each class
def plot_roc_curve(y_test_bin, y_pred_bin, n_classes):
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_bin[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'green', 'red',
'purple', ', 'yellow', 'pink', 'brown', 'cyan', 'magenta', 'lime'])
for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color,
                 label='ROC curve of class {0} (AUC = {1:0.2f})'.format(i,
roc_auc[i]))

    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve for Each Class')
    plt.legend(loc="lower right")
    plt.show()

# Plot ROC curve
n_classes = len(anomaly_categories) + 1
```

```
plot_roc_curve(y_test_bin, y_pred_bin, n_classes)

# Plot EER scores
plt.figure()
plt.bar(["Normal"] + anomaly_categories, eer_scores)
plt.xlabel("Categories")
plt.ylabel("Equal Error Rate (EER)")
plt.title("EER for Each Category")
plt.show()
```

Support Vector Machine

```
import os
import cv2
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import roc_auc_score, roc_curve, classification_report,
confusion_matrix
import matplotlib.pyplot as plt
from scipy.optimize import brentq
from scipy.interpolate import interp1d

dataset_dir = "/Users/anduser/Documents/UCF-crime/"

anomalous_folders = [
    "Abuse",
    "Arrest",
    "Arson",
    "Assault",
    "Burglary",
    "Explosion",
    "Fighting",
    "RoadAccidents",
    "Robbery",
    "Shooting",
    "Shoplifting",
    "Stealing",
    "Vandalism",
]
```

```
normal_folders = [
    "Testing_Normal_Videos_Anomaly",
    "Training-Normal-Videos",
    "Normal_Videos_for_Event_Recognition",
]

# Define preprocessing and feature extraction functions
def preprocess_frame(frame, resize_shape=(64, 64), grayscale=True):
    if grayscale:
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    frame = cv2.resize(frame, resize_shape)
    return frame

def load_dataset(dataset_dir, anomalous_folders, normal_folders,
frames_to_sample=5, resize_shape=(64, 64)):
    X = []
    y_true = []

    for folder in normal_folders:
        folder_path = os.path.join(dataset_dir, folder)
        for file in os.listdir(folder_path):
            if file.endswith(".mp4"):
                video_path = os.path.join(folder_path, file)
                X, y_true = process_video(video_path, X, y_true,
frames_to_sample, resize_shape, label=1)

    for folder in anomalous_folders:
        folder_path = os.path.join(dataset_dir, folder)
        for file in os.listdir(folder_path):
            if file.endswith(".mp4"):
                video_path = os.path.join(folder_path, file)
                X, y_true = process_video(video_path, X, y_true,
frames_to_sample, resize_shape, label=0)

    return np.array(X), np.array(y_true)

def process_video(video_path, X, y_true, frames_to_sample, resize_shape,
label):
    cap = cv2.VideoCapture(video_path)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    frame_indices = np.linspace(0, total_frames - 1, frames_to_sample,
dtype=int)

    video_frames = []
    for idx in frame_indices:
        cap.set(cv2.CAP_PROP_POS_FRAMES, idx)
        ret, frame = cap.read()
        if not ret:
            continue

        frame = preprocess_frame(frame, resize_shape)
        video_frames.append(frame)

    if len(video_frames) == frames_to_sample:
        X.append(video_frames)
        y_true.append(label)

    cap.release()

    return X, y_true

# Load the dataset and split it into training and testing sets
X, y_true = load_dataset(dataset_dir, anomalous_folders, normal_folders)
X_train, X_test, y_train, y_test = train_test_split(X, y_true, test_size=0.2,
random_state=42)

# Preprocess the dataset for the SVM model
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create the SVM model
svm_model = SVC(kernel='linear', probability=True, C=1, random_state=42)

# Train the SVM model
svm_model.fit(X_train_scaled, y_train)

# Predict using the SVM model
y_pred_svm = svm_model.predict(X_test_scaled)
y_pred_svm_proba = svm_model.predict_proba(X_test_scaled)[:, 1]

# Compute AUC
auc_svm = roc_auc_score(y_test, y_pred_svm_proba)

# Compute EER
```

```
fpr_svm, tpr_svm, thresholds_svm = roc_curve(y_test, y_pred_svm_proba)
eer_svm = brentq(lambda x: 1. - x - interp1d(fpr_svm, tpr_svm)(x), 0., 1.)

print("SVM Model Evaluation")
print(f"AUC: {auc_svm}")
print(f"EER: {eer_svm}")
print("Classification Report:")
print(classification_report(y_test, y_pred_svm))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_svm))

