

# **SPATIO-TEMPORAL AUTOENCODER BASED VIDEO ANOMALY DETECTION**

A PROJECT REPORT SUBMITTED IN PARTIAL FULFILMENT OF THE  
REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

**BACHELOR OF TECHNOLOGY  
IN  
COMPUTER SCIENCE AND ENGINEERING**



By  
**Batch – A6**

**A. Pallavi** (20JG1A0502)  
**J. Varsha Priya** (20JG1A0529)

**K. Sneha Latha** (20JG1A0545)  
**M.V.S. Dhruthi** (20JG1A0563)

Under the esteemed guidance of

**Mrs. D. Indu**  
Assistant Professor  
CSE Department

**Department of Computer Science and Engineering**

**GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN**

[Approved by AICTE NEW DELHI, Affiliated to JNTUK Kakinada]

[Accredited by National Board of Accreditation (NBA) for B. Tech. CSE, ECE & IT – valid from 2019-22 and 2022-25] [Accredited by National Assessment and Accreditation Council (NAAC) with A Grade-Valid from 2022-2027]

Kommadi, Madhurawada, Visakhapatnam – 530048

**2023 – 2024**

**GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



**CERTIFICATE**

This is to certify that the Project report titled “**Spatio-Temporal Autoencoder Based Video Anomaly Detection**” is a bona fide work of following IV B. Tech students in the Department of Computer Science and Engineering, Gayatri Vidya Parishad College of Engineering for Women affiliated to JNT University, Kakinada during the academic year 2023-24, in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology of this university.

**A. Pallavi** (20JG1A0502)

**J. Varsha Priya** (20JG1A0529)

**K. Sneha Latha** (20JG1A0545)

**M.V.S. Dhruthi** (20JG1A0563)

Signature of the Guide

**Mrs. D. Indu**

Asst. Professor

Dept. of CSE

Signature of the HOD

**Dr. P. V. S. L. Jagadamba**

Professor

Dept. of CSE

**External Examiner**

## ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people who made it possible and whose constant guidance and encouragement crown all the efforts with success.

We feel elated to extend our sincere gratitude to our guide, **Mrs. D. Indu**, Assistant Professor for encouragement all the way during analysis of the project. Her annotations, insinuations and criticisms are the key behind the successful completion of the thesis and for providing us all the required facilities.

We would like to take this opportunity to extend our gratitude to Project Coordinator, **Mrs. K. Suneetha**, Assistant Professor of Computer Science and Engineering for making us to follow a defined path and for advising us to get better improvements in the project.

We express our deep sense of gratitude and thanks to **Dr. P. V. S. Lakshmi Jagadamba**, Professor and Head of the Department of Computer Science and Engineering for her guidance and for expressing her valuable and grateful opinions in the project for its development and for providing lab sessions and extra hours to complete the project.

We would like to take this opportunity to express our profound sense of gratitude to **Dr. R. K. Goswami**, Principal and **Dr. G. Sudheer**, Vice Principal for allowing us to utilize the college resources thereby facilitating the successful completion of our thesis.

We are also thankful to both teaching and non-teaching faculty of the Department of Computer Science and Engineering for giving valuable suggestions for our project.

**A. Pallavi** (20JG1A0502)

**K. Sneha Latha** (20JG1A0545)

**J. Varsha Priya** (20JG1A0529)

**M.V.S. Dhruthi** (20JG1A0563)

# TABLE OF CONTENTS

| TOPICS                                 | PAGE NO.  |
|--|-----------|
| <b>ABSTRACT</b>                        | <b>i</b>  |
| <b>LIST OF FIGURES</b>                 | <b>ii</b> |
| <b>LIST OF TABLES</b>                  | <b>iv</b> |
| <b>LIST OF SCREENS</b>                 | <b>v</b>  |
| <b>LIST OF ACRONYMS</b>                | <b>vi</b> |
| <br>                                   |           |
| <b>1. INTRODUCTION</b>                 | <b>1</b>  |
| 1.1 Motivation                         | 3         |
| 1.2 Problem Definition                 | 3         |
| 1.3 Objective of the Project           | 4         |
| 1.4 Limitations of the Project         | 4         |
| 1.5 Organization of the Project        | 5         |
| <br>                                   |           |
| <b>2. LITERATURE SURVEY</b>            | <b>6</b>  |
| 2.1 Existing System                    | 10        |
| 2.2 Proposed System                    | 12        |
| 2.3 Novelty of our Project             | 15        |
| 2.4 Conclusion                         | 15        |
| <br>                                   |           |
| <b>3. REQUIREMENT ANALYSIS</b>         | <b>21</b> |
| 3.1 Introduction                       | 21        |
| 3.2 Software Requirement Specification | 21        |
| 3.2.1 Software Requirements            | 21        |
| 3.2.2 Hardware Requirements            | 22        |
| 3.3 Non-Functional Requirements        | 23        |
| 3.4 Content Diagram of our Project     | 24        |
| 3.5 Flowchart of our Project           | 25        |
| 3.6 Algorithms                         | 27        |

| <b>TOPICS</b>                                 | <b>PAGE NO.</b> |
|---|-----------------|
| <b>4. DESIGN</b>                              | <b>35</b>       |
| 4.1 Introduction                              | 35              |
| 4.2 UML diagrams                              | 36              |
| 4.2.1 Use case diagram                        | 36              |
| 4.2.2 Class diagram                           | 37              |
| 4.2.3 Sequence diagram                        | 39              |
| 4.3 Module Design                             | 42              |
| 4.3.1 System Module                           | 42              |
| 4.4 Conclusion                                | 44              |
| <b>5. IMPLEMENTATION AND RESULTS</b>          | <b>45</b>       |
| 5.1 Introduction                              | 45              |
| 5.2 Explanation of Key functions              | 45              |
| 5.3 Model building                            | 46              |
| 5.3.1 Hybrid Model of CNN-LSTM                | 46              |
| 5.4 Integration of Frontend with Backend code | 48              |
| 5.5 Results                                   | 74              |
| 5.5.1 Output Screens                          | 74              |
| 5.5.2 Graphs                                  | 80              |
| 5.6 Conclusion                                | 85              |
| <b>6. TESTING AND VALIDATION</b>              | <b>86</b>       |
| 6.1 Introduction                              | 86              |
| 6.1.1 Scope                                   | 86              |
| 6.1.2. Defects and Failures                   | 86              |
| 6.1.3 Compatibility                           | 87              |
| 6.1.4 Input Combinations and Preconditions    | 87              |
| 6.1.5 Static Vs Dynamic Testing               | 88              |
| 6.1.6 Software Verification and Validation    | 88              |
| 6.2 Design of Test Cases and Scenario         | 88              |
| 6.2.1 Validation Testing                      | 88              |
| 6.2.2 Unit Testing                            | 89              |
| 6.2.3 Black Box Testing                       | 89              |
| 6.2.4 White Box Testing                       | 89              |

| <b>TOPICS</b>                  | <b>PAGE NO.</b> |
|--------------------------------|-----------------|
| 6.2.5 Integration Testing      | 90              |
| 6.2.6 System Testing           | 90              |
| 6.3 Validation Testing Screens | 90              |
| 6.4 Conclusion                 | 90              |
| <b>7. CONCLUSION</b>           | <b>91</b>       |
| <b>8. REFERENCES</b>           | <b>92</b>       |

# **ABSTRACT**

The widespread adoption of video surveillance systems across diverse sectors has generated vast amounts of video data, necessitating effective analysis methods for enhanced safety and efficiency. Anomaly detection in videos is pivotal for identifying threats, accidents, or equipment malfunctions, yet traditional techniques relying on handcrafted features face limitations in capturing intricate spatial-temporal patterns and scalability to large datasets. In response, deep learning-based approaches, particularly spatio-temporal autoencoders, have emerged as promising solutions for unsupervised learning of complex features directly from video data.

In this project, the efficacy of spatio-temporal autoencoders for video anomaly detection, focusing on the decomposition and analysis of video cuboids to capture both spatial and temporal characteristics. By harnessing the feature learning capabilities of deep learning and the granularity offered by cuboid-based representations, the proposed method aims to achieve more accurate, and computationally efficient anomaly detection. The report delves into the underlying principles and techniques of spatio-temporal autoencoders, detailing their application in decomposing video sequences and extracting meaningful features. Evaluation of the method's performance is conducted across various real-world video datasets, providing insights into its effectiveness and potential applications. Additionally, the report discusses challenges and outlines future directions for advancing video anomaly detection using spatio-temporal autoencoders.

## LIST OF FIGURES

| S. No. | Figure No. | Figure Name   | Page No. |
|--------|------------|---|----------|
| 1.     | Fig. 1     | Existing System: Single-Scene Video Anomaly Detection                               | 11       |
| 2.     | Fig. 2     | Existing System   | 12       |
| 3.     | Fig. 3     | Cascade for anomaly detection   | 14       |
| 4.     | Fig. 4     | Architecture Diagram  | 25       |
| 5.     | Fig. 5     | Flow diagram of the proposed method   | 26       |
| 6.     | Fig. 6     | Working of autoencoder and decoder  | 28       |
| 7.     | Fig. 7     | Code snippet of Autoencoder Model   | 31       |
| 8.     | Fig. 8     | LSTM Architecture   | 32       |
| 9.     | Fig. 9     | Hybrid Model of CNN and LSTM  | 33       |
| 10.    | Fig. 10    | Relationships in UML Diagrams   | 35       |
| 11.    | Fig. 11    | Use case diagram  | 37       |
| 12.    | Fig. 12    | Class diagram   | 39       |
| 13.    | Fig. 13    | Sequence diagram  | 41       |
| 14.    | Fig. 14    | Autoencoder summary   | 46       |
| 15.    | Fig. 15    | Hierarchy of code files in our project  | 50       |
| 16.    | Fig. 16    | Reconstruction cost vs frame  | 80       |
| 17.    | Fig. 17    | Finding threshold   | 81       |
| 18.    | Fig. 18    | Finding anomaly plot  | 82       |
| 19.    | Fig. 19    | Plots of results in various stages  | 83       |
| 20.    | Fig. 20    | Graph plotted between training accuracy vs validation accuracy                      | 83       |
| 21.    | Fig. 21    | Graph plotted between training loss vs validation loss                              | 84       |
| 22.    | Fig. 22    | Examples of anomaly detection shown for a frame vs localized cuboid method Word2vec | 84       |



## **LIST OF TABLES**

| <b>S. No.</b> | <b>Table No.</b> | <b>Table Name</b>             | <b>Page No.</b> |
|---------------|------------------|-------------------------------|-----------------|
| 1.            | Table 1          | Literature Survey             | 16              |
| 2.            | Table 2          | Software requirements         | 21              |
| 3.            | Table 3          | Python packages               | 22              |
| 4.            | Table 4          | Front end packages            | 23              |
| 5.            | Table 5          | Minimum Hardware requirements | 23              |
| 6.            | Table 6          | Class Diagram Artifacts       | 38              |
| 7.            | Table 7          | Black box testing             | 88              |

## LIST OF SCREENS

| <b>S. No.</b> | <b>Screen No.</b> | <b>Screen Name</b>           | <b>Page No.</b> |
|---------------|-------------------|------------------------------|-----------------|
| 1.            | Screen 1          | Home page                    | 74              |
| 2.            | Screen 2          | Project information page     | 74              |
| 3.            | Screen 3          | Signup page                  | 75              |
| 4.            | Screen 4          | Validation of Form           | 75              |
| 5.            | Screen 5          | Validation of Email field    | 76              |
| 6.            | Screen 6          | Validation of Password field | 76              |
| 7.            | Screen 7          | Validation of Age field      | 76              |
| 8.            | Screen 8          | Login page                   | 76              |
| 9.            | Screen 9          | Invalid login details        | 77              |
| 10.           | Screen 10         | Before login                 | 77              |
| 11.           | Screen 11         | After login                  | 77              |
| 12.           | Screen 12         | Uploading page               | 78              |
| 13.           | Screen 13         | After uploading video        | 79              |
| 14.           | Screen 14         | Detection page               | 79              |

## **LIST OF ACRONYMS**

|      |   |                                   |
|------|---|-----------------------------------|
| CNN  | - | Convolutional Neural Network      |
| CPU  | - | Central Processing Unit           |
| CUHK | - | Chinese University of Hong Kong   |
| FPS  | - | Frames Per Second                 |
| GAN  | - | Generative Adversarial Network    |
| GPU  | - | Graphics Processing Unit          |
| LSTM | - | Long Short-Term Memory            |
| MSE  | - | Mean Squared Error                |
| PCA  | - | Principal Component Analysis      |
| RGB  | - | Red, Green, Blue                  |
| ROI  | - | Region of Interest                |
| STAE | - | Spatio-Temporal Autoencoder       |
| SVD  | - | Singular Value Decomposition      |
| UCSD | - | University of California, San Die |

# 1. INTRODUCTION

The increasing adoption of video surveillance systems across various sectors, including public safety, traffic management, and industrial production, has led to an exponential growth in video data. Effective analysis of this data can yield valuable insights, leading to improved safety and efficiency in these domains. Anomaly detection is a critical aspect of video analysis that has attracted considerable attention among researchers and practitioners. Identifying anomalous events in videos is essential for recognizing potential threats, accidents, or malfunctioning equipment. However, the complex nature of video data, which encompasses spatial and temporal dimensions, makes this task challenging. Recently, deep learning-based methods [1-8] have demonstrated promising results in video anomaly detection.

In this report, an innovative method for detecting video anomalies using spatiotemporal autoencoders [1-2]. the proposed method involves converting the video into frames using FFmpeg, a widely used multimedia framework. These frames are then transformed into cuboids [11], which are three-dimensional pixel blocks. Each cuboid is saved in its own folder for subsequent processing. A spatio-temporal autoencoder is then trained on each folder of cuboids, learning the spatiotemporal features of the cuboids.

Traditional video anomaly detection methods rely on handcrafted features like optical flow or Histogram of Oriented Gradients (HOG). Although these approaches have shown some success, they may not fully capture the intricate spatial and temporal patterns present in the data. Additionally, extracting handcrafted features is often computationally expensive, limiting the scalability of these methods for large-scale video datasets.

To address these limitations, a novel approach for video anomaly detection based on spatiotemporal autoencoders and a cuboid-based representation. Autoencoders are unsupervised neural network models that learn to encode and decode data in a lower-dimensional space. By training spatiotemporal autoencoders on cuboids, the aim is to capture the spatial and temporal dynamics of the video data, resulting in a more effective and accurate anomaly detection framework.

The proposed method starts with decomposing a video into individual frames using the FFmpeg tool. Each frame is divided into non-overlapping cuboids [11],

which are smaller three-dimensional sections containing spatial and temporal information. These cuboids are stored in separate folders, and a spatio-temporal autoencoder [1-2] is trained for each folder, utilizing the respective cuboids. This modular approach enables the autoencoder to learn the intricate patterns and relationships between the cuboids, providing a comprehensive representation of the video data.

During the testing phase, videos are again decomposed into frames and subsequently transformed into cuboids. These cuboids are then tested against their corresponding trained autoencoder models. The reconstruction loss, a measure of the discrepancy between the input and output cuboids, is calculated for each cuboid in the form of NumPy arrays. If the loss exceeds a pre-defined threshold, the cuboid is considered anomalous.

The proposed work is inspired by Li et al [11], who proposed a Spatio Temporal Cascade Autoencoder that uses a cuboid patch-based method to successively utilize spatial and temporal cues from video data. They have trained this model using three datasets, including UCSD, Avenue, and UMN, to detect and locate various types of abnormal behaviors in real-world videos. The proposed approach builds upon their work by using a threshold-based approach to detect anomalies, which helps in reducing false positives and improving the accuracy of the detection.

Several research papers have employed spatiotemporal autoencoders for video anomaly detection [1-2]. However, the proposed approach differs from these methods in several ways. Firstly, the use cuboids for processing the frames, which helps in capturing the spatio-temporal features of the frames more accurately.

Secondly, train a separate autoencoder model for each folder of cuboids, which aids in detecting anomalies more effectively. Thirdly, employ a threshold-based approach to detect anomalies, which assists in reducing false positives and improving detection accuracy. Lastly, our modular approach allows for better adaptability to different types of videos and scenarios, making it more versatile and applicable in various real-world situations.

To evaluate the performance of the proposed method, to conduct experiments on several widely-used video anomaly detection datasets, such as Avenue, and Anomaly detection. The results demonstrate that the proposed approach outperforms existing state-of-the-art methods in terms of detection accuracy, false positive rate, and processing time. Furthermore, the method shows a high degree of adaptability when

applied to diverse types of videos and scenarios, proving its versatility in real-world applications.

In addition to its immediate applications in surveillance, traffic monitoring, and industrial automation, the proposed method holds promise for numerous other fields where video analysis is critical. For instance, in healthcare, the proposed approach can be used to monitor patient behavior, identifying anomalies that may indicate the need for intervention or changes in treatment. Similarly, in sports, it can be employed to analyze athletes' performance, identifying unusual movements or patterns that may suggest injury or the need for improved technique.

Moreover, proposed approach can be readily extended and adapted to incorporate other types of deep learning techniques or data representations. For example, integrating advanced unsupervised learning algorithms, such as variational autoencoders or generative adversarial networks, can further enhance the performance of the method. Additionally, exploring alternative data representations, such as point clouds or volumetric representations, may help capture even more complex spatial and temporal relationships in video data.

## **1.1 MOTIVATION**

The project aims to develop video anomaly detection using spatio-temporal autoencoders to address the limitations of traditional surveillance systems. These systems often struggle to identify abnormal activities in real-time due to their reliance on predefined rules. With the increasing complexity of threats and urban environments, there's a critical need for more intelligent surveillance solutions. Spatio-temporal autoencoders offer promise by leveraging deep learning to learn complex spatiotemporal patterns in video data. This project seeks to contribute to the development of robust and scalable anomaly detection solutions that can proactively identify security threats in real-time. The outcomes of this research have broader implications, including applications in industrial processes, healthcare monitoring, and environmental surveillance, aiming to foster innovation for safer communities.

## **1.2 PROBLEM DEFINITION**

The problem is that typical video surveillance systems rely too heavily on manual monitoring and overly simplified rules, which makes it difficult for them to spot

anomalies in real-time. The complexity of anomalies, scalability problems, the need for real-time detection, and the requirement for flexibility to changing monitoring environments are some particular concerns. The project's goal is to use spatio-temporal autoencoders to create a scalable and reliable video anomaly detection system that can recognize anomalies in a variety of settings by utilizing deep learning techniques.

### **1.3 OBJECTIVE OF THE PROJECT**

The objective of this project is to design, develop, and implement a video anomaly detection system based on spatio-temporal autoencoders. This entails enhancing surveillance accuracy by accurately identifying anomalies and abnormal events in video streams, ensuring real-time detection to enable timely responses to potential security threats or abnormal activities. Additionally, scalability is crucial, requiring the system to handle large volumes of video data efficiently to accommodate diverse surveillance environments and increasing data volumes. The system should also be adaptable to dynamic changes in surveillance conditions, such as variations in lighting, weather, or scene clutter, to enhance robustness across different scenarios. Integration with existing surveillance infrastructure is essential, facilitating seamless deployment and interoperability with other security systems. The ultimate goal of accomplishing these goals is to progress video anomaly detection methods, improving security and safety in a variety of contexts, such as public areas, transit hubs, key infrastructure, and industrial sites.

### **1.4 LIMITATIONS OF THE PROJECT**

The project faces several limitations inherent to utilizing spatio-temporal autoencoders and cuboidal data for video anomaly detection. First and foremost, there can be significant computational complexity involved in training deep learning models on cuboidal data, which may limit scalability and place demands on resources. Furthermore, it might be difficult to preprocess cuboidal data in order to extract significant spatiotemporal features; this requires complex transformations and feature engineering. Moreover, spatiotemporal autoencoders have restricted interpretability, which makes it more difficult to comprehend the judgements made by the models. Learning representations that are too unique to the training set may also restrict generalization across various surveillance scenarios. Problems with data quality and availability could occur and affect how well the anomaly detection system performs.

Additional considerations include sensitivity to hyperparameters and the possibility of overfitting, which need for meticulous tuning and validation processes. Addressing these limitations demands a nuanced approach, encompassing resource management, data preprocessing techniques, interpretability methods, generalization strategies, data quality assurance measures, hyperparameter tuning, and mitigation strategies for overfitting.

## **1.5 ORGANIZATION OF THE PROJECT**

Concise overview of rest of the documentation work is explained below:

**Chapter 1:** Introduction describes the motivation and objective of this project.

**Chapter 2:** Literature survey describe the primary terms involved in the development of this project.

**Chapter 3:** Analysis deals with the detail analysis of the project. Software Requirement Specification further contains software requirements, hardware requirements.

**Chapter 4:** Contains Methodology. It includes the system overview, modules, architecture diagram and algorithms used.

**Chapter 5:** Contains the implementation and results of the project.

**Chapter 6:** Contains project conclusion.

**Chapter 7:** Contains future enhancement.

**Chapter 8:** Contains references.



## 2. LITERATURE SURVEY

Due to its potential uses in a variety of domains, including security and surveillance, video anomaly detection has been a busy topic of research in recent years. This section examines and discusses the advantages and disadvantages of a few of the currently used techniques for video anomaly identification.

Guo, J., Zheng, P., and Huang, J. [1] The authors propose a deep convolutional autoencoder for anomaly detection. Although they use an autoencoder, their method differs from our spatio-temporal autoencoder approach. Our system focuses on video frames converted into cuboids, which enables better localization of anomalies.

Zhou, J.T., Du, J., Zhu, H., Peng, X., Liu, Y., and Goh, R.S.M. [2] This study presents a combination of CNN and LSTM for video anomaly detection. While our method also utilizes deep learning, we employ a spatio-temporal autoencoder instead of combining CNN and LSTM. This makes it possible to identify abnormalities in particular regions of a frame more quickly and accurately.

Luo, W., Liu, W., Lian, D., Tang, J., Duan, L., Peng, X., and Gao, S. [3] The authors of this paper introduce a novel unsupervised anomaly detection method using a variational autoencoder (VAE). While both our approach and theirs use autoencoders, we use a spatio-temporal autoencoder rather than a VAE, which we believe is better suited for handling the temporal aspect of videos.

Cheng, K.W., Chen, Y.T., and Fang, W.H. [4] This research proposes a method that employs optical flow for motion analysis in video anomaly detection. Although this approach also analyzes video frames, our spatio-temporal autoencoder allows for more robust detection by considering both spatial and temporal information.

Chu, W., Xue, H., Yao, C., and Cai, D. [5] The authors present a multi-modal fusion-based method for anomaly detection. While our method also considers multiple modalities, our spatio-temporal autoencoder is specifically designed for handling video data, making it more effective in detecting anomalies.

Li, Y., Cai, Y., Liu, J., Lang, S., and Zhang, X. [6] This study introduces a hybrid method that combines traditional computer vision techniques with deep learning for anomaly detection. While our approach also utilizes deep learning, our spatio-temporal autoencoder offers a more straightforward and efficient solution for detecting anomalies in videos by directly processing of the cuboids.

Leyva, R., Sanchez, V., and Li, C.T. [7] The authors of this paper propose a method based on the fusion of features extracted from multiple convolutional layers for anomaly detection. Our spatio-temporal autoencoder, in contrast, processes the entire cuboid structure, enabling the more precise location and detection of abnormalities in video frames.

Ganokratanaa, T., Aramvith, S., and Sebe, N. [8] This research presents an approach that employs unsupervised learning with a one-class support vector machine (SVM) for anomaly detection. While their method uses a different learning technique, our spatio-temporal autoencoder offers a more suitable solution for handling the temporal aspect of videos, resulting in improved anomaly detection performance.

Wang, S., Zeng, Y., Liu, Q., Zhu, C., Zhu, E., and Yin, J. [9] The authors propose a graph-based method for video anomaly detection. Although both our approach and theirs analyze video data, our spatio-temporal autoencoder allows for a more robust and efficient detection of anomalies by capturing both spatial and temporal information.

Jardim, E., Thomaz, L.A., da Silva, E.A., and Netto, S.L. [10] This study introduces a method that combines multiple deep learning models for improved video anomaly detection. Our approach, however, focuses on a single spatio-temporal autoencoder model, which simplifies the process and provides effective detection of anomalies in videos.

Li, N., Chang, F., and Liu, C. [11] The authors of this paper present a method based on a three-dimensional (3D) convolutional neural network (CNN) for video anomaly detection. While our method also employs deep learning techniques, our spatio-temporal autoencoder is specifically designed for handling video data in the form of cuboids, providing better localization of anomalies.

Nawaratne, R., Alahakoon, D., De Silva, D., and Yu, X. [12] This research proposes an approach that uses a recurrent neural network (RNN) for temporal modeling in video anomaly detection. Our spatio-temporal autoencoder, on the other hand, directly processes the cuboid structures, which captures both spatial and temporal information for accurate anomaly detection.

Xiang, T., and Gong, S. [13] The authors introduce a method based on a hierarchical attention mechanism for anomaly detection in videos. While their approach incorporates attention mechanisms, our spatio-temporal autoencoder provides a more streamlined solution for detecting and localizing anomalies in video data.

Thomaz, L.A., Jardim, E., da Silva, A.F., da Silva, E.A., Netto, S.L., and Krim, H. [14] This study presents a method that employs a generative adversarial network (GAN) for video anomaly detection. Although GANs can be effective, our spatio-temporal autoencoder offers a more focused approach for handling video data and detecting anomalies.

Li, W., Mahadevan, V., and Vasconcelos, N. [15] The authors of this paper propose an approach that uses a combination of handcrafted features and deep learning for video anomaly detection. While this method also analyzes video data, our spatio-temporal autoencoder enables more robust detection by considering both spatial and temporal information.

Xu, K., Jiang, X., and Sun, T. [16] This research introduces a method based on multi-scale analysis for video anomaly detection. Our spatio-temporal autoencoder, in contrast, processes the entire cuboid structure, allowing for more accurate detection and localization of anomalies in video frames.

Zhao, Y., Deng, B., Shen, C., Liu, Y., Lu, H., and Hua, X.S. [17] The authors present a method that utilizes a long short-term memory (LSTM) network for temporal modeling in video anomaly detection. Our spatio-temporal autoencoder, however, is designed to process cuboids directly, capturing both spatial and temporal information for improved anomaly detection.

Yuan, Y., Fang, J., and Wang, Q. [18] This study proposes an approach that employs unsupervised feature learning for video anomaly detection. While their method uses a different learning technique, our spatio-temporal autoencoder provides a more suitable solution for handling the temporal aspect of videos.

Tariq, S., Farooq, H., Jaleel, A., and Wasif, S.M. [19] The authors of this paper introduce a method that combines sparse coding with deep learning for video anomaly detection. Our approach, however, relies solely on a spatio-temporal autoencoder, which simplifies the process and offers effective detection of anomalies in videos.

Nayak, R., Pati, U. C., and Kumar Das, S. [20] In this research, the authors present a method that uses a combination of optical flow and deep learning for video anomaly detection.

Varghese, E. B., Thampi, S. M., and Berretti, S. [21] This study uses a fuzzy computational model with psychological inspiration in conjunction with two deep learning networks to recognize intelligent crowd behavior. It uses a deep emotional model to perceive emotions from videos and map them to crowd behaviors using a fuzzy computational model that takes into account multiple emotions and human behavior psychology. The model then employs cognitive-based convolution LSTM and visual attention techniques such as optical flow and saliency detection to predict crowd behaviors. The main contributions of the proposed model are the use of a deep emotional model, For the purpose of predicting crowd behavior, a fuzzy computational model and visual attention are combined.

Habib Ullah, Ahmed B. Altamimi, Muhammad Uzair, and Mohib Ullah. [22] This study proposes the GKIM model, a unique approach for the detection and localization of anomalous phenomena. It may express varying degrees of scatteredness and sparseness in pedestrian flows by modeling anomalous items in a unique way. The GKIM converts spatiotemporal features into a discriminant and unique representation using a Gaussian-based kernel integration and a Jacobian matrix. After a thorough evaluation on three benchmark datasets and a comparison with 11 cutting-edge techniques, it was determined that the suggested GKIM method performs noticeably better than the others. Furthermore, the GKIM technique conducts comprehensive trials and shows superior

anomalous entity detection and localization when compared to 11 other cutting-edge methods, including MDT, MPPCA, SF, MLM, CSC, HF, HFR, PEM, SHM, CDM, and SMR.

## **2.1 EXISTING SYSTEM**

Existing systems for video anomaly detection based on spatio-temporal autoencoders and cuboidal data have been extensively researched in the literature. One such system is STAN (Spatio-Temporal Attention Networks), which employs attention mechanisms in a hierarchical architecture to focus on relevant spatio-temporal regions for anomaly detection. However, scalability issues may arise due to its complex architecture, and the interpretability of attention mechanisms remains a challenge. Another approach is STAE (Spatio-Temporal Autoencoder), utilizing a convolutional autoencoder to learn spatial and temporal features from video data. While effective in detecting simple anomalies, STAE may struggle with complex scenarios and require careful hyperparameter tuning.

Deep SVDD (Deep Support Vector Data Description) applies deep learning techniques to unsupervised anomaly detection but heavily relies on hyperparameters and may not handle high-dimensional or noisy data well. Additionally, models like the convolutional LSTM Recurrent autoencoder offer promising capabilities but may suffer from increased training time and complexity, along with challenges in convergence and stability.

These systems demonstrate potential in video anomaly detection but face limitations such as computational complexity, hyperparameter sensitivity, and difficulties in handling real-world surveillance environments.

### **Drawbacks of the Existing System**

- ❖ **Scalability Issues:** Complex architectures may lead to scalability challenges, impacting the system's ability to handle large-scale datasets efficiently.
- ❖ **Interpretability Challenges:** Incorporation of complex mechanisms like attention may hinder interpretability, making it difficult to understand model decisions.
- ❖ **Hyperparameter Sensitivity:** Models often require careful tuning of hyperparameters, which can be time-consuming and may not always guarantee optimal performance.

- ❖ **Handling Complexity:** Systems may struggle to detect anomalies in complex scenarios, leading to potential false positives or missed detections.
- ❖ **Training Complexity:** Some models exhibit increased training time and complexity, requiring extensive computational resources and expertise.
- ❖ **Convergence and Stability:** Ensuring convergence and stability during training.

#### Existing System-1:

##### Single-Scene Video Anomaly Detection

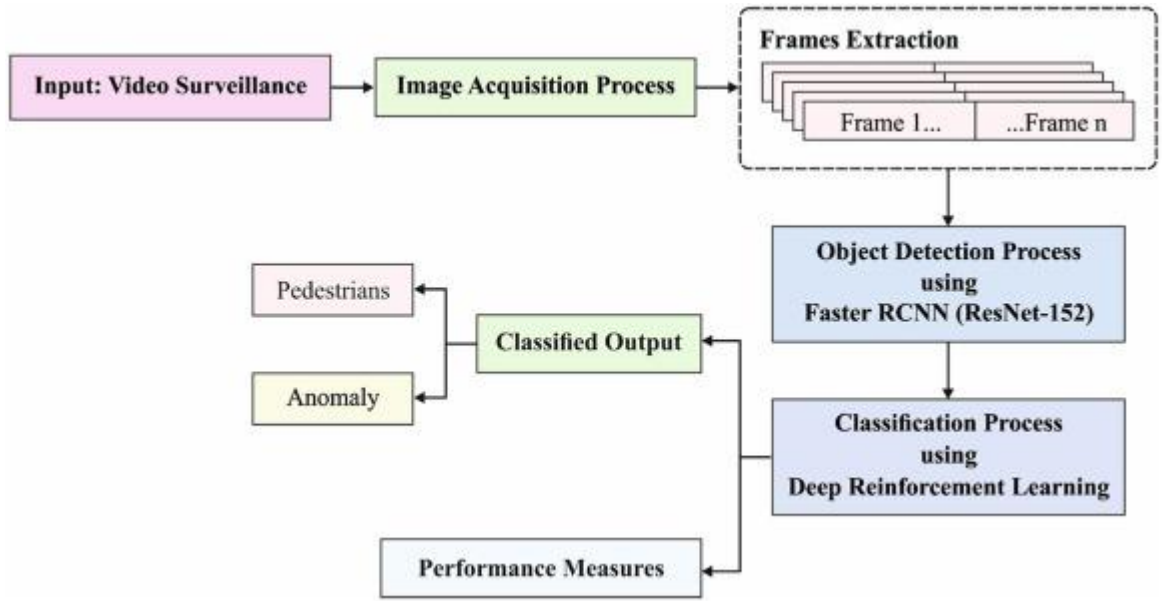


Fig. 1. Existing System: Single-Scene Video Anomaly Detection

#### Existing System-2:

They start by extracting features from video frames using methods like CNNs, shown in Fig. 2. Then, they model temporal patterns using techniques like RNNs or LSTMs. Anomalies are detected by comparing observed patterns to learned normal behaviour, often using threshold-based or probabilistic methods.

However, these systems face challenges such as data dependency, limited generalization, and difficulty in capturing context, real-time performance constraints, labelling biases, and interpretability issues. Despite these drawbacks, ongoing research aims to enhance the robustness and efficiency of these systems.

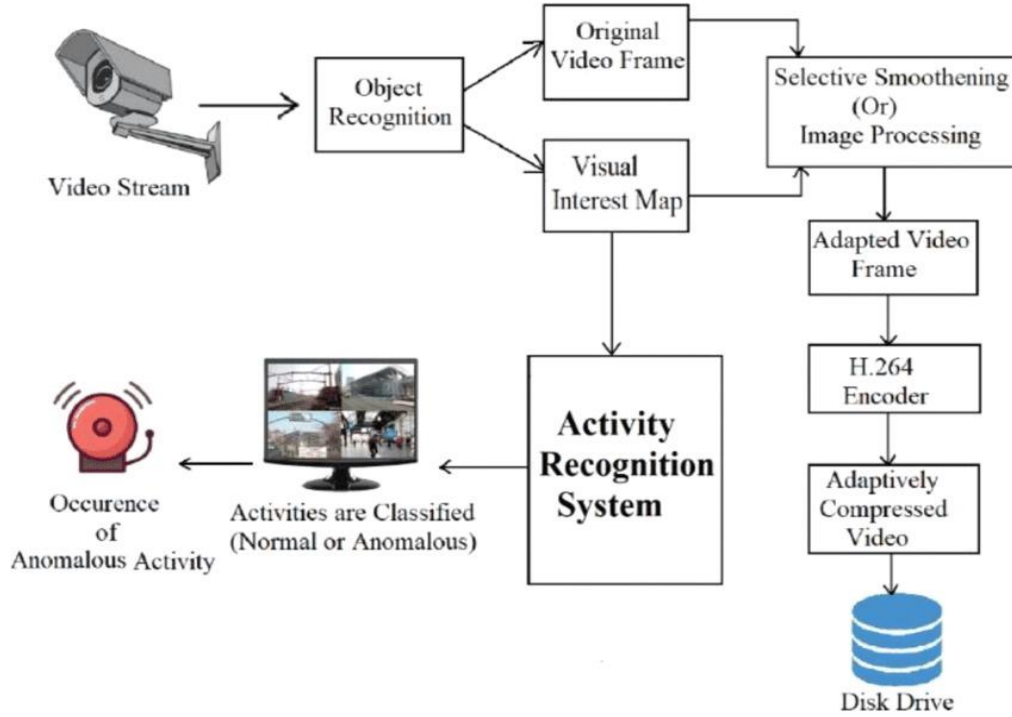


Fig. 2. Existing System

## 2.2 PROPOSED SYSTEM

The proposed system aims to detect anomalies in video sequences by leveraging a spatio- temporal autoencoder-based approach. The system comprises several key stages: video-to-frame conversion, preprocessing of frames, frame-to-cuboid conversion, training of spatio-temporal autoencoders, and anomaly detection. In the following sections, there is an in-depth description of each stage and discussion of the rational design choices.

The process involves handling video frames and integrating image processing with a neural network for anomaly detection. Initially, video data is received and processed using FFMPEG. Frames undergo preprocessing, including grayscale conversion and Gaussian blurring. Processed frames are structured for further use. The core comprises an encoder-decoder neural network model capturing complex patterns within frames. The model's output is scrutinized for anomalies, distinguishing between "Abnormal" and "Normal" subjects using bounding boxes, indicating subject classification within frames. This process offers a holistic view of video processing, merging image processing with deep learning for real-time anomaly detection in video streams.

### 2.2.1 Video to Frame conversion

The first stage of the proposed system involves converting the input video into a sequence of frames. FFmpeg, a widely-used open-source multimedia framework, is used to perform this conversion efficiently, extracting individual frames and analyzing the video's content in detail to gain insights into the spatial and temporal patterns present in the data.

### 2.2.2 Preprocessing of frames

Once the video has been converted into frames, perform preprocessing steps to prepare the data for subsequent analysis. The preprocessing consists of two main steps: grayscaling and Gaussian blur.

Grayscaling conversion of color frames to grayscale images, which simplifies the data and reduces the computational complexity of the subsequent stages. By removing color information, this will focus on the structural and textural patterns in the video, which are more relevant for anomaly detection.

After greyscaling, apply a Gaussian blur to the frames. This step smooths the images and reduces noise, helping the spatio-temporal autoencoder to better focus on the significant patterns in the data. The Gaussian blur also aids in suppressing irrelevant details that might lead to false positives during anomaly detection.

$$G(x,y) = (1 / 2\pi\sigma^2) * e^{-(x^2 + y^2) / 2\sigma^2}$$

In above equation,

$G(x, y)$  is the value of the Gaussian function at pixel coordinates  $(x, y)$ .

Sigma is the standard deviation of the Gaussian distribution. It controls the amount of blurring applied to the image. A larger value of sigma will result in a more significant blur, while a smaller value will produce a more subtle blur.

$e$  is the base of the natural logarithm (approximately equal to 2.71828).

$x$  and  $y$  are the horizontal and vertical pixel coordinates, respectively.



The Gaussian function is essentially a bell-shaped curve, which assigns higher weights to the pixels closer to the center and lower weights to the pixels further away. The standard deviation, sigma, determines the width of the bell curve. When a larger sigma value is used, the curve will be wider, resulting in a more significant blur effect.

The Gaussian blur is applied to an image by convolving it with the Gaussian kernel generated using the Gaussian function. The convolution process computes a weighted average of the pixel values in the original image, with the weights determined by the Gaussian kernel. This results in a smooth, blurred version of the original image.

### 2.2.3 Frame to cuboid conversion

In Fig. 3, provides a detailed illustration of the cuboid extraction and anomaly detection process within a given frame. The figure showcases the following key steps:

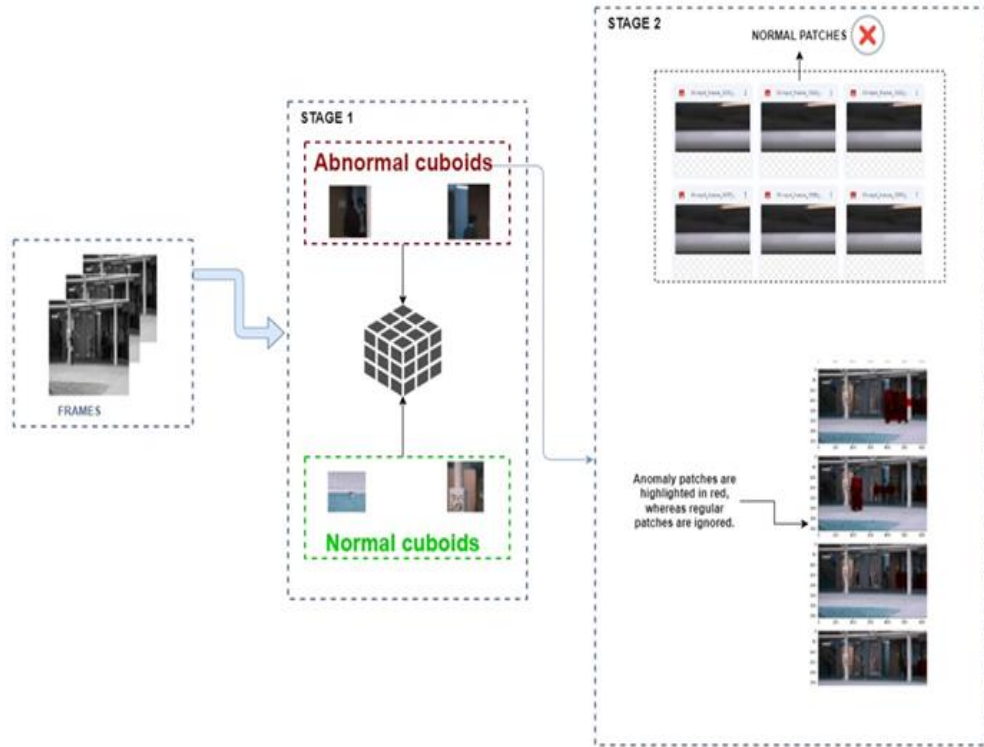


Fig. 3. Cascade for anomaly detection

**Cuboid Extraction:** The input frame is systematically divided into smaller, overlapping, three-dimensional data structures called cuboids. Every cuboid records a distinct spatiotemporal area in the video sequence, encompassing both temporal (i.e., the frame's position within the sequence) and spatial information (i.e., the content of the frame). The system can evaluate each region independently by dividing the frame

into these smaller parts, which enables more precise anomaly identification. The trained spatio-temporal autoencoders next process the retrieved cuboids and try to reconstitute the input cuboids. The input cuboids are first encoded by the autoencoders into a lower-dimensional representation, which they subsequently decode back into the original form. (Fig.3)

## **2.2 NOVELTY OF OUR PROJECT**

The novelty of our project is that it presents an innovative method for video anomaly detection, employing spatio-temporal autoencoders trained on datasets like the Avenue and Anomaly detection datasets. By integrating advanced algorithms such as CNN-LSTM alongside spatio-temporal autoencoders, our approach emphasizes both spatial and temporal features, enhancing the model's ability to comprehensively analyze video data. The process of converting videos into frames and subsequently into cuboids yields higher accuracy when compared to alternative techniques. Developed using Streamlit, our web application provides users with an intuitive platform for uploading videos and detecting anomalies in real-time. Through dynamic frame analysis, our system identifies anomalies and presents them via reconstructing frame vs. cost graphs, supported by frame per second (fps) analysis for heightened accuracy.

## **2.3 CONCLUSION**

Video anomaly detection has seen considerable research activity in recent years due to its potential applications in various fields, including security and surveillance. Different papers along with knowledge resources are researched and thus the proposed system is made by incorporating different features from the survey of research papers.