# Visualize the ROC curve
plt.plot(fpr_svm, tpr_svm, label=f"ROC curve (AUC = {auc_svm:.2f}, EER =
{eer_svm:.2f})")
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.title("ROC curve")
plt.show()
```

Recurrent Neural Network

```
import os
import cv2
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, TimeDistributed
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, roc_curve
from scipy.optimize import brentq
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

dataset_dir = "/Users/anduser/Documents/UCF-crime/"
anomalous_folders = [
    "Abuse",
    "Arrest",
    "Arson",
    "Assault",
    "Burglary",
    "Explosion",
    "Fighting",
    "RoadAccidents",
    "Robbery",
    "Shooting",
    "Shoplifting",
    "Stealing",
    "Vandalism",
]

normal_folders = [
    "Testing_Normal_Videos_Anomaly",
    "Training-Normal-Videos",
    "Normal_Videos_for_Event_Recognition",
]

# Define preprocessing and feature extraction functions
def preprocess_frame(frame, resize_shape=(64, 64), grayscale=True):
    if grayscale:
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    frame = cv2.resize(frame, resize_shape)
    return frame

def load_dataset(dataset_dir, anomalous_folders, normal_folders,
frames_to_sample=5, resize_shape=(64, 64)):
    X = []
    y_true = []

    for folder in normal_folders:
        folder_path = os.path.join(dataset_dir, folder)
        for file in os.listdir(folder_path):
            if file.endswith(".mp4"):
                video_path = os.path.join(folder_path, file)
                X, y_true = process_video(video_path, X, y_true,
frames_to_sample, resize_shape, label=1)

    for folder in anomalous_folders:
        folder_path = os.path.join(dataset_dir, folder)
        for file in os.listdir(folder_path):
            if file.endswith(".mp4"):
                video_path = os.path.join(folder_path, file)
                X, y_true = process_video(video_path, X, y_true,
frames_to_sample, resize_shape, label=0)

    return np.array(X), np.array(y_true)

def process_video(video_path, X, y_true, frames_to_sample, resize_shape,
label):
    cap = cv2.VideoCapture(video_path)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    frame_indices = np.linspace(0, total_frames - 1, frames_to_sample,
dtype=int)

    video_frames = []
    for idx in frame_indices:
        cap.set(cv2.CAP_PROP_POS_FRAMES, idx)
        ret, frame = cap.read()
        if not ret:
            continue

        frame = preprocess_frame(frame, resize_shape)
        video_frames.append(frame)

    if len(video_frames) == frames_to_sample:
        X.append(video_frames)
        y_true.append(label)

    cap.release()

    return X, y_true

# Load the dataset and split it into training and testing sets
X, y_true = load_dataset(dataset_dir, anomalous_folders, normal_folders)
X_train, X_test, y_train, y_test = train_test_split(X, y_true, test_size=0.2,
random_state=42)

# Preprocess the dataset for the RNN model
X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255

# Define the RNN model
def create_rnn_model(input_shape):
    model = Sequential()
    model.add(LSTM(64, return_sequences=True, activation='relu',
input_shape=input_shape))
```

```
    model.add(LSTM(128, return_sequences=True, activation='relu'))
    model.add(LSTM(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))
    return model

def train_rnn_model(X_train, y_train, input_shape):
    model = create_rnn_model(input_shape)
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    X_train_reshaped = X_train.reshape(X_train.shape[0], 5, -1)
    history = model.fit(X_train_reshaped, y_train, epochs=20, batch_size=32,
validation_split=0.2)
    return model, history

# Compile and train the RNN model
input_shape = (5, 64 * 64)
rnn_model, rnn_history = train_rnn_model(X_train, y_train, input_shape)

# Test the model
X_test_reshaped = X_test.reshape(X_test.shape[0], 5, -1)
y_pred_rnn = rnn_model.predict(X_test_reshaped)

# Compute AUC
auc_rnn = roc_auc_score(y_test, y_pred_rnn)

# Compute EER
fpr_rnn, tpr_rnn, thresholds_rnn = roc_curve(y_test, y_pred_rnn)
eer_rnn = brentq(lambda x: 1. - x - interp1d(fpr_rnn, tpr_rnn)(x), 0., 1.)

print(f"AUC (RNN): {auc_rnn}")
print(f"EER (RNN): {eer_rnn}")

# Visualize the ROC curve.
fpr_rnn, tpr_rnn, _ = roc_curve(y_test, y_pred_rnn)
plt.plot(fpr_rnn, tpr_rnn, label=f"ROC curve (AUC = {auc_rnn:.2f}, EER =
{eer_rnn:.2f})")
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.title("ROC curve (RNN)")
plt.show()
```