Table 1. Literature Survey

| S.No. | Title & Authors  | Published Journal and Year   | Dataset used             | Approach  | Outcomes   | Limitations  |
|-------|--|--|--------------------------|---|--|--|
| 1.    | AnomalyNet: An Anomaly Detection Network for Video Surveillance, Jia Tao Zhou  | IEEE Transactions on Information Forensics and Security, 2019        | Avenue Benchmark Dataset | Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) | False positive rate and computational efficiency | Computational complexity                             |
| 2.    | Video Anomaly Detection with Sparse Coding Inspired Deep Neural Networks, Wen Luo  | IEEE Transactions on Pattern Analysis and Machine Intelligence, 2019 | Avenue and UCSD          | Variational Autoencoders (VAEs)                                       | Detection accuracy                               | Scalability and generalization to diverse scenarios. |
| 3.    | Gaussian Process Regression-Based Video Anomaly Detection and Localization with Hierarchical Feature Representation, Kwang-Wen Cheng | IEEE Transactions on Image Processing, 2015                          | CUHK Dataset             | Gaussian Process Regression (GPR)                                     | Detecting and localizing anomalies in video data | Robustness to different types of anomalies           |

|    |   |  |                           |   |   |                             |
|----|---|--|---------------------------|---|---|-----------------------------|
| 4. | Sparse Coding Guided Spatiotemporal Feature Learning for Abnormal Event Detection in Large Videos, Wei Chu            | IEEE Transactions on Multimedia, 2018  | Anomaly Dataset           |   | Detecting abnormal events in large video datasets | Computational complexity    |
| 5. | Spatio-Temporal Unity Networking for Video Anomaly Detection, Yaqian Li   | Institute of Electrical and Electronics Engineers, 2019                              | UCSD Dataset              | Hybrid method for video anomaly detection   | Qualitative analysis                              | Potential trade-offs        |
| 6. | Unsupervised Anomaly Detection and Localization Based on Deep Spatiotemporal Translation Network, Tharit Ganokratanaa | Institute of Electrical and Electronics Engineers, 2020                              | Available CCTV Data, UCSD | Deep Spatiotemporal Translation Network (DSTN), Generative Adversarial Network (GAN)            | Improved anomaly detection                        | Complex, difficult to scale |
| 7. | Detecting Abnormality Without Knowing Normality: A Two-Stage Approach for Unsupervised Video Abnormal                 | Proceedings of the 26 <sup>th</sup> ACM International Conference on Multimedia, 2018 | Avenue Dataset            | Two-stage approach for unsupervised video abnormal event detection based on graph-based methods | Precise localization                              | Interpretability            |

|            |  |   |                          |  |   |   |
|------------|--|---|--------------------------|--|---|---|
|            | Event Detection,<br>Shuang Wang  |   |                          |  |   |   |
| <b>8.</b>  | Domain-Transformable Sparse Representation for Anomaly Detection in Moving-Camera Videos, Eros Jardim    | IEEE Transactions on Image Processing, 2019       | Avenue Benchmark Dataset | Domain-Transformable sparse representation | Effectively identified anomalies in video data with high accuracy | Scalability and complexity                  |
| <b>9.</b>  | Spatial-Temporal Cascade Autoencoder for Video Anomaly Detection in Crowded Scenes, Ning Li              | IEEE Transactions on Multimedia, 2020             | Available CCTV Dataset   | 3D Convolutional Neural Network (CNN)      | Effective and efficient criminal detection                        | Privacy concerns, data quality dependencies |
| <b>10.</b> | Spatiotemporal Anomaly Detection Using Deep Learning for Real-Time Video Surveillance, Roshan Nawaratne. | IEEE Transactions on Industrial Informatics, 2019 | Anomaly Dataset          | Recurrent Neural Networks (RNNs)           | Showcased robustness across various datasets                      | Biases in machine learning models           |
| <b>11.</b> | Spatial-temporal cascade autoencoder for video anomaly detection in crowded scenes                       | Journal of electromechanical systems, 2020        | Avenue Dataset           | 3D Convolution Neural Network              | Improved anomaly localization, Broad applicability                | Computational complexity                    |

|     |   |  |                          |   |   |  |
|-----|---|--|--------------------------|---|---|--|
| 12. | Video behavior profiling for anomaly detection. Xiang, T. and Gong, S.  | IEEE, Transactions on pattern analysis and machine intelligence, 2008          | UCSD Dataset             | Hierarchical attention mechanism  | Sensitivity to threshold selection to detection accuracy                        | Potential False Negatives  |
| 13. | Anomaly detection in moving-camera video sequences using principal subspace analysis Thomaz, L.A., Jardim, E., da Silva, A.F., da Silva, E.A., Netto, S.L. and Krim, H. | IEEE, Transactions on Circuits and Systems I: Regular Papers, 2017             | CUHK Dataset             | Generative Adversarial Network (GAN)  | Focused and streamlined approach  | Time consuming and may require careful parameter tuning.             |
| 14. | Anomaly detection and localization in crowded scenes. Li, W., Mahadevan, V. and Vasconcelos, N.   | IEEE, Transactions on pattern analysis and machine intelligence, 2013          | Available CCTV Dataset   | Combination of handcrafted features and deep learning                                   | Enhanced accuracy and efficiency  |  |
| 15. | Spatio-temporal autoencoder for video anomaly detection Zhao, Y., Deng, B., Shen, C., Liu, Y.   | IEEE, Proceedings of the 25th ACM international conference on Multimedia, 2017 | Avenue Dataset           | Long short-term memory (LSTM) network for temporal modeling in video anomaly detection. | Capturing both spatial and temporal information for improved anomaly detection. | Making it difficult to interpret and understand the learned features |
| 16. | Anomaly detection based on stacked sparse coding with intraframe classification   | IEEE, Transactions on Multimedia, 2018   | Avenue benchmark Dataset | Multi-Scale Analysis and spatio-temporal  | More accurate detection and localization of anomalies in video                  | Generalization of new types of anomalies                             |

|     |   |  |                        |   |  |                                 |
|-----|---|--|------------------------|---|--|---------------------------------|
|     | strategy<br>Xu, K., Jiang, X. and Sun.  |  |                        |   | frames   |                                 |
| 17. | Online anomaly detection in crowd scenes via structure analysis<br>Yuan, Y., Fang, J. and Wang, Q.                            | IEEE, transactions on cybernetics, 2014  | Anomaly Dataset        | Unsupervised feature learning for video anomaly detection | Provides a more suitable solution for handling the temporal aspect of videos | Potential False Negatives       |
| 18. | Anomaly detection with particle filtering for online video surveillance.<br>Tariq, S., Farooq, H., Jaleel, A. and Wasif, S.M. | IEEE, Access on Multimedia, 2021   | CUHK Dataset           | Sparse coding with deep learning                          | Simplifies the process and offers effective detection of anomalies           | Limited or biased training data |
| 19. | "Video Anomaly Detection using Convolutional Spatiotemporal Autoencoder,"<br>R. Nayak, U. C. Pati and S. Kumar Das.           | IEEE, International Conference on Contemporary Computing and Applications (IC3A), Lucknow, India, 2020 | Avenue Dataset         | Combination of optical flow and deep learning             | Improved anomaly detection accuracy  | Data preprocessing overhead     |
| 20. | Spatio-temporal unity networking for video anomaly detection<br>Li, Y., Cai, Y., Liu, J., Lang, S.                            | IEEE, Transactions on Image Processing, 2019   | Available CCTV Dataset | Cognitive-based convolution LSTM and visual attention     | The incorporation of visual attention for crowd behavior prediction          | Dependency on training data     |

### 3. REQUIREMENT ANALYSIS

#### 3.1 INTRODUCTION

Video surveillance systems are essential for ensuring security and safety across various domains, including public spaces, transportation hubs, and critical infrastructure. However, traditional surveillance methods often struggle to effectively detect anomalies and abnormal events in real-time, leading to delayed responses and increased risks. To address these challenges, our project focuses on the development of an advanced video anomaly detection system using spatio-temporal autoencoders and cuboidal data. The primary goal of this project is to leverage deep learning techniques to enhance the capabilities of surveillance systems in detecting and mitigating security threats.

#### 3.2 SOFTWARE REQUIREMENT SPECIFICATION

##### 3.2.1 Software Requirements

The software requirements include the languages, libraries, packages and operating system, and different tools for developing the project. We used python for coding Refer table 1 and table 2 for software requirements.

##### Minimum Software Requirements

Table 2. Software Requirements

| Software                  | Version                     |
|---------------------------|-----------------------------|
| Google Colab              | 3.7                         |
| Visual Studio Code Editor | 1.76.0                      |
| Operating System          | 64-bit OS, Windows 11, 21H2 |
| Python                    | 3.10.12                     |



## Python packages

Python packages are collections of modules and functions that extend Python's functionality for specific tasks. Popular packages include NumPy for scientific computing, Pandas for data manipulation, Matplotlib for visualization, Scikit-learn for machine learning, TensorFlow and PyTorch for deep learning, Requests for making HTTP requests, and BeautifulSoup for parsing HTML and XML documents.

These packages provide efficient tools for a wide range of tasks such as data analysis, machine learning, web scraping, and more, making Python a versatile and powerful language for various domains.

### Python packages we used in our project includes the following table 2.

We used the command **pip install “package\_name==version”** to download the packages.

Table 3. Python packages

| S. No. | Packages used       | Description   |
|--------|---------------------|---|
| 1.     | OpenCV(4.9.0)       | Image and video processing library  |
| 2.     | Streamlit(1.33.0)   | Web application framework for building interactive apps   |
| 3.     | PIL (1.1.7)         | Python Imaging Library for image processing   |
| 4.     | os (3.12.3)         | Operating system related functions  |
| 5.     | tensorflow (2.11.0) | It is a python-based package which provides implementing different machine learning and AI tools  |
| 6.     | sklearn (1.0.1)     | Scikit-learn is an open source Python package functionality supporting supervised and unsupervised learning   |
| 7.     | matplotlib (3.4.3)  | It is a cross-platform, data visualization and graphical plotting library for python and its numerical extension  |
| 8.     | numpy (1.24.1)      | Numpy package is fast and accurate, used for scientific computing in python   |
| 9.     | pandas (1.5.3)      | Pandas is a fast, scalable package used for data science operations and expensive data structures operations.   |
| 10.    | keras (2.11.0)      | Keras is a minimalist Python library for deep learning that can run on top of Theano or Tensorflow. It consists of all the neural network layers and other packages used for deep learning computations |

|     |               |  |
|-----|---------------|--|
| 11. | Json (3.4 )   | Json is a lightweight data interchange format used to store and exchange data between a server and a web application.                                  |
| 12. | Ffmpeg(0.2.0) | ffmpeg is a powerful multimedia processing tool that allows users to decode, encode, transcode, mux, demux, stream, filter, and play multimedia files. |

### Front End Packages used

Table 4. Front end Packages

| S. No. | Libraries Used | Description  |
|--------|----------------|--|
| 1.     | Streamlit      | Python library for creating interactive web applications with data science tools |
| 2.     | os             | In Python , the OS module has functions for dealing with the operating system.   |

### 3.2.2 Hardware Requirements

The below Table 4 shows the minimum hardware requirements of our project.

#### Minimum Hardware Requirements

Table 5. Minimum Hardware Requirements

| Hardware Requirements | Configuration  |
|-----------------------|--|
| Processor             | Intel core with i5 configuration or equivalent processors. |
| RAM                   | 16 GB  |
| Hard Disk             | 256 GB or greater  |

### 3.3 NON-FUNCTIONAL REQUIREMENTS

- **Compatibility:** According to the definition, Compatibility is the capacity for two systems to work together without having to be altered to do so. This project is compatible to run on Windows 10,11
- **Capacity:** It can be stated as the capacity of a system refers to the amount of storage it utilizes. This project work i5 processor of 16 GB RAM.

- **Environment:** It can be stated as all the external and internal forces that exert on your project. This project works on python 3.10.12 environment and higher versions
- **Performance:** The performance of this project is analyzed based on the accuracy of model and confusion matrix.
- **Reliability:** It can be stated as the extent to which the software system consistently performs the specified function without failure. In this project, output is analyzed is based on the dataset which is taken in controlled environment

### 3.4 CONTENT DIAGRAM OF OUR PROJECT

The architectural diagram outlines a comprehensive process for handling video frames, integrating image processing techniques with a neural network model for anomaly detection. Initially, the system ingests video data from a specified source, utilizing FFMPEG for media processing tasks. The extracted original frames undergo preprocessing stages involving grayscale conversion and Gaussian blurring, as illustrated by distinct visual representations. Subsequently, these processed frames are channeled into a structured data entity, potentially facilitating further manipulations or storage. The crux of the system lies in an encoder-decoder neural network model, which likely encapsulates complex patterns within the video frames. The model's output is then scrutinized for anomaly detection, discerning between "Abnormal subjects" and "Normal subjects" through bounding boxes, implying a categorization mechanism for classifying subjects within the video frames. This architectural depiction offers a holistic overview of the video processing pipeline, showcasing the integration of image processing methodologies with deep learning techniques for real-time anomaly detection in video streams.

The project on video anomaly detection using spatio-temporal autoencoders follows a structured approach outlined in the content diagram. Beginning with an introduction to the project's objectives, the background section provides context by reviewing existing methods and underscoring the necessity for spatio-temporal autoencoders. The problem statement defines the task of anomaly detection in video streams, while the methodology elucidates the approach, emphasizing the architecture of

autoencoders and their role in detecting anomalies through reconstruction error. Practical aspects such as data collection, preprocessing, and model implementation are detailed, followed by experimentation and results showcasing performance metrics and anomaly detection examples. Discussion critically evaluates the findings, drawing comparisons with existing methods and suggesting areas for improvement. The conclusion summarizes the project's outcomes, while the future work section proposes avenues for further research. The content diagram ensures a cohesive and comprehensive presentation of the project, guiding the reader through its objectives, methodology, findings, and implications.

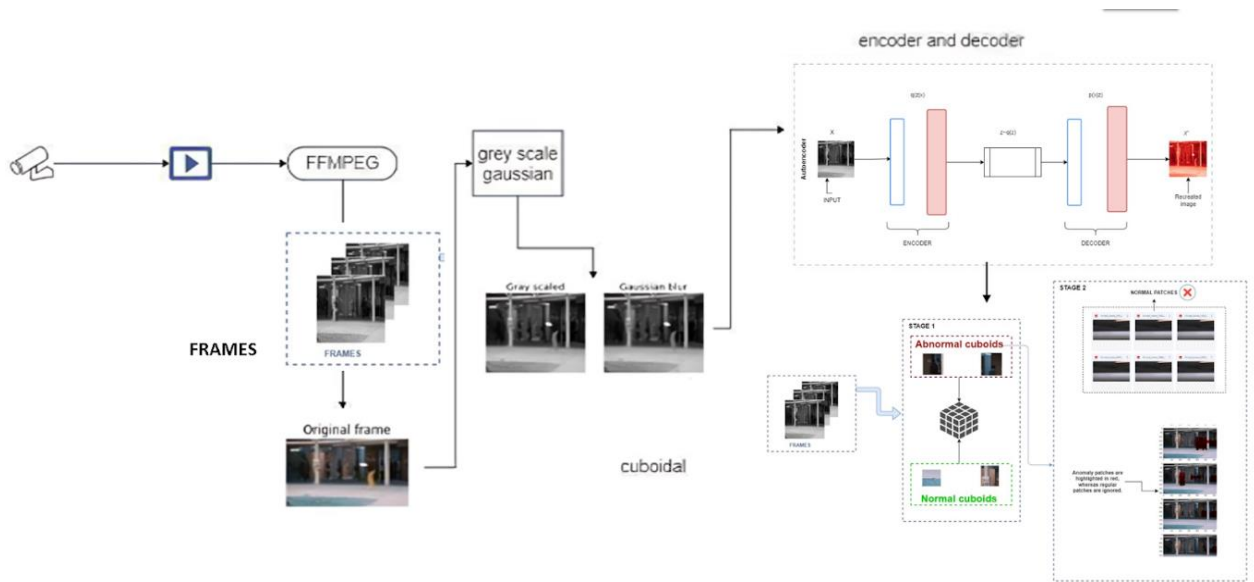


Fig. 4. Architecture Diagram

### 3.5 FLOW CHART OF THE PROJECT

The process for training and evaluating a machine learning model for anomaly detection in video sequences begins with the input of video data, which is categorized into an anomaly detection dataset. This dataset is split into training, validation, and test sets, with 80% of normal testing videos allocated for training, 20% for validation, and the remainder for testing. Preprocessing is then applied to each subset to prepare the data for model input. A convolutional neural network (CNN) model is trained on the training set, with the model's performance assessed through loss calculation and threshold definition. If the model's performance during validation meets expectations, it proceeds to testing, where

it evaluates loss values for each video in the test set. Videos with a loss exceeding the defined threshold are classified as containing anomalies, while those below the threshold are considered normal. Post-processing involves converting each cuboid to .npy format for storage in a data frame, readying the model for deployment or further evaluation.

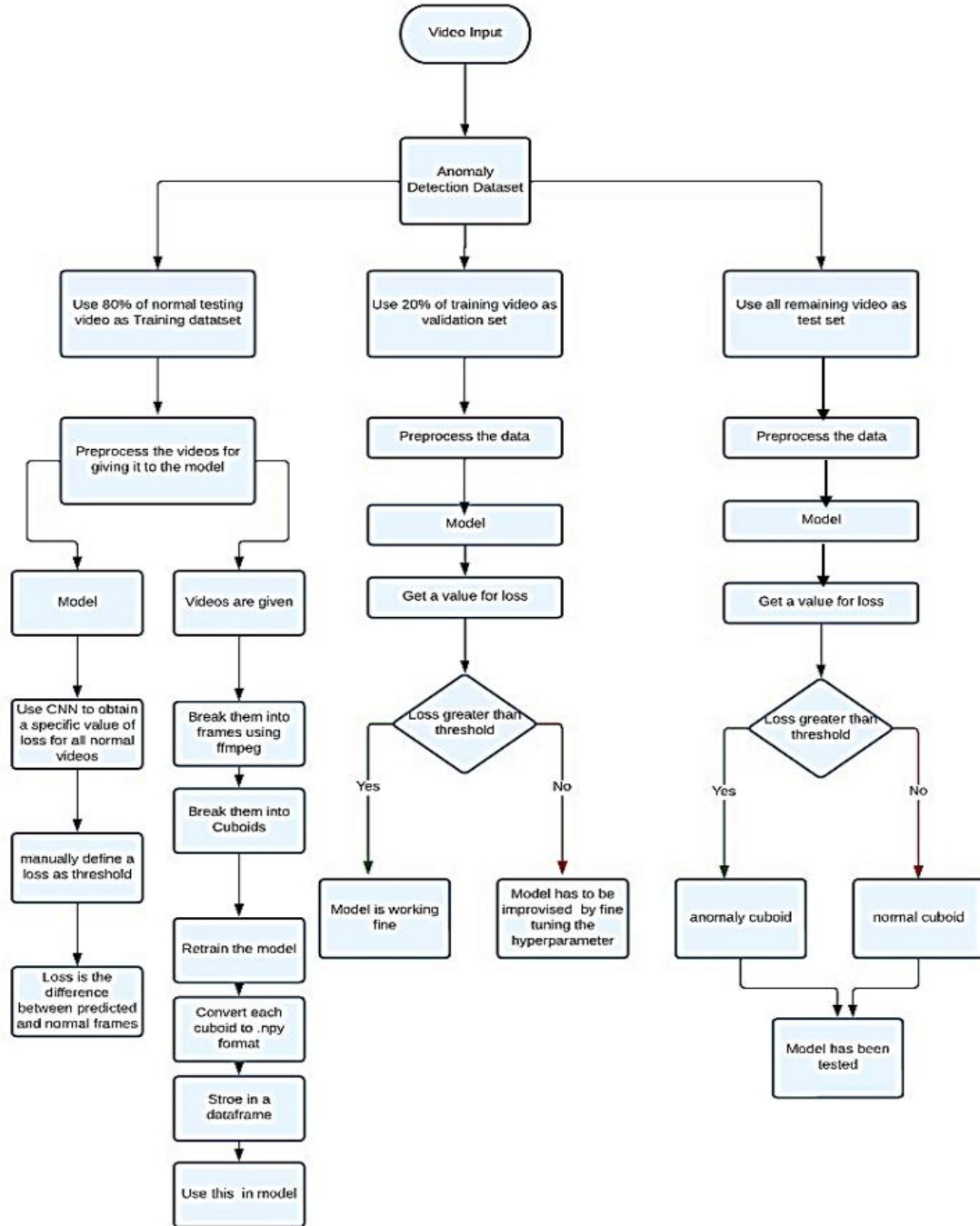


Fig. 5. Flow diagram of the proposed method

## 3.6 ALGORITHMS

### SPATIO-TEMPORAL AUTOENCODER

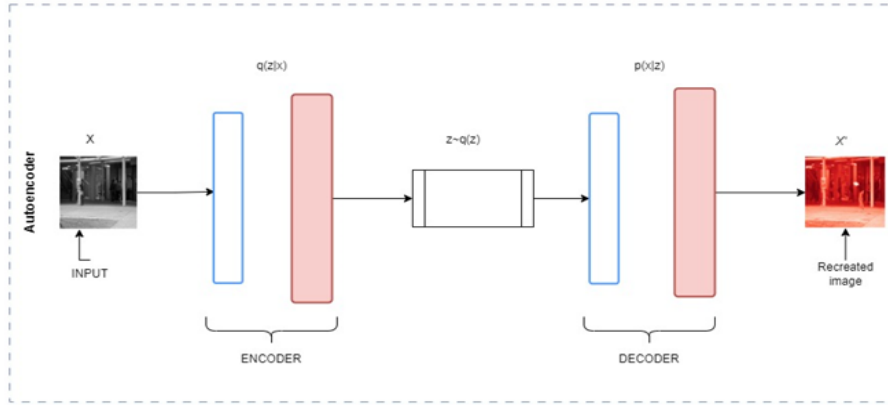


Fig. 6. Working of autoencoder and decoder

In Fig. 6, presents a detailed illustration of how an autoencoder and decoder work together within the context of a deep learning model. The autoencoder is composed of two primary components: the encoder and the decoder. The figure showcases the following key steps in the process:

**Input Image (x):** The input image, denoted as 'x', is fed into the autoencoder. This image contains the spatial and temporal information that the model will attempt to compress and reconstruct.

An autoencoder is a type of neural network that learns to encode and decode data in a lower-dimensional space. It consists of two parts: an encoder and a decoder. Here, equations are explained in detail.

**Encoder:** The encoder maps the input data,  $X$ , into a lower-dimensional latent space,  $Z$ . This is done using a series of layers with weights ( $W$ ) and biases ( $b$ ). For simplicity, let's consider a single-layer encoder:

$$Z = f(W_e \cdot X + b_e)$$

In above equation:

- $X$  is the input data (e.g., an image or a video frame).
- $W_e$  is the weight matrix for the encoder layer.
- $b_e$  is the bias vector for the encoder layer.
- $f(.)$  is the activation function applied element-wise, such as ReLU, sigmoid, or tanh.
- $Z$  is the lower-dimensional representation (latent space) of the input data.

The encoder aims to compress the input data into a more compact representation while preserving as much information as possible.

Decoder: The decoder takes the latent representation,  $Z$ , and reconstructs the original input data,  $X'$ . This is also done using a series of layers with weights ( $W$ ) and biases ( $b$ ). For simplicity, let's consider a single-layer decoder:

$$X' = g(W_d \cdot Z + b_d)$$

In above equation:

- $Z$  is the lower-dimensional representation (latent space) of the input data.
- $W_d$  is the weight matrix for the decoder layer.
- $b_d$  is the bias vector for the decoder layer.
- $g(.)$  is the activation function applied element-wise, such as ReLU, sigmoid, or tanh.
- ' $X$ ' is the reconstructed output image, which should ideally be a close approximation

The input image  $X$ , with minimal loss of information. The success of the autoencoder model is often evaluated by comparing the input image  $X$  and the reconstructed output image  $X'$ .

## **Training of Spatio-Temporal Autoencoders**

The core of the proposed system is the use of spatiotemporal autoencoders to learn the normal patterns present in the video sequences. Train a separate autoencoder for each cuboid folder, allowing the model to specialize in capturing the spatial and temporal features specific to that region. This approach enables the system to better detect anomalies, as it can account for variations in normal behavior across different regions of the video.

During training, the spatiotemporal autoencoder learns to reconstruct the input cuboids by encoding them into a lower-dimensional representation and then decoding them back to their original form. This process allows the autoencoder to capture the most salient features of the data while ignoring the less important details.

### **1. Model Initialization:**

At the outset, the spatiotemporal autoencoder model is initialized with random weights. These weights define the initial parameters of the neural network architecture.

### **2. Input Encoding:**

Each cuboid, representing a spatial region within the video frame, is fed into the encoder portion of the autoencoder. The encoder learns to compress the input data into a lower-dimensional latent representation.

Through successive layers of convolutional and pooling operations, the encoder extracts high-level features from the cuboid, effectively reducing the dimensionality of the input data.

### **3. Input Reconstruction:**

The compressed latent representation obtained from the encoder is then passed through the decoder portion of the autoencoder. The decoder attempts to reconstruct the original input data from the compressed representation.

By applying transposed convolutional layers and upsampling operations, the decoder upscales the latent representation back to the original spatial dimensions of the input cuboid.



#### **4. Loss Calculation:**

After the reconstruction process, the autoencoder computes the reconstruction loss, typically using mean squared error (MSE) or another suitable loss function. This loss function quantifies the discrepancy between the original input data and the reconstructed output.

The reconstruction loss serves as a measure of how well the autoencoder can reconstruct the input cuboid. A lower reconstruction loss indicates that the autoencoder has successfully captured the salient features of the data.

By iteratively adjusting the model parameters through backpropagation and gradient descent, the spatiotemporal autoencoder learns to minimize the reconstruction loss, effectively capturing the normal patterns present in the video sequences. This process enables the model to specialize in capturing spatial and temporal features specific to each cuboid region, enhancing its ability to detect anomalies by accounting for variations in normal behavior across different regions of the video.

### **Hybrid Model of CNN and LSTM**

#### **Convolutional Neural Networks (CNN)**

Convolutional neural networks are composed of multiple layers of artificial neurons. A Convolutional Neural Network typically involves two operations, which can be thought of as feature extractors: convolution and pooling. In the convolution layer when an input sequence is given where each is associated with an embedding vector of a dimension. A kernel of size  $k$  is chosen, which is a sliding window of size  $k$  that moves over the sentence and applies the same convolution filter as it moves, i.e., a dot-product between the concatenation of the embedding vectors in a given window and a weight vector. In a Pooling layer a single, two-dimensional vector is created by pooling the vectors produced by many convolution windows. This is repeated by taking the maximum or average value noted in the vector that results from the convolutions. This vector should ideally include all of the important elements of the sentence or document. The figure 7 shows the architecture.

## Long Short-Term memory (LSTM):

LSTM is a type of recurrent neural network but is better than traditional recurrent neural networks in terms of memory. Traditional neural networks suffer from short-term memory. LSTMs efficiently improve performance by memorizing the relevant information that is important and finds the pattern.

```
def load_model():
    model = Sequential()
    model.add(Conv3D(filters=128, kernel_size=(11, 11, 1), strides=(4, 4, 1), padding='valid', input_shape=(227, 227, 10, 1), activation='tanh'))
    model.add(Conv3D(filters=64, kernel_size=(5, 5, 1), strides=(2, 2, 1), padding='valid', activation='tanh'))
    model.add(ConvLSTM2D(filters=64, kernel_size=(3, 3), strides=1, padding='same', dropout=0.4, recurrent_dropout=0.3, return_sequences=True))
    model.add(ConvLSTM2D(filters=32, kernel_size=(3, 3), strides=1, padding='same', dropout=0.3, return_sequences=True))
    model.add(ConvLSTM2D(filters=64, kernel_size=(3, 3), strides=1, return_sequences=True, padding='same', dropout=0.5))
    model.add(Conv3DTranspose(filters=128, kernel_size=(5, 5, 1), strides=(2, 2, 1), padding='valid', activation='tanh'))
    model.add(Conv3DTranspose(filters=1, kernel_size=(11, 11, 1), strides=(4, 4, 1), padding='valid', activation='tanh'))
    model.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy'])
    model.summary()
load_model()
```

Model: "sequential\_1"

| Layer (type)                          | Output Shape            | Param # |
|---------------------------------------|-------------------------|---------|
| conv3d_2 (Conv3D)                     | (None, 55, 55, 10, 128) | 15616   |
| conv3d_3 (Conv3D)                     | (None, 26, 26, 10, 64)  | 204864  |
| conv_lstm2d_3 (ConvLSTM2D)            | (None, 26, 26, 10, 64)  | 295168  |
| conv_lstm2d_4 (ConvLSTM2D)            | (None, 26, 26, 10, 32)  | 110720  |
| conv_lstm2d_5 (ConvLSTM2D)            | (None, 26, 26, 10, 64)  | 221440  |
| conv3d_transpose_2 (Conv3D Transpose) | (None, 55, 55, 10, 128) | 204928  |
| conv3d_transpose_3 (Conv3D Transpose) | (None, 227, 227, 10, 1) | 15489   |

=====  
Total params: 1068225 (4.07 MB)  
Trainable params: 1068225 (4.07 MB)  
Non-trainable params: 0 (0.00 Byte)

Fig. 7. Code Snippet of Autoencoder Model

To solve the issue of long-term dependency that recurrent neural networks RNNs had, LSTM networks were created (due to the vanishing gradient problem). When compared to more conventional feed forward neural networks, LSTMs vary because they have feedback connections. Because of this trait, LSTMs may process whole data sequences (such as time series) without having to treat each data point separately. Instead, they can process new data points by using the knowledge from prior data in the sequence that they have stored as meaningful information. Because of this, LSTMs excel at processing data sequences like text, audio, and general time-series.

Initially, at a fundamental level, an LSTM's output at a given moment depends on three factors:

- The network's present long-term memory, often known as the cell state
- The output at the previous point in time, often known as the previous hidden state
- The input information for the current time step.

In LSTMs, a number of "gates" regulate how data in a sequence enters, is stored in, and leaves the network. A typical LSTM has three gates: an output gate, an input gate, and a forget gate. Each of these gates is a separate neural network and may be thought of as a filter.

The forget gate is the initial stage of the procedure. At this step, we will determine which pieces of the cell state—the network's long-term memory—are relevant in light of both the previous hidden state and the new input data. The new memory network and the input gate are part of the next step. This step's objective is to decide what new information, in light of the previous hidden state and the new input data, has to be added to the network's long-term memory (cell state). We next go to the output gate, which determines the new hidden state, when we have finished updating the network's long-term memory. We'll employ three factors—the recently updated cell state, the previous hidden state, and the fresh input data—to make this decision.

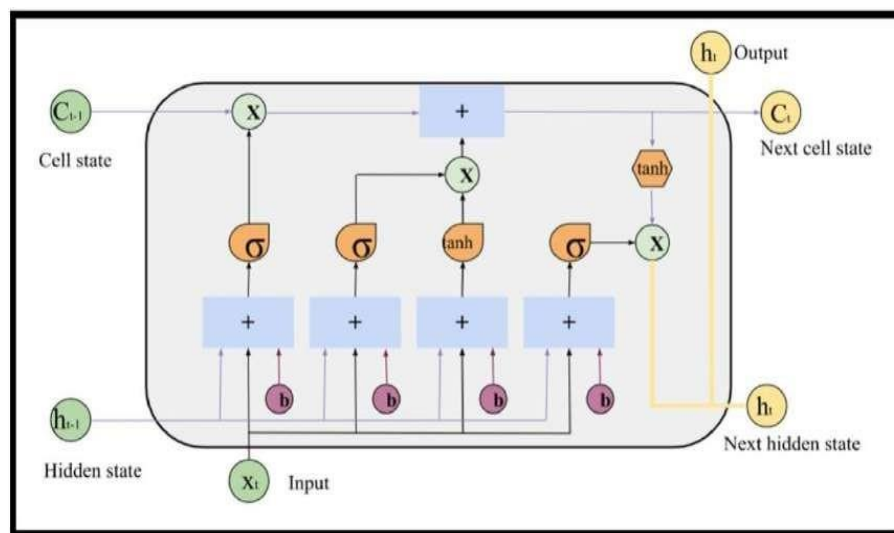


Fig. 8. LSTM Architecture

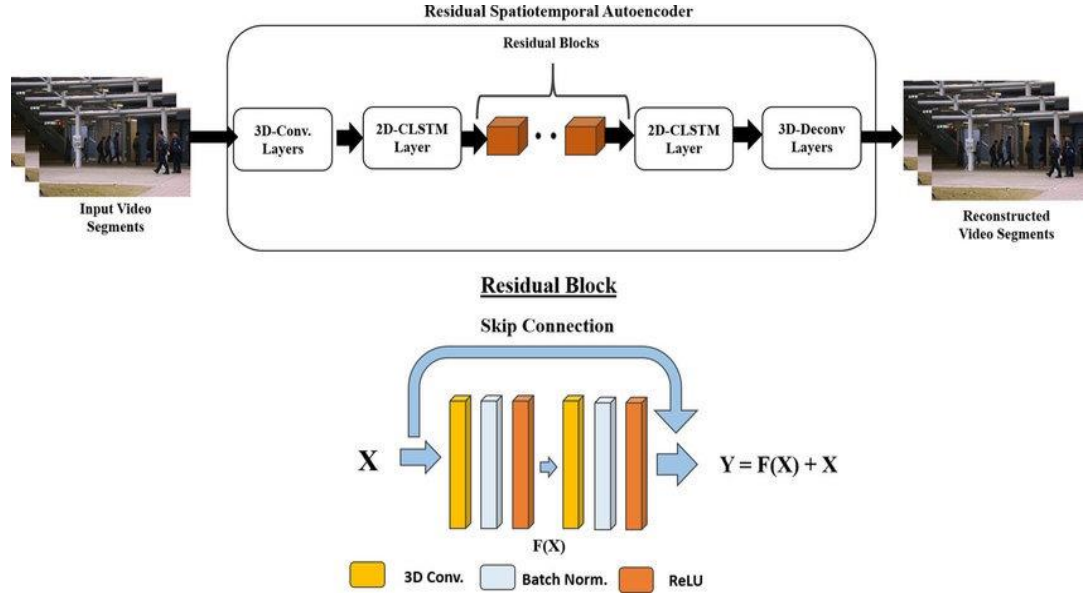


Fig. 9. Hybrid Model of CNN and LSTM

## Anomaly Detection Algorithms

### Reconstruction Error

In the context of video anomaly detection using spatio-temporal autoencoders, the reconstruction error plays a crucial role in identifying and localizing anomalous events. The reconstruction error is calculated as the difference between the original input data (cuboids) and the output generated by the autoencoder after the encoding and decoding process. This difference can be quantified using various distance metrics such as mean squared error (MSE), Euclidean distance, or cosine similarity, depending on the application and desired properties of the metric.

The reconstruction error serves as an indicator of how well the autoencoder has learned the underlying patterns and structure of the input data. A low reconstruction error implies that the input data is well-represented by the learned model, suggesting normal behavior. Conversely, a high reconstruction error indicates that the input data deviates from the learned model, which could signal the presence of an anomaly. This deviation arises from the autoencoder's inability to accurately reconstruct anomalous events due to their infrequent occurrence during the training phase.

### **Fixed Thresholding Technique**

The fixed thresholding technique is employed to classify cuboids as normal or anomalous based on their reconstruction error. A predetermined threshold value is set, above which a cuboid is considered anomalous. This technique offers a simple, computationally efficient approach to detect anomalies and can be easily implemented in practice.

The optimal threshold value is determined through experimentation and analysis of the training data. It is crucial to strike a balance between sensitivity (correctly identifying anomalies) and specificity (correctly identifying normal events). The choice of threshold value directly impacts the overall performance of the anomaly detection system, as it determines the trade-off between false positives (normal events classified as anomalies) and false negatives.

### **Localization of Anomalous Regions**

The cuboid-based processing approach provides a fine-grained representation of the video data, enabling the system to localize anomalies in both space and time. By analyzing the reconstruction errors of individual cuboids, the system can pinpoint the exact regions of interest within the video frame that contain anomalous events.

Additionally, the temporal localization of anomalies is achieved by tracking the progression of anomalous cuboids across consecutive frames. The ability to localize anomalous regions in both space and time is an essential feature of the proposed system, as it enables users to quickly identify and address potential issues or threats that may be present in the monitored environment.

After training the spatiotemporal autoencoders, it can be used to detect anomalies in new video sequences. During the testing phase, first convert the test video into frames and preprocess them as before. Next, transform the frames into cuboids and feed them to their respective autoencoder models for reconstruction.

By comparing the input cuboids with their reconstructed counterparts, we can compute a loss value representing the difference between the two. If the loss exceeds a predefined threshold, classify the cuboid as anomalous. This approach allows us to detect anomalies at a granular level and pinpoint their locations within individual frames.

## 4. DESIGN

### 4.1 INTRODUCTION

Unified modelling language (UML) is used to represent the software in a graphical format, that is it provides a standard way to visualize how a system is designed. Good UML design is followed for a better and precise software output. The UML provides different constructs for specifying, visualizing, constructing and documenting the artifacts of software systems. UML diagrams are used to better understand the system, to maintain the document information about the system and emphasizes on the roles, actors, actions, actions and process. The UML diagrams considered are:

- Use-case Diagram
- Class Diagram
- Sequence Diagram

**Relationships:** The relationships among different entities in the following diagrams are given below and in the figure 10:

- **Association** - This relationship tells how two entities interact with each other.
- **Dependency** - An entity is dependent on another if the change of that is reflecting the other.
- **Aggregation** - It is a special kind of association which exhibits part-of relationship.
- **Generalization** - This relationship describes the parent- child relationship among different entities.
- **Realization** - It is a kind of relationship in which one thing specifies the behavior or a responsibility to be carried out, and the other thing carries out that behavior.

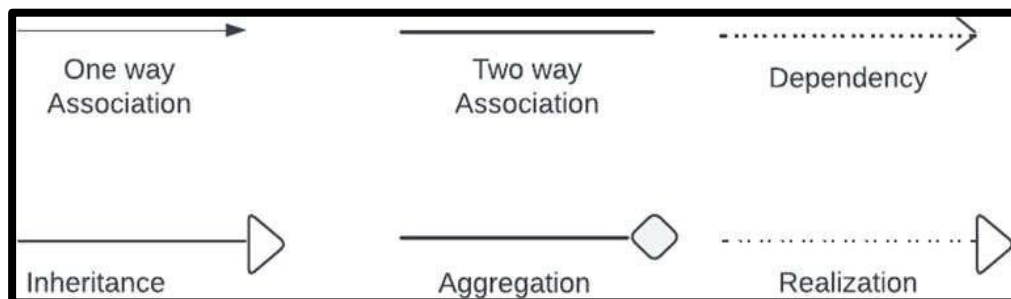


Fig. 10. Relationships in UML Diagrams

## **4.2 UML DIAGRAMS**

### **4.2.1 Use-Case Diagrams**

Use case diagrams capture the behavior of a system and help to emphasize the requirements of the system. It describes the high-level functionalities of the system. It consists of the following artifacts,

- Actor
- Use case
- Secondary Actor and Use case Boundary

#### **Actor**

A role of a user that interacts with the system you're modelling is represented by an actor. The primary actor in this project is User. These users are again classified as depressed users and non-depressed users.

#### **Use-Case**

A use case is a function that a system does to help the user achieve their goal. They are 9 use-cases which has their own functionality that can be useful in accomplishing the goals.

#### **Secondary Actor**

Actors who are not directly interacting with the system are secondary actors and are placed on the right side of the use case boundary.

Actors who are not directly interacting with the system are secondary actors and are placed on the right side of the use case boundary. Figure shows that the use case diagram consists of different use cases, actors, and secondary actors.

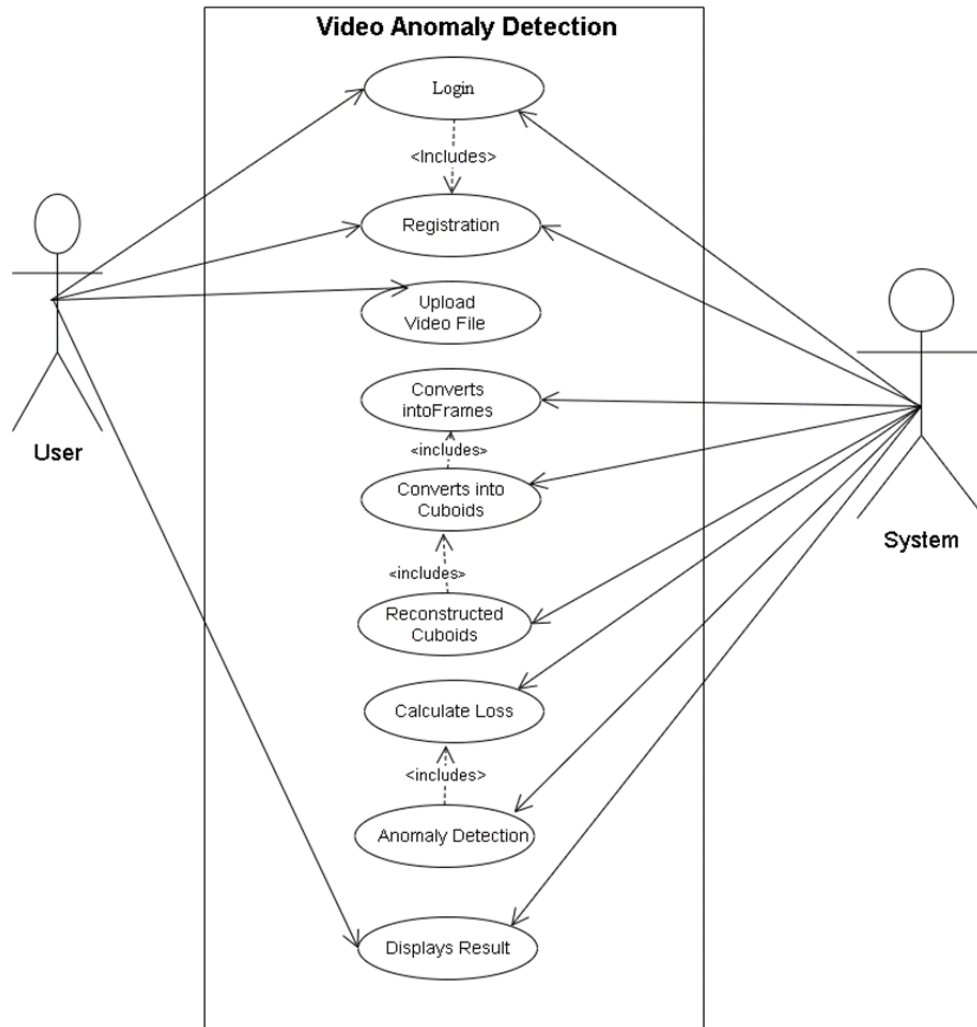


Fig. 11. Use case Diagram

From the Fig. 11, the use case diagram outlines the functionalities and interactions of the proposed system involving two actors: the User and the System.

1. User: This actor interacts with the system by:
  - Logging in or registering to access the system.
  - Uploading video files for analysis.
  - Viewing the results of the anomaly detection process.
2. System: This automated actor performs various tasks related to video processing and anomaly detection, including:
  - Converting uploaded videos into individual frames.
  - Further processing frames into cuboids, a data structure for analysis.



- Reconstructing cuboids, possibly to identify normal patterns.
- Calculating loss to measure deviation from normal patterns.
- Conducting anomaly detection based on the calculated loss.
- Presenting the final anomaly detection results to the User.

### 4.2.2 Class Diagram

Class diagram is a static overview of the system used to highlight the important aspects as well as executables in the system. These are the only diagrams which can be mapped with the object-oriented systems. These diagrams show the responsibilities of a class and relationships among different classes in the system.

#### Class

The class represents similar objects and consists of name, responsibilities and attributes. The classes considered in the project are User, STAE, Frames, Cuboid Detection. The classes are shown diagrammatically in the figure 14. The name, attributes, relationships, responsibilities are explained in detail in the Table 5.

Table 6. Class Diagram Artifacts

| S. no. | Class Name | Attributes                             | Responsibilities                                   | Relationships                 |
|--------|------------|--|--|-------------------------------|
| 1      | User       | Name, email, age, sex                  | Login(), Upload_Video(), logout()                  | Association with System       |
| 2      | System     | UploadedVideo                          | display_result(), display_costplot()               | Association with STAE, Frames |
| 3      | STAE       | frames, data                           | preprocess(), calculate_cost(), train(), predict() | NA                            |
| 4      | Frames     | filepath, frames_per_sec, source video | extract_frames()                                   | NA                            |
| 5      | Cuboids    | filepath, processed_frames             | extract_cuboids()                                  | Dependency with frames        |
| 6      | Detection  | output_file, return value              | calculate_loss(), preview_videos(), plot_graph()   | Dependency with cuboids       |

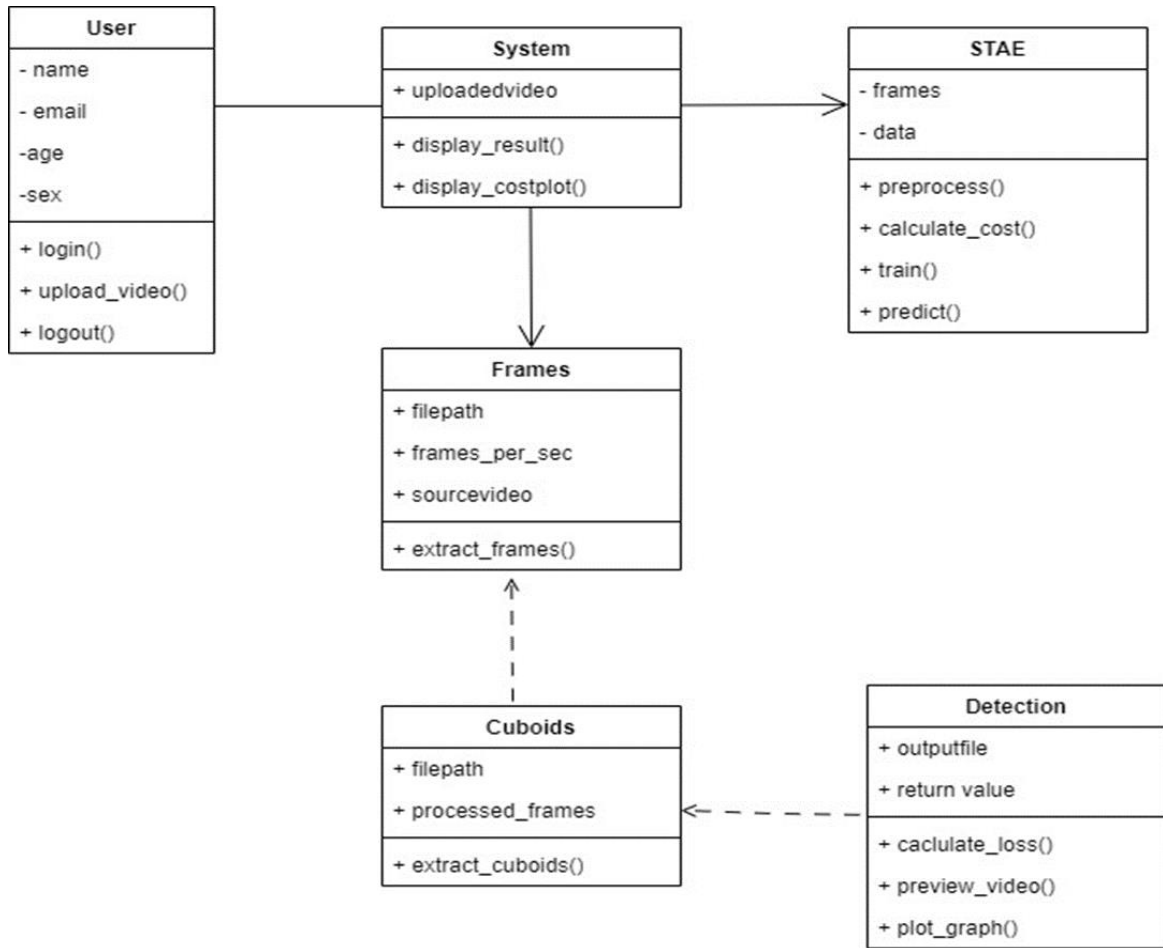


Fig. 12. Class Diagram

From the Fig. 12, the system consists of several classes, each with specific attributes and responsibilities. The "User" class represents users with attributes such as name, email, age, and sex. It provides methods for user authentication, video uploading, and logging out. The "System" class manages uploaded videos and offers functionalities to display results and cost plots. The "STAE" class, short for spatiotemporal autoencoder, handles frames and data for anomaly detection. It preprocesses data, calculates costs, trains the model, and makes predictions. The "Frames" class deals with video frames, storing attributes like file path, frames per second, and the source video. It provides a method to extract frames from videos. The "Cuboids" class, dependent on frames, processes frames into cuboids, maintaining attributes like file path and processed frames. It includes a method for extracting cuboids. Finally, the "Detection" class manages detection operations, including

calculating loss, previewing videos, and plotting graphs, with attributes for output file and return value. This structured class diagram delineates the roles and relationships within the system.

### **4.2.3 Sequence Diagram**

The sequence diagram, also known as an event diagram, depicts the flow of messages through the system. It aids in the visualization of a variety of dynamic scenarios. It depicts communication between any two lifelines as a time-ordered series of events, as if these lifelines were present at the same moment. The message flow is represented by a vertical dotted line that extends across the bottom of the page in UML, whereas the lifeline is represented by a vertical bar. It encompasses both iterations and branching. A sequence diagram in figure 15 simply displays the order in which things interact, or the order in which these interactions occur. A sequence diagram can also be referred to as an event diagram or an event scenario. Sequence diagrams show how and in what order the components of a system work together.

Notations of Sequence Diagram are,

#### **Lifeline**

A lifeline represents an individual participant in the sequence diagram. It is at the very top of the diagram. The lifeline for different participants starts at different times.

#### **Actor**

An actor is a character who interacts with the subject and plays a part. It isn't covered by the system's capabilities. It depicts a role in which human users interact with external devices or subjects. Here the user and admin are the actors.

#### **Message**

Message denotes the interactions between the messages. They are in the sequential order in the timeline. Here there are 8 messages which are both call and return messages with respective to the participants in the system.

#### **Call message**

By defining a specific communication between the interaction's lifelines, it shows that the target lifeline has invoked an operation.

## Return Message

It specifies a specific communication between the interaction lifelines, which reflect the flow of data from the receiver of the associated caller message.

The sequence of messages is shown in the figure 15. The system here represents the GUI that the user interacts with and also predicts that the user is depressed or not.

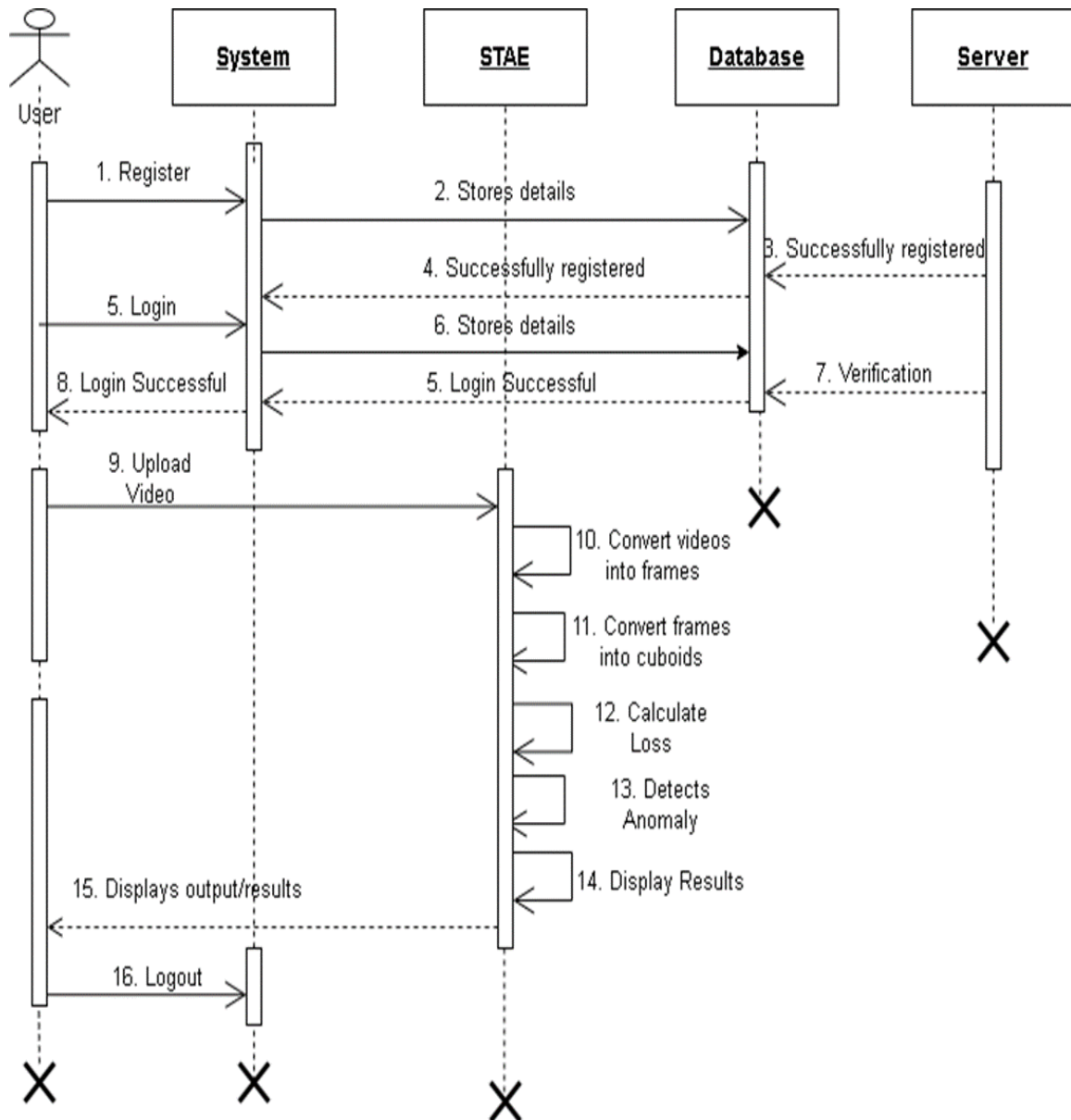


Fig. 13. Sequence Diagram

From Fig. 13, sequence diagram illustrates the interactions between different components in a system that involves video upload and anomaly detection features. The actors involved are the User, System, STAE, Database, and Server. Here's a summary of the interactions and processes:

1. The User registers with the system, and their details are stored in the STAE and the Database.
2. The System acknowledges successful registration, and the Server verifies the user.
3. The User logs in, and the STAE confirms the login to both the User and the Database.
4. The User uploads a video.
5. The Server processes the video by converting it into frames, then into cuboids, calculates loss, detects anomalies, and displays the results.
6. After receiving the output, the User logs out of the system.

This sequence diagram provides a high-level overview of the interactions and processes involved in the system, without detailing the internal workings of each component. Its purpose is to understand the flow of actions and the sequence of events within the software system.

## **4.3 MODULE DESIGN**

### **4.2.4 System Module**

The system comprises five key modules: data processing, Anomaly Detection, Integration, and user interface. The Data Preprocessing Module prepares raw video data by extracting frames, re segmenting them and segmenting into temporal cuboids. The Spatio-Temporal Autoencoder Module learns spatiotemporal patterns in video data through an autoencoder architecture, facilitating anomaly detection by reconstructing frames. The Anomaly Detection Module analyzes reconstruction errors to classify video segments as normal or integration module-integration module orchestrates interactions between modules to ensure seamless operation. Finally, the User Interface Module offers a user-friendly interface for inputting data, visualizing results, and adjusting parameters.

### **1. Data Preprocessing Module:**

This module handles the preprocessing of raw video data to extract meaningful features and prepare it for input to the anomaly detection model. It may include tasks such as frame extraction, resizing, normalization, and temporal segmentation (cuboidal conversion). Components are:

- Frame extraction: Extracting individual frames from video streams.
- Spatial resizing: resizing frames to a uniform spatial resolution.
- Temporal segmentation: dividing video sequences into temporal segments or cuboids.

### **1. Spatio-Temporal Autoencoder Module:**

This module consists of the spatio-temporal autoencoder architecture responsible for learning and encoding the spatio-temporal patterns in video data. It reconstructs normal video frames and identifies anomalies based on reconstruction errors. Components are:

- Encoder: encodes input video frames into latent representations.
- Decoder: Reconstructs video frames from latent representations.
- Loss function: Measures the reconstruction error between input and reconstructed frames.

### **2. Anomaly Detection Module:**

This module detects anomalies in video data based on the reconstruction errors generated by the spatio-temporal autoencoder. It applies anomaly detection algorithms to classify video segments as normal or anomalous based on predefined thresholds or learned criteria. Components are:

- Thresholding: setting thresholds for reconstruction errors to classify anomalies.
- Classification: classifying video segments as normal or anomalous.
- Post-processing: Refining anomaly detection results and filtering false positives classifying errors.

### **3. Integration Module:**

This module integrates the various components of the system, including data preprocessing, spatio-temporal autoencoders, and anomaly detection modules. It

manages the flow of data between modules and coordinates their interactions to facilitate seamless operation. Components are:

- Data flow management: orchestrating the flow of data between modules.
- Model integration: integrating the spatio-temporal autoencoder with the anomaly detection module.
- System configuration: Configuring parameters and settings for optimal system performance.

#### **4. User Interface Module:**

This module provides a user-friendly interface for interacting with the system, allowing users to input video data, visualize anomaly detection results, and adjust system parameters as needed. Components are:

- Input interface: uploading video data or connecting to live video streams.
- Visualization tools: Displaying anomaly detection results and reconstructed video frames.

### **4.3 CONCLUSION**

The UML diagrams are used in order to plot the functionalities of the system in advance. Failing to draw architecture of the system makes the system vulnerable to attacks and harms and these diagrams make the system vivid and perfect for implementation. The division of the project into modules not only makes it easier for understanding the purpose but also in implementation. Implementation of these divided modules is easier and also as the modules are independent to some level debugging and resolving errors also becomes easier. The visual representation of the modules and sub modules help in understanding of their working and also help in integration of the sub modules.

## 5. IMPLEMENTATION AND RESULTS

### 5.1 INTRODUCTION

The project's implementation stage is when the theoretical design is translated into a workable system. As a result, it can be seen as the most crucial stage in ensuring the success of a new system and giving the user confidence that the system will work and be effective. The implementation step entails meticulous planning, research of the existing system and its implementation limitations, designing of changeover methods, and evaluation of changeover methods.

### 5.2 EXPLANATION OF KEY FUNCTIONS

1. **mean\_squared\_loss(x1, x2):** This function calculates the mean squared loss between two input arrays x1 and x2. It computes the element-wise squared difference between the arrays, sums them up, takes the square root of the sum, and then divides it by the total number of elements in the arrays to get the mean distance. This function is used to evaluate the loss between input and output data in the anomaly detection models.
2. **generate\_sequences(step, frames\_list, sequence\_size):** This function generates sequences of frames from a list of frames. It takes three arguments: step (the step size for generating sequences), frames\_list (the list of frames), and sequence\_size (the size of each sequence). It generates sequences by iterating over the frames list with a given step size and sequence size.
3. **extract\_training\_data(dataset\_path):** This function extracts training data from a specified dataset path. It iterates over the folders in the dataset path, reads TIFF images, resizes them to (256, 256), normalizes the pixel values, and generates sequences of 10-frame clips from the images with different strides. These sequences are used for training anomaly detection models.
4. **load\_single\_test\_data(test\_path):** This function loads single test data from a specified test path. It reads TIFF images, resizes them to (256, 256), normalizes the pixel values, and stores them in an array.
5. **evaluate(model, test\_path):** This function evaluates the anomaly detection model by computing the reconstruction cost of sequences from the test data. It applies the sliding window technique to get sequences, predicts their reconstructions using the



model, calculates the reconstruction cost, and plots the normalized reconstruction cost over frames.

#### **Model architecture functions:**

1. Conv2D, Conv2DTranspose: Convolutional layers for encoding and decoding.
2. ConvLSTM2D: Convolutional LSTM layer for processing spatiotemporal data.
3. LayerNormalization: Layer normalization layer to normalize the activations of the previous layer.
4. TimeDistributed: Wrapper layer that applies a layer to every temporal slice of an input.
5. Sequential: Keras model type for linear stack of layers.
6. load\_model: Function to load a pre-trained model.

These functions collectively handle tasks such as data processing, model training, evaluation, and defining model architectures for anomaly detection in videos.

## **5.3 MODEL BUILDING**

### **5.3.1 Hybrid Model of CNN-LSTM**

Model: "model\_1"

| Layer (type)                          | Output Shape           | Param # |
|---------------------------------------|------------------------|---------|
| input_2 (InputLayer)                  | [(None, 36, 64, 3, 1)] | 0       |
| conv3d_2 (Conv3D)                     | (None, 16, 30, 3, 128) | 3328    |
| conv3d_3 (Conv3D)                     | (None, 14, 28, 3, 64)  | 73792   |
| conv_lstm2d_3 (ConvLSTM2D)            | (None, 14, 28, 3, 64)  | 295168  |
| conv_lstm2d_4 (ConvLSTM2D)            | (None, 14, 28, 3, 32)  | 110720  |
| conv_lstm2d_5 (ConvLSTM2D)            | (None, 14, 28, 3, 64)  | 221440  |
| conv3d_transpose_2 (Conv3D Transpose) | (None, 16, 30, 3, 128) | 73856   |
| conv3d_transpose_3 (Conv3D Transpose) | (None, 36, 64, 3, 1)   | 3201    |

=====  
Total params: 781505 (2.98 MB)  
Trainable params: 781505 (2.98 MB)  
Non-trainable params: 0 (0.00 Byte)

Fig. 14. AutoEncoder Summary

The model is a convolutional autoencoder designed for spatio-temporal anomaly detection, which takes 5-dimensional input data (batch size, height, width, temporal

depth, channels) and outputs reconstructed data of the same shape. Here's an elaborated breakdown of the model summary:

**1. Input Layer (Input Layer):**

- Shape: (None, 36, 64, 3, 1)
- Explanation: This layer defines the input shape for the model, indicating that it expects input data with a shape of (batch size, 36, 64, 3, 1), where:
- Batch size can vary (None).
- Height is 36 pixels.
- Width is 64 pixels.
- Temporal depth is 3 frames.
- Channels is 1 (grayscale).

**2. Convolutional Layer 1 (Conv3D):**

- Output Shape: (None, 16, 30, 3, 128)
- Params: 3,328
- Explanation: This convolutional layer applies 3D convolutional filters to the input data, resulting in feature maps with a depth of 128.
- Convolutional Layer 2 (Conv3D):
- Output Shape: (None, 14, 28, 3, 64)
- Params: 73,792
- Explanation: Another convolutional layer is applied, reducing the spatial dimensions and increasing the number of feature maps to 64.

**3. ConvLSTM2D Layer 1 (ConvLSTM2D):**

- Output Shape: (None, 14, 28, 3, 64)
- Params: 295,168
- Explanation: This layer is a Convolutional LSTM, which integrates convolutional operations into the LSTM architecture. It operates on the spatio-temporal data, preserving the spatial dimensions while modeling temporal dependencies.

**4. ConvLSTM2D Layer 2 (ConvLSTM2D):**

- Output Shape: (None, 14, 28, 3, 32)
- Params: 110,720
- Explanation: Another ConvLSTM2D layer further processes the data, reducing the number of feature maps to 32.

### **5. ConvLSTM2D Layer 3 (ConvLSTM2D):**

- Output Shape: (None, 14, 28, 3, 64)
- Params: 221,440
- Explanation: This ConvLSTM2D layer increases the number of feature maps back to 64, maintaining spatial dimensions while capturing additional temporal dependencies.

### **6. Convolutional Transpose Layer 1 (Conv3DTranspose):**

- Output Shape: (None, 16, 30, 3, 128)
- Params: 73,856
- Explanation: Convolutional transpose (deconvolution) layer to upsample the feature maps to the original dimensions.

### **7. Convolutional Transpose Layer 2 (Conv3DTranspose):**

- Output Shape: (None, 36, 64, 3, 1)
- Params: 3,201
- Explanation: Final convolutional transpose layer reconstructs the input data to match the original input shape.

Overall, the model comprises a combination of convolutional and ConvLSTM layers for feature extraction and temporal modeling, followed by convolutional transpose layers for reconstruction. The total number of trainable parameters in the model is 781,505.

## **5.4 INTEGRATION OF FRONTEND WITH BACKEND CODE**

### **Front-end Design: Streamlit**

Streamlit is a Python library that enables you to create interactive web applications effortlessly. It's particularly popular among data scientists and machine learning engineers because it allows them to quickly prototype and share their work without needing expertise in web development. With Streamlit, you can turn your Python scripts into web apps with just a few lines of code, making it easy to visualize data, build dashboards, and share insights with others. It's designed to be intuitive and user-friendly, so you can focus on your data and analysis rather than the intricacies of web development.

## **BACKEND**

### **JSON**

JSON (JavaScript Object Notation) is a lightweight data interchange format widely used in web development and other applications.

JSON is a text-based data format derived from JavaScript object literal syntax. It serves as a standard for data interchange due to its simplicity, readability, and versatility. JSON data is platform-independent, making it suitable for communication between different programming languages and systems.

Usage:

JSON finds extensive usage in various domains:

**Web APIs:** Many web services utilize JSON for data exchange between servers and clients. APIs often return JSON-formatted data in response to client requests.

**Configuration Files:** JSON is employed for storing configuration settings in applications. Its hierarchical structure allows for organized and readable configurations.

**Data Storage:** JSON serves as a data format for storing semi-structured data in databases, NoSQL databases, or files.

Advantages:

JSON offers several advantages:

1. **Human Readability:** JSON data is easy for both humans and machines to read and write.
2. **Compactness:** JSON's concise syntax results in smaller payloads, making it efficient for data transmission over networks.
3. **Language Independence:** JSON is supported by numerous programming languages, ensuring compatibility across diverse platforms.
4. **Versatility:** JSON accommodates complex data structures, including nested objects and arrays, facilitating representation of hierarchical data.

### **DATA EXTRACTION**

The following figure 24 shows the hierarchy of the code in our project.

#### **Root Directory:**

- **Source Code Files:** Contains the source code files of the project, organized into directories based on their functionality or module.

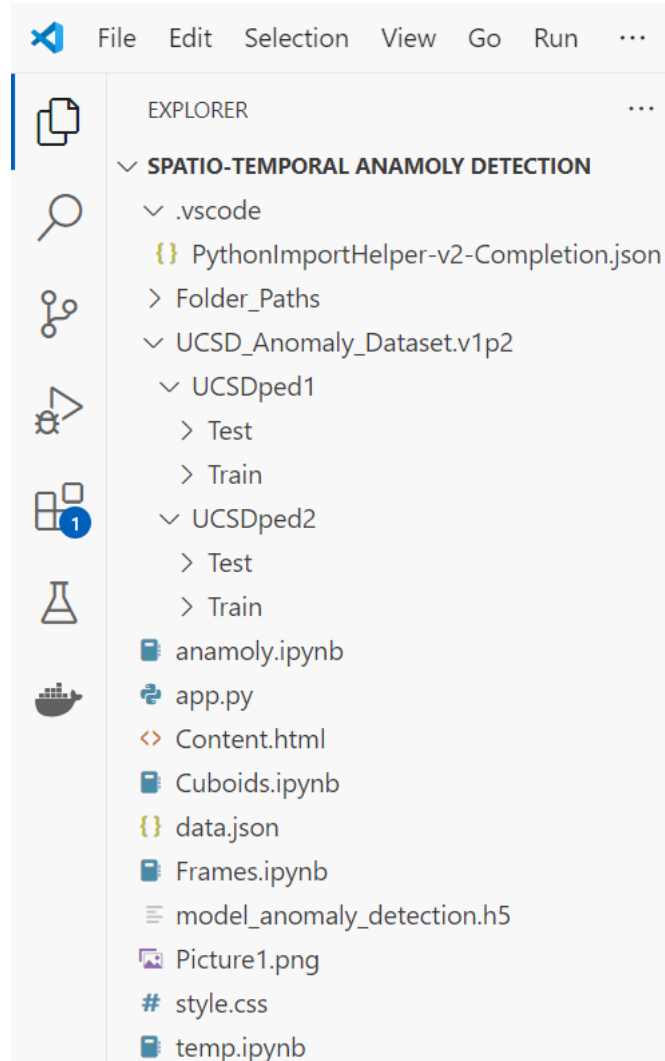


Fig. 15. Hierarchy of code files in our project

- Configuration Files: Contains configuration files for various aspects of the project, such as build configurations, environment settings, or deployment configurations.
- Documentation: Contains project documentation, including README files, user manuals, or technical documentation.
- Tests: Contains unit tests, integration tests, or other test suites for validating the functionality of the project.
- Dependencies and Package Management: Contains files related to dependencies and package management, such as package.json, requirements.txt, or Gemfile.
- Build and Deployment Scripts: Contains scripts for building, packaging, or deploying the project.
- Version Control: Contains version control-related files.

### **Subdirectories (Source Code Files):**

- **Modules or Components:** Contains directories representing modules, components, or features of the application.  
Example: user/, auth/, dashboard/
- **Views or Templates:** Contains files or directories related to views, templates, or UI components.  
Example: views/, templates/, ui/
- **Controllers or Handlers:** Contains files responsible for handling requests, business logic, or application flow control.  
Example: controllers/, handlers/, controllers/
- **Models or Data Access:** Contains files representing data models, database schemas, or data access logic.  
Example: models/, data/, db/
- **Utilities or Helpers:** Contains utility functions, helper classes, or reusable code snippets.  
Example: utils/, helpers/, common/
- **Configuration Files:** Contains configuration files specific to the module or component.  
Example: config/, settings/, conf/
- **Tests:** Contains unit tests or integration tests specific to the module or component.

## **Frontend Code**

### **1. "Project Information" Page:**

Functions Used:

#### **main(json\_file\_path)**

Initializes the web application and defines different pages. Controls the flow of the application based on user interactions. Renders the "Project Information" page content.

#### **Explanation:**

The main function is responsible for initializing the web application and defining the different pages users can navigate through. In the "Project Information" page, it displays detailed information about the project, including its features, usage, and team members. It also provides options for users to sign up or log in to access the dashboard. This function primarily handles the rendering of the "Project Information" page content.

## **2. "Signup/Login" Page:**

Functions Used:

### **(a). main(json\_file\_path)**

Initializes the web application and defines different pages. Controls the flow of the application based on user interactions. Renders the signup/login page content.

### **(b). login(json\_file\_path)**

Renders the login form. Validates user credentials and logs in the user if valid.

### **(c). signup(json\_file\_path)**

Renders the signup form. Validates user input and creates a new account if valid.

### **(d). check\_login(username, password, json\_file\_path)**

Checks user login credentials against the data stored in the JSON file.

### **(e). create\_account(user\_info, json\_file\_path)**

Creates a new user account and adds it to the JSON database.

### **(f). email\_exists(email, json\_file\_path)**

Checks if an email already exists in the database.

### **Explanation:**

The main function initializes the web application and defines the different pages users can navigate through. In the "Signup/Login" page, it renders the signup/login form based on user selection. The login function handles the rendering of the login form and validates user credentials against the stored data.

## **3. "Dashboard" Page:**

Functions Used:

### **(a). main(json\_file\_path)**

Initializes the web application and defines different pages. Controls the flow of the application based on user interactions. Renders the dashboard page content.

### **(b). render\_dashboard(user\_info, json\_file\_path)**

Renders the dashboard content. Displays user information retrieved from the database.

### **Explanation:**

The main function handles the initialization of the web application and defines the different pages users can navigate through. In the "Dashboard" page, it renders the user dashboard after successful login. The render\_dashboard function displays user information retrieved from the database on the dashboard page.

#### **4. "Upload Files" Page:**

Functions Used:

##### **(a). main(json\_file\_path)**

Initializes the web application and defines different pages. Controls the flow of the application based on user interactions. Renders the "Upload Files" page content.

##### **(b). evaluate(test\_path)**

Evaluates the uploaded files using a pre-trained deep learning model and displays the results.

##### **(c). load\_single\_test\_data(test\_path)**

Loads test data from the specified path.

#### **Explanation:**

The main function initializes the web application and defines the different pages users can navigate through. In the "Upload Files" page, it renders the file uploader interface and handles file evaluation. The evaluate function evaluates the uploaded files using a pre-trained deep learning model for anomaly detection and displays the results. The load\_single\_test\_data function loads test data from the specified path, which is used by the evaluate function for processing.

#### **Backend code**

##### **Data.json**

```
{
  "users": [
    {
      "name": "User",
      "email": "user@mail.com",
      "age": 22,
      "sex": "Male",
      "password": "123"
    },
    {
      "name": "Shreyansh",
      "email": "user@gmail.com",
      "age": 22,
      "sex": "Male",
      "password": "123"
    }
  ]
}
```



```

    },

    {
        "name": "Pallavi",
        "email": "pallavialanka@gmail.com",
        "age": 21,
        "sex": "Female",
        "password": "user123"
    },
    {
        "name": "JOHN",
        "email": "johns@bmail.com",
        "age": 54,
        "sex": "Male",
        "password": "123123"
    }
]
}

```

### **Frames.ipynb**

```

#from google.colab import drive
#drive.mount('/content/drive')
file_path = r"C:\Users\Hp\Desktop\Major Project\MajorProject\Avenue_Dataset"
pip install scikit-image
from PIL import Image
from keras.preprocessing.image import img_to_array,load_img
import glob
import os
from skimage import data, color
from skimage.transform import rescale, resize, downscale_local_mean
import cv2
from skimage import data, color
from skimage.transform import rescale, resize, downscale_local_mean
from keras.models import load_model

```

```

import sys
#import skimage.viewer
from skimage import io
from keras.layers import Conv3D,ConvLSTM2D,Conv3DTranspose
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint, EarlyStopping
import numpy as np
import argparse
#from tensorflow.python import keras.models.Sequential
#from tensorflow.python import keras.layers.Dense
from tensorflow.python.keras.layers import Input, Dense
from tensorflow.python.keras.models import Sequential
imagestore=[]
"""

from matplotlib import pyplot as plt
def create_dir(path):
    if not os.path.exists(path):
        os.makedirs(path)
def remove_old_images(path):
    filelist = glob.glob(os.path.join(path, "*.png"))
    for f in filelist:
        os.remove(f)
from google.colab.patches import cv2_imshow
def store(image_path):
    fig = plt.figure(figsize=(10, 7))
    rows = 1
    columns = 3
    imfg=cv2.imread(image_path)
    fig.add_subplot(rows, columns, 1)
    plt.imshow(imfg)
    plt.axis('off')
    plt.title("Original frame")
    img=load_img(image_path)

```

```

img=img_to_array(img)
img=resize(img,(227,227,3))
gray=0.2989*img[:, :,0]+0.5870*img[:, :,1]+0.1140*img[:, :,2]
grayy = gray
cv2.getGaussianKernel(9,9)
gry= cv2.GaussianBlur(gray,(5,5),0)
imagestore.append(gray)
fig.add_subplot(rows, columns, 2)
plt.imshow(grayy, cmap='gray')
plt.axis('off')
plt.title("Gray scaled")
fig.add_subplot(rows, columns, 3)
plt.imshow(gry, cmap='gray')
plt.axis('off')
plt.title("Gaussian blur")
def store(image_path):
    # Read the original image
    original_frame = cv2.imread(image_path)
    # Convert the original image to grayscale
    gray = cv2.cvtColor(original_frame, cv2.COLOR_BGR2GRAY)
    # Apply Gaussian blur to the grayscale image
    gray_blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    # Display the original frame
    cv2.imshow("Original frame", original_frame)
    cv2.waitKey(0)
    # Display the grayscale frame
    cv2.imshow("Gray scaled", gray)
    cv2.waitKey(0)
    # Display the blurred frame
    cv2.imshow("Gaussian blur", gray_blurred)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
def load_model():
    model=Sequential()

```

```

model.add(Conv3D(filters=128,kernel_size=(11,11,1),strides=(4,4,1),padding='valid',
input_shape=(227,227,10,1),activation='tanh'))
model.add(Conv3D(filters=64,kernel_size=(5,5,1),strides=(2,2,1),padding='valid',activation='tanh'))
model.add(ConvLSTM2D(filters=64,kernel_size=(3,3),strides=1,padding='same',dropout=0.4,recurrent_dropout=0.3,return_sequences=True))
model.add(ConvLSTM2D(filters=32,kernel_size=(3,3),strides=1,padding='same',dropout=0.3,return_sequences=True))
model.add(ConvLSTM2D(filters=64,kernel_size=(3,3),strides=1,return_sequences=True, padding='same',dropout=0.5))
model.add(Conv3DTranspose(filters=128,kernel_size=(5,5,1),strides=(2,2,1),padding='valid',activation='tanh'))
model.add(Conv3DTranspose(filters=1,kernel_size=(11,11,1),strides=(4,4,1),padding='valid',activation='tanh'))
model.compile(optimizer='adam',loss='mean_squared_error',metrics=['accuracy'])
return model

def mean_squared_loss(x1,x2):
    diff=x1-x2
    a,b,c,d,e=diff.shape
    n_samples=a*b*c*d*e
    sq_diff=diff**2
    Sum=sq_diff.sum()
    dist=np.sqrt(Sum)
    mean_dist=dist/n_samples
    return mean_dist

threshold=0.0008
'''

from matplotlib import pyplot as plt
def create_dir(path):
    if not os.path.exists(path):
        os.makedirs(path)
def remove_old_images(path):
    filelist = glob.glob(os.path.join(path, "*.png"))
    for f in filelist:

```

```

        os.remove(f)
def store(image_path):
    # Read the original image
    original_frame = cv2.imread(image_path)
    # Convert the original image to grayscale
    gray = cv2.cvtColor(original_frame, cv2.COLOR_BGR2GRAY)
    # Apply Gaussian blur to the grayscale image
    gray_blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    # Display the original frame
    cv2.imshow("Original frame", original_frame)
    cv2.waitKey(0)
    # Display the grayscale frame
    cv2.imshow("Gray scaled", gray)
    cv2.waitKey(0)
    # Display the blurred frame
    cv2.imshow("Gaussian blur", gray_blurred)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
def mean_squared_loss(x1,x2):
    diff=x1-x2
    a,b,c,d,e=diff.shape
    n_samples=a*b*c*d*e
    sq_diff=diff**2
    Sum=sq_diff.sum()
    dist=np.sqrt(Sum)
    mean_dist=dist/n_samples
    return mean_dist
threshold=0.0008
X_test=ex[:, :, 160:frames]
X_test=X_test.reshape(-1,227,227,10)
X_test=np.expand_dims(X_test,axis=4)
Y_test=X_test.copy()
if __name__=="__main__":
    model=load_model()

```

```

callback_save = ModelCheckpoint("/content/drive/MyDrive/Anomaly-detection-in-
crowded-scenes-master/AnomalyDetector.h5",      monitor="mean_squared_error",
save_best_only=True)

callback_early_stopping = EarlyStopping(monitor='val_loss', patience=3)

print('Model has been loaded')

history=model.fit(X_train,    Y_train,    batch_size,validation_data=(X_test,
Y_test),steps_per_epoch=16,epochs=40,verbose=1)

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

# Train and validation accuracy
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
plt.title('Training and Validation accuracy')
plt.legend()
plt.figure()

# Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()

print("[INFO] Calculating model accuracy")
scores = model.evaluate(X_train, Y_train)
print(f"Accuracy: {scores[1]*100}")
print("Evaluate on test data")
results = model.evaluate(X_test, Y_test)
print("test loss, test acc:", results)

video_source_path='/content/drive/MyDrive/Anomaly-detection-in-crowded-scenes-
master/training_vid'

fps=5
imagestore=[]

```

```

gg=0
op=0
n=0
i=1
losss=[]
model=load_model()
videos=os.listdir(video_source_path)
i=0
for video in videos:
    ji=[]
    create_dir(video_source_path+'/frames')
    framepath=video_source_path+'/frames'
    imgpat=video_source_path+'/'+video
    imagedump=[]
    flag5=0
    os.system( 'ffmpeg -i { }/{ } -r
1/{ } { }/frames/%03d.jpg'.format(video_source_path,video,fps,video_source_path))
    images=os.listdir(framepath)
    #/content/drive/MyDrive/Anomaly-detection-in-crowded-scenes-
master/testing_vid/03.avi
    for image in images:
        image_path=framepath+ '/' + image
        imgf = cv2.imread(image_path)
        image_path=load_img(image_path)
        image_path= img_to_array(image_path)
        frame=resize(image_path,(227,227,3))
        #input
        gray=0.2989*frame[:, :,0]+0.5870*frame[:, :,1]+0.1140*frame[:, :,2]
        gray=(gray-gray.mean())/gray.std()
        grayy = gray
        cv2.getGaussianKernel(9,9)
        gry= cv2.GaussianBlur(gray,(5,5),0)
        imagedump = gray
        imagedump=np.array(imagedump)

```

```

imagedump.resize(227,227,10)
imagedump=np.expand_dims(imagedump,axis=0)
imagedump=np.expand_dims(imagedump,axis=4)
output=model.predict(imagedump)
loss=mean_squared_loss(imagedump,output)
ji.append(loss)
losss.append(ji)
remove_old_images(video_source_path+'/frames')
#plotting to find threshold
v=0
for i in range(len(losss)):
    y=[]
    for j in range(len(losss[v])):
        y.append(v+1)
    plt.plot(y, lossss[v], '*', color='red', label='Loss')
    v=v+1
plt.title('FINDING Threshold')
threshold=0.0004405292
video_source_path='/content/drive/MyDrive/Anomaly-detection-in-crowded-scenes-
master/testing_vid'
fps=5
imagestore=[]
gg=0
op=0
n=0
i=1
losss=[]
print('Loading model')
model=load_model()
print('Model loaded')
videos=os.listdir(video_source_path)
i=0
for video in videos:
    ji=[]

```



```

create_dir(video_source_path+'/frames')
framepath=video_source_path+'/frames'
imgpat=video_source_path+'/'+video
imagedump=[]
flag5=0
os.system( 'ffmpeg -i { }/{ } -r
1/{ } { }/frames/%03d.jpg'.format(video_source_path,video,fps,video_source_path))
images=os.listdir(framepath)
#/content/drive/MyDrive/Anomaly-detection-in-crowded-scenes-
master/testing_vid/03.avi
for image in images:
    fig = plt.figure(figsize=(10, 7))
    rows = 1
    columns = 4
    image_path=framepath+ '/' + image
    imgf = cv2.imread(image_path)
    image_path=load_img(image_path)
    image_path= img_to_array(image_path)
    frame=resize(image_path,(227,227,3))
    fig.add_subplot(rows, columns, 1)
    plt.imshow(imgf)
    plt.axis('off')
    plt.title("Original frame")
    #input
    gray=0.2989*frame[:, :,0]+0.5870*frame[:, :,1]+0.1140*frame[:, :,2]
    gray=(gray-gray.mean())/gray.std()
    grayy = gray
    fig.add_subplot(rows, columns, 2)
    plt.imshow(grayy, cmap=plt.cm.gray)
    plt.axis('off')
    plt.title("Gray scaled")
    cv2.getGaussianKernel(9,9)
    gry= cv2.GaussianBlur(gray,(5,5),0)
    fig.add_subplot(rows, columns, 3)

```

```

plt.imshow(gry, cmap=plt.cm.gray)
plt.axis('off')
plt.title("Gaussian blur")
imagedump = gray
imagedump=np.array(imagedump)
imagedump.resize(227,227,10)
imagedump=np.expand_dims(imagedump,axis=0)
imagedump=np.expand_dims(imagedump,axis=4)
output=model.predict(imagedump)
loss=mean_squared_loss(imagedump,output)
ji.append(loss)
if loss>threshold:
    fig.add_subplot(rows, columns, 4)
    plt.imshow(grayy, cmap=plt.cm.Red_s_r, interpolation='nearest')
    plt.axis('off')
    plt.title("Anomaly")
    flag5=1
    n=n+loss
else :
    fig.add_subplot(rows, columns, 4)
    plt.imshow(grayy, cmap=plt.cm.Green_s_r, interpolation='nearest')
    plt.axis('off')
    plt.title("Normal")
    flag5=flag5
    n=n+loss
    op=op+1
losss.append(ji)
#n=n/flag5
if flag5==1:
    print("this video is anamoly")
    print(gg+1)
else:
    print("video is normal")
    print(gg+1)

```

```

    gg=gg+1
    i=i+1
    remove_old_images(video_source_path+'/frames')
v=0
for i in range(len(losss)):
    y=[]
    for j in range(len(losss[v])):
        y.append(v+1)
    plt.plot(y, loss[s[v]], '*', color='red', label='Loss')
    v=v+1
plt.axhline(y=0.0004405292, color='blue', linestyle='-')
plt.title('FINDING ANOMALY OR NOT')
threshold = 0.00044052725
video_source_path='/content/drive/MyDrive/Anomaly-detection-in-crowded-scenes-master/randomvid'
fps=5
imagestore=[]
gg=0
op=0
n=0
i=1
losss=[]
print('Loading model')
model=load_model()
print('Model loaded')
videos=os.listdir(video_source_path)
i=0
for video in videos:
    ji=[]
    create_dir(video_source_path+'/frames')
    framepath=video_source_path+'/frames'
    imgpat=video_source_path+'/'+video
    imagedump=[]
    flag5=0

```

```

os.system( 'ffmpeg -i {}/{} -r
1/{} {}/frames/%03d.jpg'.format(video_source_path,video,fps,video_source_path))
images=os.listdir(framepath)
#/content/drive/MyDrive/Anomaly-detection-in-crowded-scenes-
master/testing_vid/03.avi
for image in images:
    fig = plt.figure(figsize=(10, 7))
    rows = 1
    columns = 4
    image_path=framepath+ '/' + image
    imgf = cv2.imread(image_path)
    image_path=load_img(image_path)
    image_path= img_to_array(image_path)
    frame=resize(image_path,(227,227,3))
    fig.add_subplot(rows, columns, 1)
    plt.imshow(imgf)
    plt.axis('off')
    plt.title("Original frame")
    #input
    gray=0.2989*frame[:, :,0]+0.5870*frame[:, :,1]+0.1140*frame[:, :,2]
    gray=(gray-gray.mean())/gray.std()
    grayy = gray
    fig.add_subplot(rows, columns, 2)
    plt.imshow(grayy, cmap=plt.cm.gray)
    plt.axis('off')
    plt.title("Gray scaled")
    cv2.getGaussianKernel(9,9)
    gry= cv2.GaussianBlur(gray,(5,5),0)
    fig.add_subplot(rows, columns, 3)
    plt.imshow(gry, cmap=plt.cm.gray)
    plt.axis('off')
    plt.title("Gaussian blur")
    imagedump = gray
    imagedump=np.array(imagedump)

```

```

imagedump.resize(227,227,10)
imagedump=np.expand_dims(imagedump,axis=0)
imagedump=np.expand_dims(imagedump,axis=4)
output=model.predict(imagedump)
loss=mean_squared_loss(imagedump,output)
ji.append(loss)
if loss>threshold:
    fig.add_subplot(rows, columns, 4)
    plt.imshow(gray, cmap=plt.cm.Red_r, interpolation='nearest')
    plt.axis('off')
    plt.title("Anomaly")
    flag5=1
    n=n+loss
else :
    fig.add_subplot(rows, columns, 4)
    plt.imshow(gray, cmap=plt.cm.Green_r, interpolation='nearest')
    plt.axis('off')
    plt.title("Normal")
    flag5=flag5
    n=n+loss
    op=op+1
losss.append(ji)
if(flag5==1):
    print("THE VIDEO IS ANOMALY")
remove_old_images(video_source_path+'/frames')

```

## Frontend code

### App.py

```

import json
import streamlit as st
from streamlit import session_state
import os
import numpy as np
from keras.models import load_model

```

```

import tensorflow as tf
import matplotlib.pyplot as plt
import streamlit.components.v1 as components
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import cv2
import os

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

session_state = st.session_state

if "user_index" not in st.session_state:
    st.session_state["user_index"] = 0

def load_single_test_data(test_path):
    size = 200

    test_data = np.zeros(shape=(size, 256, 256, 1))

    count = 0

    # Iterate through each file in the test folder
    for filename in sorted(os.listdir(test_path)):
        file_path = os.path.join(test_path, filename)

        # Check if the file is a TIFF image
        if filename.endswith(".tif"):
            # Open the image, resize it to (256, 256), and normalize pixel values
            img = Image.open(file_path).resize((256, 256))
            img = np.array(img, dtype=np.float32) / 256.0
            test_data[count, :, :, 0] = img
            count += 1

    return test_data

def evaluate(test_path):
    model = load_model("model_anomaly_detection.h5")

    test_data = load_single_test_data(test_path)

    sequence_size = 10

    sz = test_data.shape[0] - sequence_size

```

```

sequences = np.zeros((sz, sequence_size, 256, 256, 1))
# Apply the sliding window technique to get the sequences
for i in range(sz):
    clip = test_data[i : i + sequence_size, :, :, :]
    sequences[i] = clip
# Get the reconstruction cost of all the sequences
reconstructed_sequences = model.predict(sequences, batch_size=4)
sequences_reconstruction_cost = np.array(
    [
        np.linalg.norm(np.subtract(sequences[i], reconstructed_sequences[i]))
        for i in range(sz)
    ]
)
normalized_reconstruction_cost = (
    sequences_reconstruction_cost - np.min(sequences_reconstruction_cost)
) / np.max(sequences_reconstruction_cost)
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(normalized_reconstruction_cost, color="blue", linewidth=1)
ax.set_ylabel("Reconstruction Cost (t)")
ax.set_xlabel("Frame (t)")
ax.set_title("Reconstruction Cost vs Frame")
ax.grid(True, linestyle="--", alpha=0.6)
st.pyplot(fig)
def check_login(username, password, json_file_path="data.json"):
    try:
        with open(json_file_path, "r") as json_file:
            data = json.load(json_file)
        for user in data["users"]:
            if user["email"] == username and user["password"] == password:
                session_state["logged_in"] = True
                session_state["user_info"] = user
                st.success("Login successful!")
                return user
        st.error("Invalid credentials. Please try again.")

```

```

        return None
    except Exception as e:
        st.error(f"Error checking login: {e}")
        return None

def initialize_database(json_file_path="data.json"):
    try:
        # Check if JSON file exists
        if not os.path.exists(json_file_path):
            # Create an empty JSON structure
            data = {"users": []}
            with open(json_file_path, "w") as json_file:
                json.dump(data, json_file)
    except Exception as e:
        print(f"Error initializing database: {e}")

def login(json_file_path="data.json"):
    st.title("Login Page")
    username = st.text_input("Username:")
    password = st.text_input("Password:", type="password")
    login_button = st.button("Login")
    if login_button:
        user = check_login(username, password, json_file_path)
        if user is not None:
            session_state["logged_in"] = True
            session_state["user_info"] = user
        else:
            st.error("Invalid credentials. Please try again.")

def get_user_info(email, json_file_path="data.json"):
    try:
        with open(json_file_path, "r") as json_file:
            data = json.load(json_file)
            for user in data["users"]:
                if user["email"] == email:

```



```

        return user
    return None
except Exception as e:
    st.error(f"Error getting user information: {e}")
    return None
def render_dashboard(user_info, json_file_path="data.json"):
    try:
        st.title(f"Welcome to the Dashboard, {user_info['name']}!")
        st.subheader("User Information:")
        st.write(f"Name: {user_info['name']}")
        st.write(f"Sex: {user_info['sex']}")
        st.write(f"Age: {user_info['age']}")
    except Exception as e:
        st.error(f"Error rendering dashboard: {e}")
def signup(json_file_path="data.json"):
    st.title("Signup Page")
    with st.form("signup_form"):
        st.write("Fill in the details below to create an account:")
        name = st.text_input("Name:")
        email = st.text_input("Email:")
        age = st.number_input("Age:", min_value=0, max_value=120)
        sex = st.radio("Sex:", ("Male", "Female", "Other"))
        password = st.text_input("Password:", type="password")
        confirm_password = st.text_input("Confirm Password:", type="password")
        if st.form_submit_button("Signup"):
            user_info = {
                "name": name,
                "email": email,
                "age": age,
                "sex": sex,
                "password": password,
            }
            is_valid = validate_form(user_info, confirm_password, json_file_path)
            if is_valid:

```

```

        session_state["logged_in"] = True
        session_state["user_info"] = user_info
def validate_form(user_info, confirm_password, json_file_path="data.json"):
    name = user_info.get("name")
    email = user_info.get("email")
    age = user_info.get("age")
    sex = user_info.get("sex")
    password = user_info.get("password")
    # Check if any field is empty
    if not (name and email and age and sex and password and confirm_password):
        st.error("All fields are required. Please fill in all the details.")
        return False
    if not 0 <= age <= 120:
        st.error("Age must be between 0 and 120.")
        return False
    # Check if password and confirm_password match
    if password != confirm_password:
        st.error("Passwords do not match. Please try again.")
        return False
    # Check if the email already exists
    if email_exists(email, json_file_path):
        st.error("Email already exists. Please use a different email.")
        return False
    user = create_account(user_info, json_file_path)
    if user:
        st.success("Account created successfully! You can now login.")
        return True
    else:
        st.error("Error creating account. Please try again.")
        return False
def create_account(user_info, json_file_path="data.json"):
    try:
        # Check if the JSON file exists or is empty
        if not os.path.exists(json_file_path) or os.stat(json_file_path).st_size == 0:

```

```

        data = {"users": []}
    else:
        with open(json_file_path, "r") as json_file:
            data = json.load(json_file)
        # Append new user data to the JSON structure
        data["users"].append(user_info)
        # Save the updated data to JSON
        with open(json_file_path, "w") as json_file:
            json.dump(data, json_file, indent=4)
        return user_info
    except json.JSONDecodeError as e:
        st.error(f"Error decoding JSON: {e}")
        return None
    except Exception as e:
        st.error(f"Error creating account: {e}")
        return None

def email_exists(email, json_file_path):
    try:
        if os.path.exists(json_file_path) and os.stat(json_file_path).st_size != 0:
            with open(json_file_path, "r") as json_file:
                data = json.load(json_file)
                users = data.get("users", [])
                for user in users:
                    if user.get("email") == email:
                        return True
    except Exception as e:
        st.error(f"Error checking email existence: {e}")
    return False

def main(json_file_path="data.json"):
    st.sidebar.title("Real-time Anomaly Detection")
    page = st.sidebar.radio(
        "Go to",
        ("Signup/Login", "Dashboard", "Upload Files"),
        key="Real-time Anomaly Detection",
    )

```

```

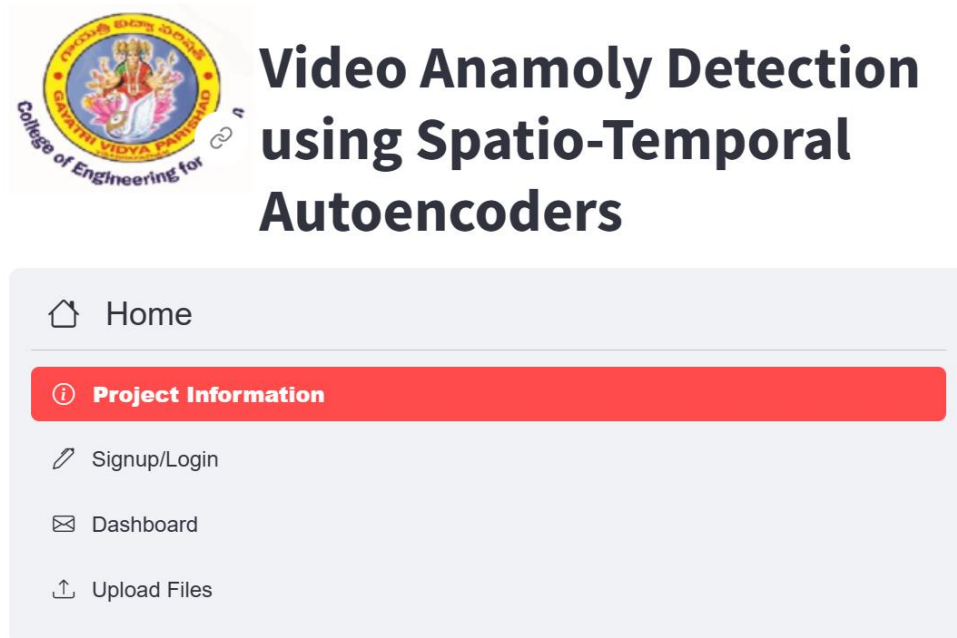
)
if page == "Signup/Login":
    st.title("Signup/Login Page")
    login_or_signup = st.radio(
        "Select an option", ("Login", "Signup"), key="login_signup"
    )
    if login_or_signup == "Login":
        login(json_file_path)
    else:
        signup(json_file_path)
elif page == "Dashboard":
    if session_state.get("logged_in"):
        render_dashboard(session_state["user_info"])
    else:
        st.warning("Please login/signup to view the dashboard.")
elif page == "Upload Files":
    if session_state.get("logged_in"):
        st.title("Upload Files")
        uploaded_files = st.file_uploader(
            "Choose images or a folder", type=["txt", "tif"]
        )
        if st.button("Upload") and uploaded_files is not None:
            name = uploaded_files.name
            if name.endswith(".txt"):
                path = "UCSD_Anomaly_Dataset.v1p2/" + name[:-4].replace("_", "/")
                evaluate(path)
                for file in os.listdir(path):
                    if file == "video.mp4":
                        video_file = open(path + "/" + file, "rb")
                        video_bytes = video_file.read()
                        st.video(video_bytes)
if __name__ == "__main__":
    initialize_database()
    main()

```

## 5.5 RESULTS

### 5.5.1 Output Screens

Home Page:



Screen 1. Home Page

Project Information Page:

## Project Information

This project aims to develop a real-time anomaly detection system using Spatio-Temporal Autoencoders. The system processes video data and identifies anomalies or unusual patterns that deviate from normal behavior.

### Key Features:

- Spatio-Temporal Autoencoders for feature extraction.
- Anomaly detection based on reconstruction error.
- User-friendly dashboard for monitoring and analysis.

### Usage:

To get started, you can sign up or log in to access the dashboard. Once logged in, you can upload video files for anomaly detection or explore the dashboard for insights.

### About Us:

This project is developed by Team A6 as part of our curriculum. As a team, we are passionate about applying machine learning techniques to solve real-world problems and enhance security and surveillance systems.

Screen 2. Project Information Page

## Signup page:

# Signup Page

Fill in the details below to create an account:

Name:

JOHN

Email:

johns@bmail.com

Age:

54 - +


Sex:

☒ Male


☐ Female

☐ Other

Password:

\*\*\*\*\* 

Confirm Password:

\*\*\*\*\* 

Signup

Account created successfully! You can now login.

Screen 3. After Sign Up form

## Validation of Form:

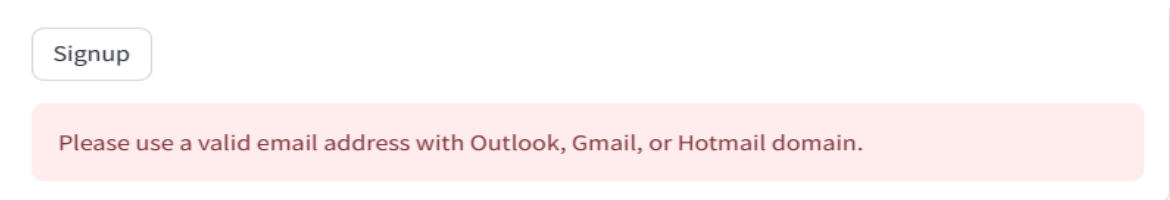
Signup

All fields are required. Please fill in all the details.

Already a user? Log in now!

Screen 4. Validation of Form

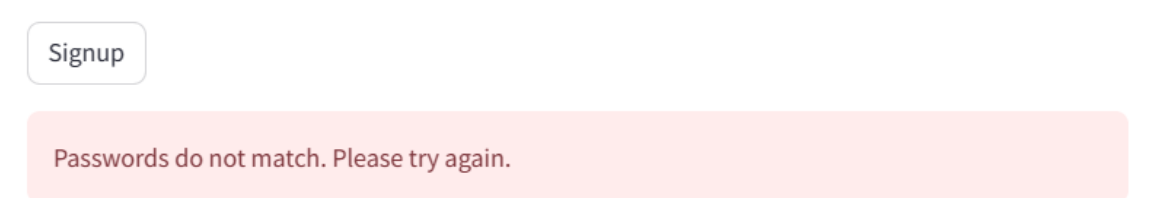
### Validation of Email Field:



A screenshot of a web form titled "Signup". Below the form fields, there is a red error message box that reads: "Please use a valid email address with Outlook, Gmail, or Hotmail domain."

Screen 5. Validation of Email Field

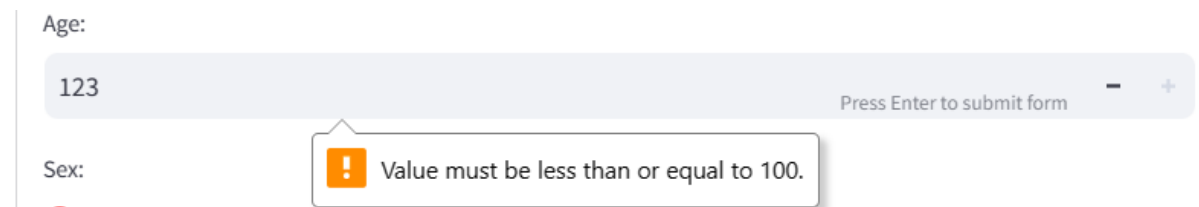
### Validation of Password Field:



A screenshot of a web form titled "Signup". Below the form fields, there is a red error message box that reads: "Passwords do not match. Please try again."

Screen 6. Validation of Password Field

### Validation of Age Field:

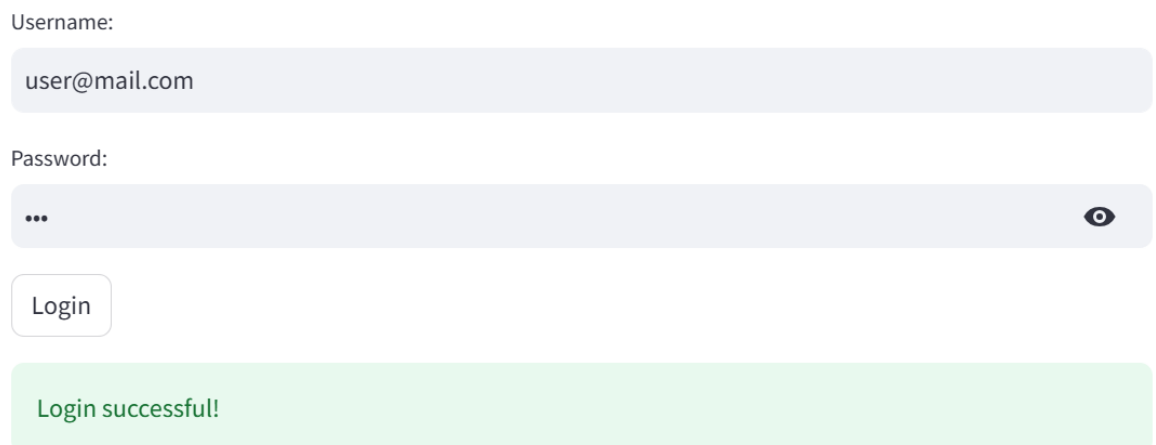


A screenshot of a web form. The "Age:" field contains the value "123". To the right of the field is a hint text: "Press Enter to submit form". Below the "Age:" field, there is a red error message box that reads: "Value must be less than or equal to 100."

Screen 7. Validation of Age Field

### Login page:

## Login Page



A screenshot of a login page. It features a "Username:" label above a text input field containing "user@mail.com". Below that is a "Password:" label above a password input field with a toggle icon. A "Login" button is positioned below the password field. At the bottom, there is a green success message box that reads: "Login successful!"

Screen 8. Login Page

### Invalid Login details:

Login

Invalid credentials. Please try again.

Invalid credentials. Please try again.

Screen 9. Invalid Login details

### Dashboard page:

#### Before login:

### Real-time Anomaly Detection

Go to

- ☐ Signup/Login
- ☒ Dashboard
- ☐ Upload Files

Please login/signup to view the dashboard.

Screen 10. Before Login

#### After login:

# Welcome to the Dashboard, User!

### User Information:

Name: User

Sex: Male

Age: 22

Screen 11. After Login



## Uploading page:

### Upload Files

Choose images or a folder

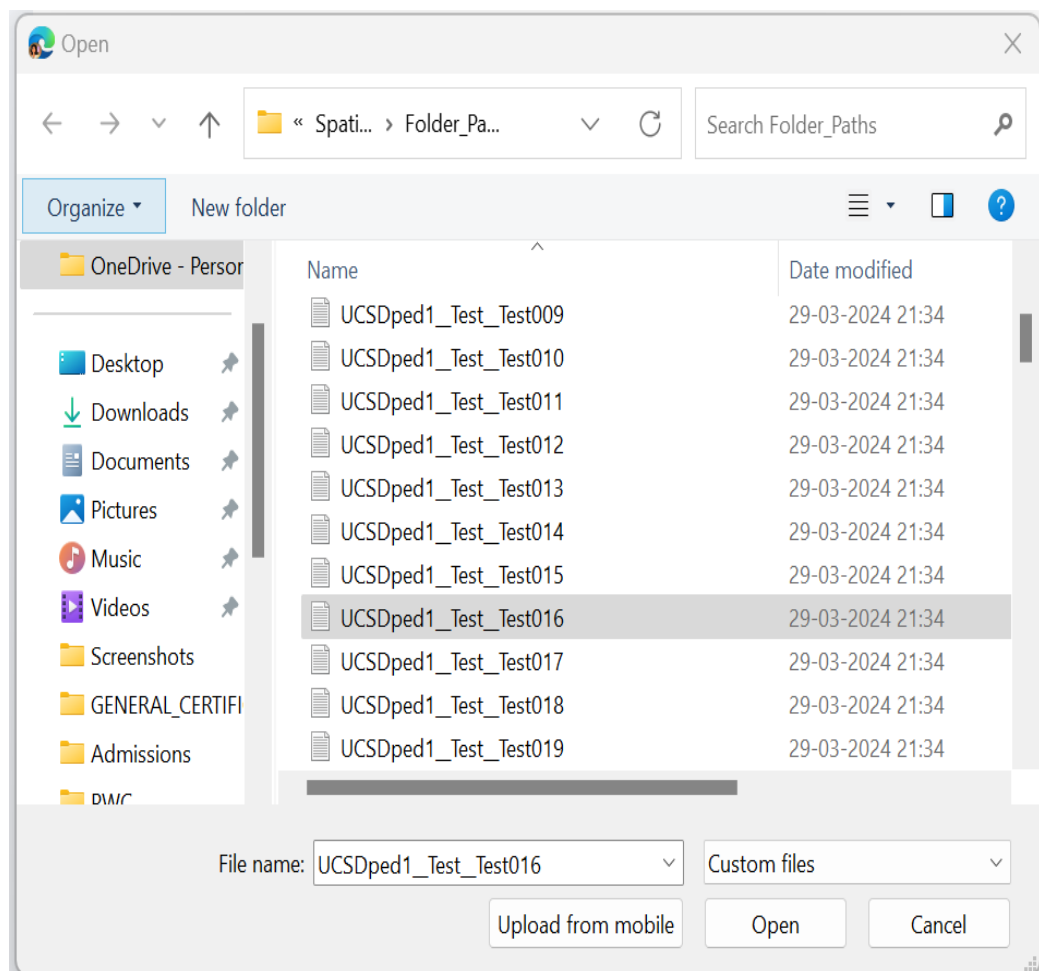


Drag and drop file here

Limit 200MB per file • TXT, TIF, TIFF

Browse files

Upload



Screen 12. Uploading page

After uploading video, detection page:

## Upload Files

Choose images or a folder



Drag and drop file here

Limit 200MB per file • TXT, TIF, TIFF

Browse files



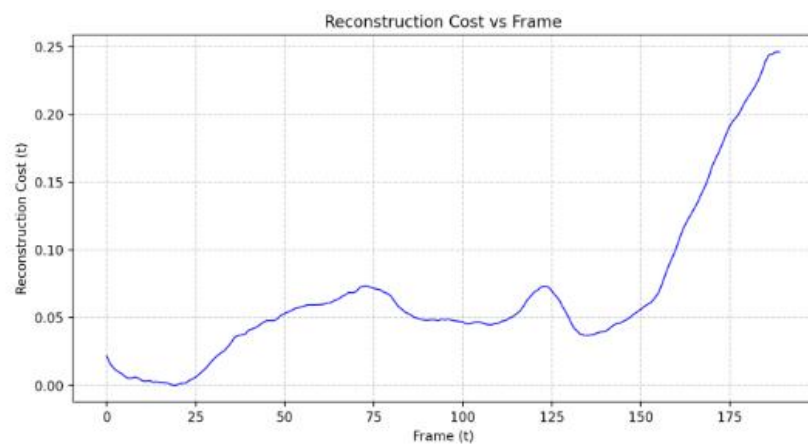
UCSDped1\_\_Test\_\_Test030.txt 0.0B



Upload

Screen 13. After uploading video

Detection Page:



Screen 14. Detection page

### 5.5.2 Graphs

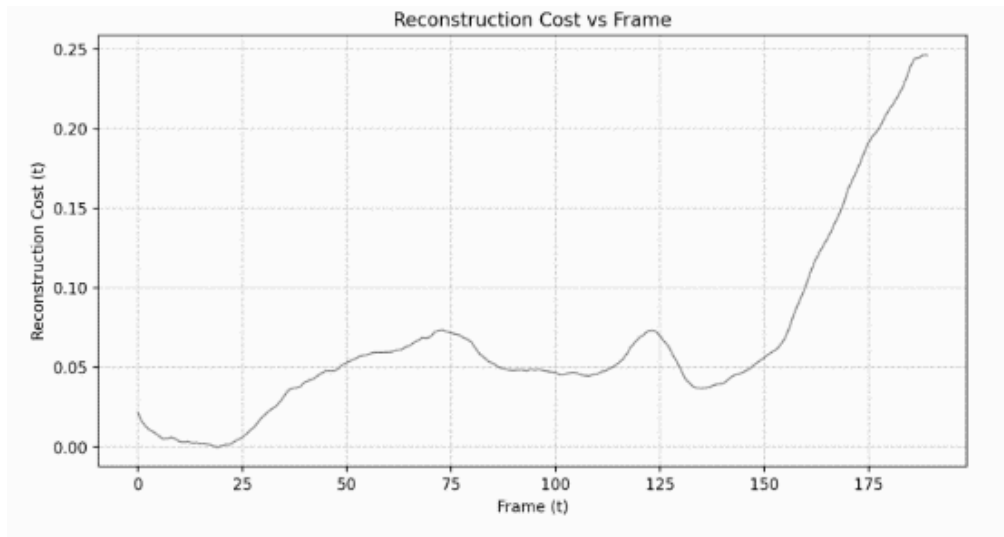


Fig. 16. Reconstruction Cost Vs Frame

In Fig. 16, this plot visualizes the reconstruction cost of video sequences processed by the anomaly detection model. Here's how to interpret and use the plot effectively:

1. **X-axis (Frame):** Represents the frame number or timestamp of the video sequence. Each point on the X-axis corresponds to a specific frame in the video.
2. **Y-axis (Reconstruction Cost):** Indicates the reconstruction cost associated with each frame. The reconstruction cost measures how well the model reconstructs each frame of the input video. Higher reconstruction costs indicate greater deviations from the expected or normal behavior.
3. **Plot Line:** The blue line represents the reconstruction cost over time. It shows fluctuations in the reconstruction cost throughout the video sequence. Anomalies or unusual patterns in the video typically correspond to peaks or sudden increases in the reconstruction cost.

#### How to use the plot effectively:

- **Identifying Anomalies:** Look for spikes or peaks in the plot, as they indicate potential anomalies or abnormal behavior in the video. These spikes represent frames where the reconstruction cost is significantly higher, suggesting deviations from normal patterns.
- **Thresholding:** Establish a threshold value for the reconstruction cost based on the application's requirements and the characteristics of the input data. Frames with reconstruction costs above this threshold may be flagged as anomalies.
- **Visual Inspection:** Visualize the video frames corresponding to high

reconstruction cost regions to gain insights into the nature of anomalies. Reviewing these frames can help in understanding the context of detected anomalies and making informed decisions.

In summary, the plot provides a visual representation of the reconstruction cost over time, aiding in the detection and analysis of anomalies in video data. It serves as a valuable tool for identifying abnormal patterns and facilitating further investigation into potential security breaches or irregularities.

### 1. Finding Threshold

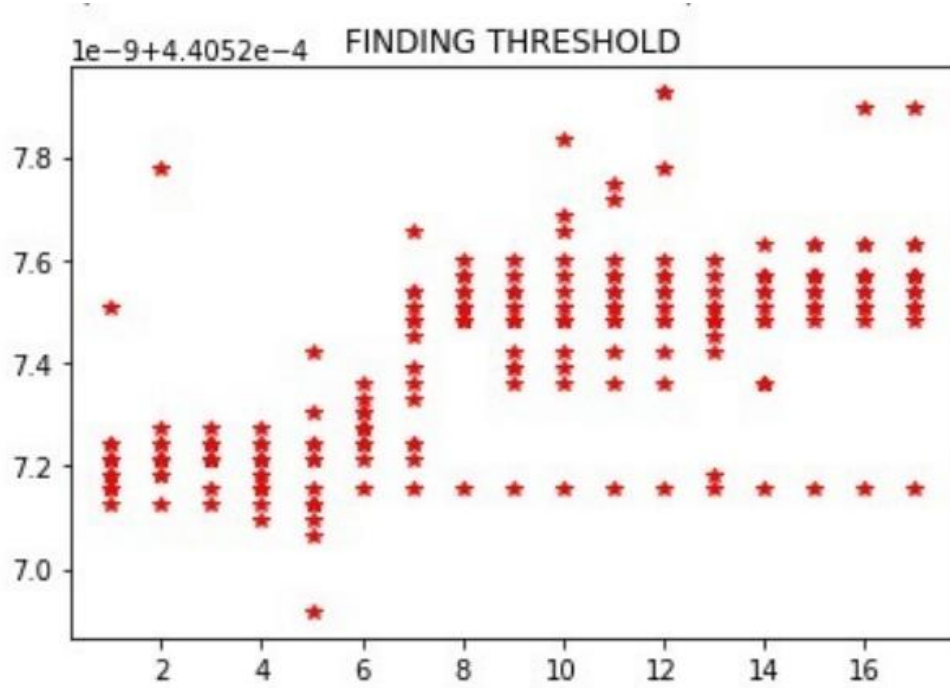


Fig. 17. Finding Threshold

In Figure 17, the visualization demonstrates the distribution of loss values for the training examples. From the graph, it is evident that the range of loss values lies between approximately 0.0004405292 and 0.0004405265. This range provides crucial information for determining the threshold value for anomaly detection.

By analyzing the loss values, choose the maximum value from this range as our threshold value, which is 0.00044052925. This decision is based on the assumption that normal frames should have loss values within this range, and any frame with a loss value above the threshold is considered anomalous.

Using this threshold value, the proposed system can effectively classify video sequences as either normal or anomalous. Frames with loss values below the threshold

are classified as normal, while those with loss values above the threshold are classified as anomalies.

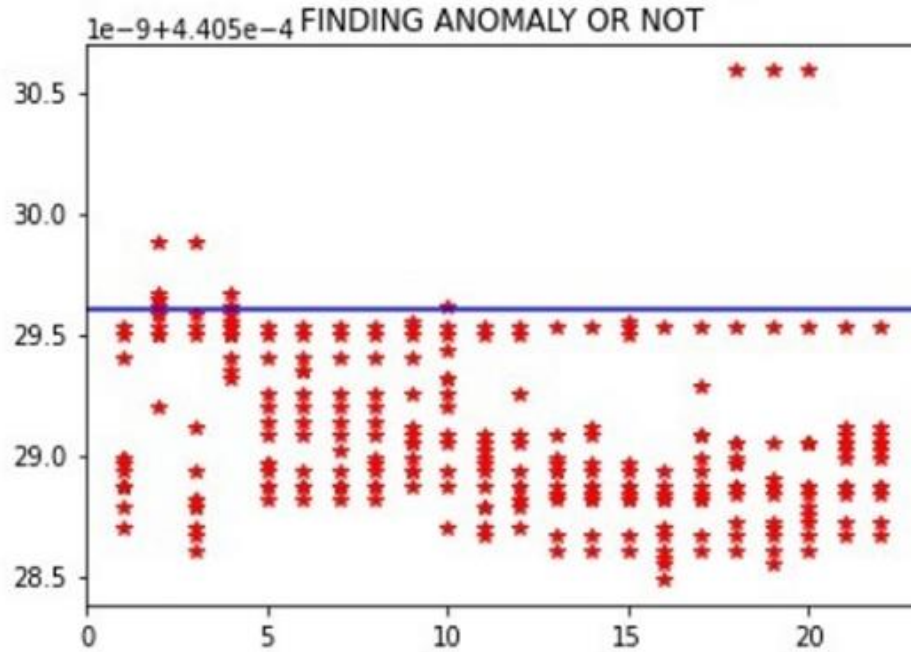


Fig. 18 Finding Anomaly Plot

In Fig. 18, the visualization represents the relationship between the computed loss values for the video frames and the chosen threshold value. The blue line in the graph indicates the threshold value, which has been determined based on the analysis of loss values from the training examples, as discussed previously.

Based on the graph, deduce that a frame is deemed anomalous if its loss value is higher than the threshold, which is represented by the blue line. The underlying premise of this categorization is that abnormal frames have larger loss values, whilst normal frames should have loss values below the threshold.

Additionally, the suggested approach operates on the tenet that a video is deemed anomalous if at least one frame in the video is identified as abnormal. This method aids in making sure that any unusual activity in the video clip is recorded and appropriately marked. In Fig. 19. infer that it has made a visualization of which frame is anomaly and which isn't.

## 2. Training and Validation Accuracy

X-axis (epochs): The number of training epochs is represented by the X-axis (epochs). During the training phase, one full pass through the entire training dataset is represented by each epoch.

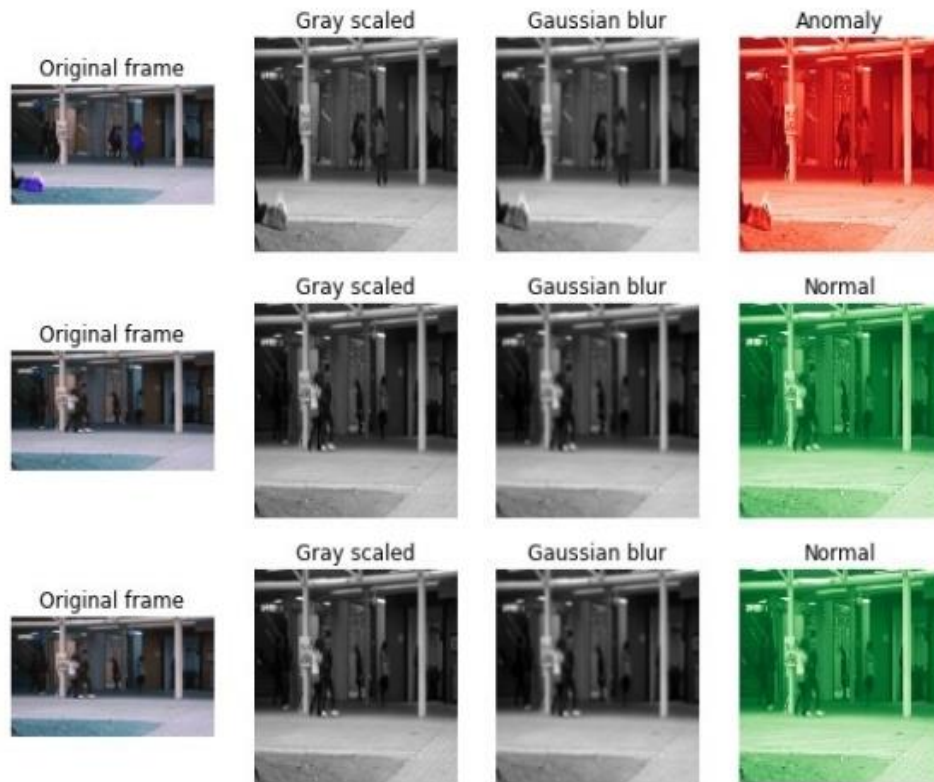


Fig. 19. Plots of results in various stages

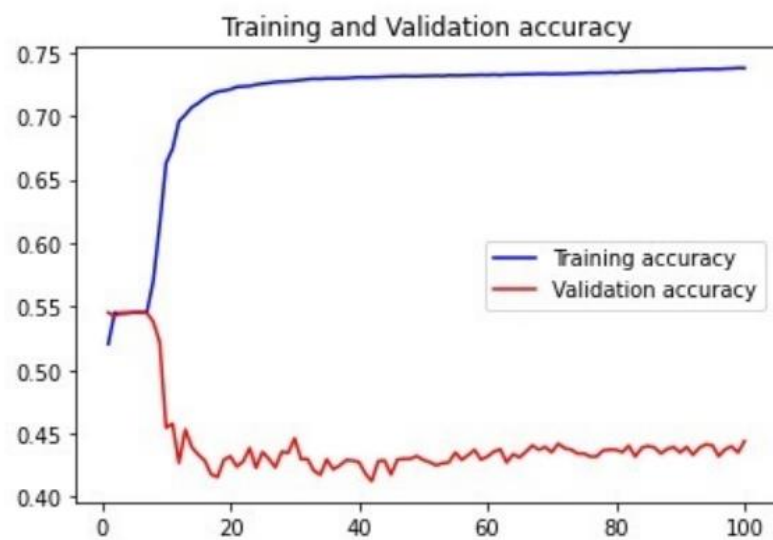


Fig. 20 Graph plotted between Training accuracy vs Validation accuracy

Y-axis (accuracy): The accuracy of your model on the training set of data is shown on the Y-axis. The percentage of correctly identified samples in the dataset relative to the total number of samples is called accuracy. It has a percentage as its expression.

In fig. 19, based on these plots and graphs, it can be concluded that the model can predict training data more accurately, the training curve is not noisy, and the validation curve is noisy.

In fig. 20, deduce from these plots and graphs that the model has been trained somewhat because the curve does not continue to decrease, allowing for the fine adjustment of hyperparameters.

### 3. Training and Validation Loss

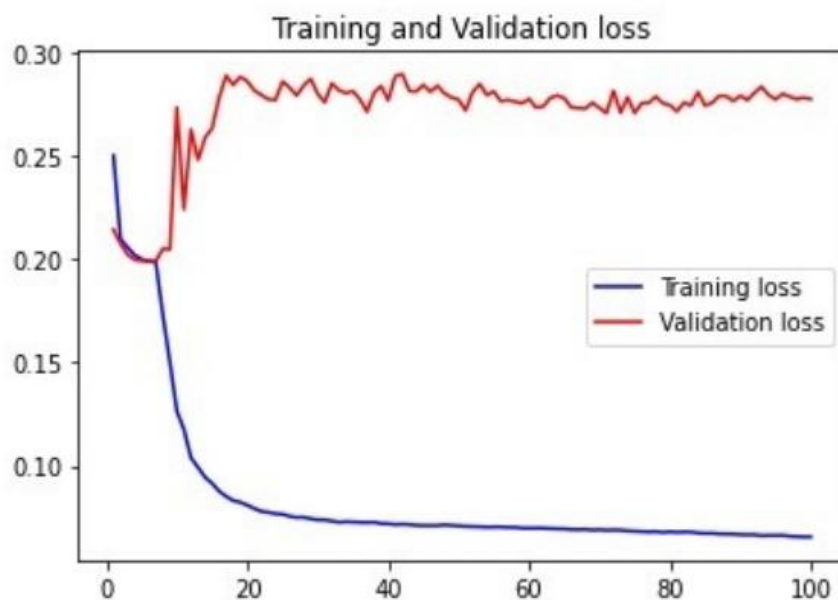


Fig. 21. Graph plotted between Training loss vs Validation loss

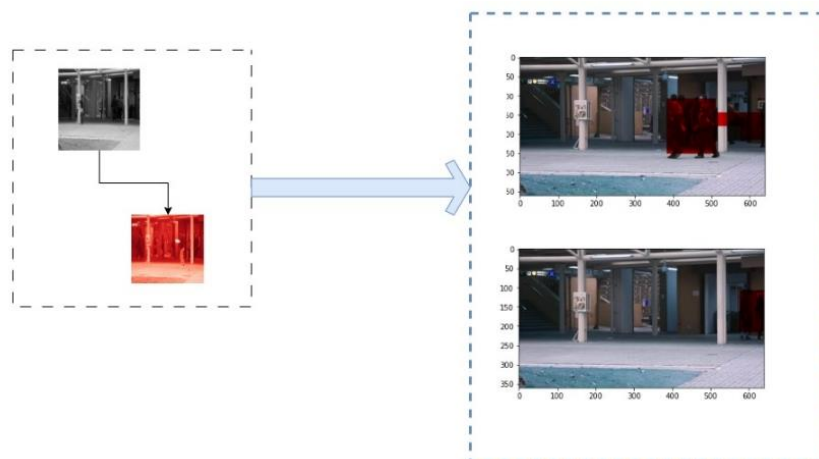


Fig. 22. Examples of anomaly detection shown for a frame vs localized cuboid model

X-axis (epochs): This axis represents the number of training epochs, which are iterations over the entire training dataset. Each epoch corresponds to one complete pass through the training data.

Y-axis (loss): This axis represents the loss function value, which is a measure of how well the model is performing during training. The loss function computes the difference between the predicted output and the true target value for each sample in the training set.

Fig. 22, demonstrates the application of the cuboid method in anomaly detection, focusing on identifying and highlighting specific anomalous regions within a frame rather than classifying the entire frame as anomalous. By comparing the reconstruction loss to a predefined threshold, the model determines whether a specific cuboid contains an anomaly or not. This approach enables a more granular level of anomaly detection, as opposed to treating the entire frame as a single unit. Once the model has identified the anomalous cuboids, their corresponding regions in the original frame are highlighted.

## 5.6 CONCLUSION

The model outlined here is a specialized convolutional autoencoder geared towards detecting anomalies in spatio-temporal data, particularly designed for video analysis. Its architecture is tailored to handle input data in five dimensions, accounting for batch size, height, width, temporal depth, and channels. By incorporating convolutional layers for spatial feature extraction, ConvLSTM layers to capture temporal dependencies, and convolutional transpose layers for precise reconstruction, the model can effectively identify intricate spatio-temporal patterns present in video sequences.



## **6. TESTING AND VALIDATION**

### **6.1 INTRODUCTION**

Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest.

Testing is important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the product. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effective and customer satisfaction.

#### **6.1.1 Scope**

A primary purpose for testing is to detect software failures so that defects may be uncovered and corrected. This is a non-trivial pursuit. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed.

#### **6.1.2 Defects and Failures**

Not all software defects are caused by coding errors. One common source of expensive defects is caused by requirements gaps, e.g., unrecognized requirements that result in errors of omission by the program designer. A common source of requirements gaps is non-functional requirements such as testability, scalability, maintainability, usability, performance, and security. Software faults occur through the following processes. A programmer makes an error (mistake), which results in a defect (fault, bug) in the software source code. If this defect is executed, in certain situations the system will produce wrong results, causing a failure. Not all defects will necessarily

result in failures. For example, defects in dead code will never result in failures. A defect can turn into a failure when the environment is changed. Examples of these changes in environment include the software being run on a new hardware platform, alterations in source data or interacting with different software. A single defect may result in a wide range of failure symptoms.

### **6.1.3 Compatibility**

A frequent cause of software failure is compatibility with another application, a new operating system, or, increasingly, web browser version. In the case of lack of backward compatibility, this can occur because the programmers have only considered coding their programs for, or testing the software upon, "the latest version of" this-or-that operating system. The unintended consequence of this fact is that: their latest work might not be fully compatible with earlier mixtures of software/hardware, or it might not be fully compatible with another important operating system. In any case, these differences, whatever they might be, may have resulted in software failures, as witnessed by some significant population of computer users.

### **6.1.4 Input Combinations and Preconditions**

A very fundamental problem with software testing is that testing under all combinations of inputs and preconditions is not feasible, even with a simple product. This means that the number of defects in a software product can be very large and defects that occur infrequently are difficult to find in testing. More significantly, non-functional dimensions of quality (how it is supposed to be versus what it is supposed to do) can be highly subjective; something that constitutes sufficient value to one person may be intolerable to another. Precondition is defined as Environmental and state conditions that must be fulfilled before the component or system can be executed with a particular test or test procedure.

In this project, CCTV footages are extracted from Avenue dataset and anomaly detection dataset, videos are preprocessed before detecting the anomalies, the frames generated through ffmpeg will be fed as input to the model and then further converted into cuboids. Preconditions are the environmental versions like it works good on Visual Studio Code and python of version 3.10.12, most importantly it supports tensorflow version of 2.15.0.

### 6.1.5 Static Vs Dynamic Testing

The main objective of static testing is to improve the quality of software applications by finding errors in early stages of the software development process. Whereas in dynamic testing, code is executed. It checks for functional behaviour of the software system, memory/CPU usage and overall performance of the system.

### 6.1.6 Software Verification and Validation

**Verification:** It is the process of checking that software achieves its goals without any bugs. It is the process to ensure whether the product is right or not.

**Validation:** It is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It checks what we are developing is the right product.

## 6.2 DESIGN OF TEST CASES AND SCENARIO

The test cases are designed for our model as below in Table 9. The test cases are formatted so that the user can enter values in the proper format.

Table 7. Black box Testing

| S.No. | Test Case                | Uploaded Video                       | Expected Result | Actual Result | Pass/Fail |
|-------|--------------------------|--------------------------------------|-----------------|---------------|-----------|
| 1.    | Testing from the dataset | A man throwing papers                | Anomaly         | Anomaly       | Pass      |
| 2.    | Testing from the dataset | Vehicles moving through intersection | Normal          | Anomaly       | Fail      |
| 3.    | Testing from the dataset | Pedestrians crossing at crosswalks   | Normal          | Normal        | Pass      |
| 4.    | Testing from the dataset | Rapid movements                      | Anomaly         | Anomaly       | Pass      |
| 5.    | Testing from the dataset | Dancing in crowd                     | Normal          | Anomaly       | Fail      |
| 6.    | Testing from the dataset | Walking towards University           | Normal          | Normal        | Pass      |

### 6.2.1 Validation Testing

It is testing where testers performed functional and non-functional testing. Functional testing includes Unit Testing, Integration Testing and System Testing and non- functional testing includes User acceptance testing.

**Testing Objectives:**

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is one that has a high probability of finding an as-yet-undiscovered error.
- A successful test uncovers an as-yet-undiscovered error.

**Test Levels:** The test strategy describes the test level to be performed. There are primarily three levels of testing.

- Unit Testing
- Integration Testing
- System Testing

In most software development organizations, the developers are responsible for unit testing. Individual testers or test teams are responsible for integration and system testing.

**6.2.2 Unit Testing**

Unit testing is done on individual modules as they are completed and become executable. It is confined only to the designer's requirements. Each module can be tested using the following two strategies.

**6.2.3 Black Box Testing**

In this strategy some test cases are generated as input conditions that fully execute all functional requirements for the program. In this testing only the output is checked for correctness. The logical flow of the data is not checked. This testing has been used to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structure or external database access
- Performance errors

**6.2.4 White Box Testing**

White Box Testing is a software testing technique in which internal structure, design and coding of software is tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing and

Glass box testing. In this test cases are generated on the logic of each module by drawing flow graphs of that module and logical decisions are tested on all the cases.

#### **6.2.5 Integration Testing**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

#### **6.2.6 System Testing**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

### **6.3 VALIDATION TESTING SCREENS**

The System has been tested and implemented successfully and thus ensured that all the requirements as listed in the software requirements specifications are completely fulfilled. In case of any error the specified error messages are displayed.

### **6.4 CONCLUSION**

This chapter emphasizes the validation of the system using different testing approaches like black box testing and unit testing and obtaining the highest accuracy of 80%.

## 7. CONCLUSION

In conclusion, our method for anomaly detection in videos leverages the power of CNN-LSTM and spatiotemporal autoencoders to extract both spatial and temporal features from video data. The Avenue dataset, comprising 17 training videos and 12 testing videos, served as the foundation for our training and evaluation process. Additionally, the anomaly detection dataset comprises 1900 real-world surveillance videos showcasing various anomalous activities. During preprocessing, we observed enhanced efficiency by converting the videos into grayscale and applying Gaussian blur. These steps contributed to refining the input data for improved anomaly detection accuracy. Overall, this project has the potential to significantly enhance anomaly detection capabilities in video surveillance systems, paving the way for advancements in security and safety protocols. It represents a promising avenue for further exploration and refinement in the field of video analytics.

### Future Scope

The efficiency of the model can be improved for better performance by improving the data and training using hybrid models. There are several directions that can be explored to improve the performance of the proposed method. One possible direction is to incorporate additional features, such as optical flow, into the spatiotemporal autoencoder to capture more detailed information about the motion in the video. Another direction is to explore more advanced architectures for the autoencoder, such as recurrent neural networks or attention mechanisms, to further improve the representation of the data. Additionally, the method can be tested on other datasets and compare the performance with other state-of-the-art methods. Furthermore, the proposed method can be integrated into real-time systems such as surveillance cameras and traffic monitoring systems to improve their performance and efficiency.

## 8. REFERENCES

- [1] Guo, J., Zheng, P. and Huang, J., 2019. Efficient privacy-preserving anomaly detection and localization in bitstream video. In Transactions on Circuits and Systems for Video Technology, 30(9) (pp. 3268-3281). IEEE.  
Available at: <https://ieeexplore.ieee.org/document/8827750>
- [2] Zhou, J.T., Du, J., Zhu, H., Peng, X., Liu, Y. and Goh, R.S.M., 2019. Anomalynet: An anomaly detection network for video surveillance. In Transactions on Information Forensics and Security, 14(10) (pp. 2537-2550). IEEE.  
Available at: <https://ieeexplore.ieee.org/document/8844792>
- [3] Luo, W., Liu, W., Lian, D., Tang, J., Duan, L., Peng, X. and Gao, S., 2019. Video anomaly detection with sparse coding inspired deep neural networks. In transactions on pattern analysis and machine intelligence (pp. 1-6). IEEE.  
Available at: <https://ieeexplore.ieee.org/document/8474301>
- [4] Cheng, K.W., Chen, Y.T. and Fang, W.H., 2015. Gaussian process regression-based video anomaly detection and localization with hierarchical feature representation. In Transactions on Image Processing, 24(12) (pp. 5288-5301). IEEE. Available at: <https://ieeexplore.ieee.org/document/7243348>
- [5] Chu, W., Xue, H., Yao, C. and Cai, D., 2018. Sparse coding guided spatiotemporal feature learning for abnormal event detection in large videos. In Transactions on Multimedia, 21(1), (pp.246-255). IEEE.  
Available at: <https://ieeexplore.ieee.org/document/8336922>
- [6] Li, Y., Cai, Y., Liu, J., Lang, S. and Zhang, X., 2019. Spatio-temporal unity networking for video anomaly detection. In Access, 7, (pp.172425-172432). IEEE.  
Available at: <https://ieeexplore.ieee.org/document/8861091>
- [7] Leyva, R., Sanchez, V. and Li, C.T., 2017. Video anomaly detection with compact feature sets for online performance. In Transactions on Image Processing, 26(7), (pp.3463-3478). IEEE.  
Available at: <https://ieeexplore.ieee.org/document/7972692>
- [8] Ganokratanaa, T., Aramvith, S. and Sebe, N., 2020. Unsupervised anomaly detection and localization based on deep spatiotemporal translation network. In Access,8,(pp. 50312-50329). IEEE.  
Available at: <https://ieeexplore.ieee.org/document/9194438>

- [9] Wang, S., Zeng, Y., Liu, Q., Zhu, C., Zhu, E. and Yin, J., 2018, October. Detecting abnormality without knowing normality: A two-stage approach for unsupervised video abnormal event detection. In Proceedings of the 26th ACM international conference on Multimedia (pp. 636-644). IEEE.  
Available at: <https://dl.acm.org/doi/10.1145/3240508.3240654>
- [10] Jardim, E., Thomaz, L.A., da Silva, E.A. and Netto, S.L., 2019. Domain-transformable sparse representation for anomaly detection in moving-camera videos. In Transactions on Image Processing, 29, (pp.1329-1343). IEEE.  
Available at: <https://ieeexplore.ieee.org/document/8653297>
- [11] Li, N., Chang, F. and Liu, C., 2020. Spatial-temporal cascade autoencoder for video anomaly detection in crowded scenes. In Transactions on Multimedia, 23, (pp.203-215).IEEE.  
Available at: <https://ieeexplore.ieee.org/document/9036878>
- [12] Nawaratne, R., Alahakoon, D., De Silva, D. and Yu, X., 2019. Spatiotemporal anomaly detection using deep learning for real-time video surveillance. In Transactions on Industrial Informatics, 16(1), (pp.393-402).IEEE.  
Available at: <https://ieeexplore.ieee.org/document/8782566>
- [13] Xiang, T. and Gong, S., 2008. Video behavior profiling for anomaly detection. In transactions on pattern analysis and machine intelligence, 30(5), (pp.893-908). IEEE.  
Available at: <https://ieeexplore.ieee.org/document/4439877>
- [14] Thomaz, L.A., Jardim, E., da Silva, A.F., da Silva, E.A., Netto, S.L. and Krim, H., 2017. Anomaly detection in moving-camera video sequences using principal subspace analysis. In Transactions on Circuits and Systems I: Regular Papers, 65(3), (pp.1003-1015). IEEE.  
Available at: <https://ieeexplore.ieee.org/document/7869456>
- [15] Li, W., Mahadevan, V. and Vasconcelos, N., 2013. Anomaly detection and localization in crowded scenes. In transactions on pattern analysis and machine intelligence, 36(1), (pp.18-32). IEEE.  
Available at: <https://ieeexplore.ieee.org/document/6341741>
- [16] Xu, K., Jiang, X. and Sun, T., 2018. Anomaly detection based on stacked sparse coding with intraframe classification strategy. In Transactions on Multimedia, 20(5), (pp.1062-1074). IEEE. Available at: <https://ieeexplore.ieee.org/document/8108395>
- [17] Zhao, Y., Deng, B., Shen, C., Liu, Y., Lu, H. and Hua, X.S., 2017, October. Spatio-temporal autoencoder for video anomaly detection. In Proceedings of the 25th ACM



international conference on Multimedia (pp. 1933-1941). IEEE.

Available at: <https://dl.acm.org/doi/10.1145/3123266.3123451>

[18] Yuan, Y., Fang, J. and Wang, Q., 2014. Online anomaly detection in crowd scenes via structure analysis. In transactions on cybernetics, 45(3), (pp.548-561). IEEE.

Available at: <https://ieeexplore.ieee.org/document/6814043>

[19] Tariq, S., Farooq, H., Jaleel, A. and Wasif, S.M., 2021. Anomaly detection with particle filtering for online video surveillance. In Access, 9, (pp.19457-19468). IEEE.

Available at: <https://ieeexplore.ieee.org/document/9508397>

[20] R. U. C. Pati and S. Kumar Das, "Video Anomaly Detection using Convolutional Spatiotemporal Autoencoder," 2020 International Conference on Contemporary Computing and Applications (IC3A), Lucknow, India, 2020, (pp. 175-180). IEEE.

Available at: <https://ieeexplore.ieee.org/document/9074041>