ENHANCED DEEP LEARNING MODEL TO DETECT ANOMALIES IN SURVEILLANCE VIDEOS

John Gatara Munyua

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of Master of Science in Information Technology of Murang'a University of Technology

August, 2022

DECLARATION

I hereby declare that this thesis is my original work and to the best of my knowledge has not been presented for a degree award in this or any other university.

16/08/2022

John Gatara Munyua Date

SC401/5075/2018

APPROVAL

The undersigned certify that they have read and hereby recommend for acceptance of Murang'a University of Technology a thesis entitled "**Enhanced Deep Learning Model to Detect Anomalies in Surveillance Videos**."

_____ _____

Dr. Geoffrey Mariga Wambugu, Ph.D. Date

Department of Information Technology,

Murang'a University of Technology.

_____ _____

Dr. Stephen Thiiru Njenga, Ph.D. Date

Department of Computer Science,

Murang'a University of Technology.

DEDICATION

Dedicated to:

My lovely wife Grace and our daughter Gillian,

You have made my life extremely comfortable and given me hope.

My Mother Mary,

You have always believed in me.

My siblings Catherine, Florence, and Winnie,

Thank you for the support and encouragement.

ACKNOWLEDGEMENT

My humble gratitude goes to the almighty God, for blessing me with know-how, peace of mind and good health to finish this thesis.

My vote of thanks, goes to my brilliant supervisors; Dr Geoffrey Mariga and Dr Stephen Njenga, who has guided me well throughout my research journey. I was fortunate to have supervisors, who encouraged me when the journey was rocky.

To Prof. Geoffrey Muchiri, I appreciate the role you played in the initial stages of this thesis. To Dr Gabriel Kamau, Dr Aaron Oirere and Dr John Ndia I appreciate your continued support and guidance in my work. To the School of Computing and Information Technology (SCIT) fraternity at MUT, thank you for the directions you have provided and for continued support throughout my work.

I will be forever, grateful to my family for their patience, understanding and tolerance throughout this journey.

MAY GOD BLESS YOU ABUNDANTLY

ABSTRACT

Increased security challenges and advancements in technology have led to heavy usage of surveillance cameras. This has resulted in an overwhelming abundance of video data which requires automated analytics for better utilization. The big volume of the video data generated by surveillance devices presents an enormous problem to the security personnel since they must monitor the footage frame by frame to identify the abnormal activities (security threats) like violence, and thuggery, among others. Successful identification of anomalies in surveillance footage will ease the work of Closed-Circuit Television (CCTV) operators greatly since they can search through a big volume of the video data easily. Another importance of this research is the contribution to computer vision since the model can be applied in other areas like robotic surveillance or unmanned surveillance. There have been attempts to automate the surveillance process using smart surveillance. However, these solutions are challenged due to high error rates and inefficiency while identifying abnormal scenes. Modern automated video analytics, use deep learning algorithms like; Convolutional Neural Networks (CNN), Long-Short Term Memory (LSTM), convolutional LSTM and 3DCNN. These approaches have their strengths and weaknesses, and it becomes a research challenge to determine the best model to use in detecting anomalies. Another challenge presented herein is the accuracy of detecting anomalies in surveillance videos. A comparative study was carried out to cross-examine deep learning models used in anomaly detection. Empirical data was collected to measure the accuracy of the deep learning models in anomaly detection. The best model was determined by analyzing the accuracies of the model published since 2016. Experiments were set up in Google Collab and Google Cloud. These environments were configured to use Python 3.7, Keras and TensorFlow machine learning frameworks. The study improved the selected deep learning model through, optimization of the model structure and depth tuning. The study found that deeper autoencoders have high prediction accuracy and deeper spatial autoencoders draws more features from the videos and that increases their accuracy. Validation of the enhanced model was done through further experiments that compared the prediction accuracy acquired from the enhanced model against the existing model set as the control group. Their Receiver Operating Characteristic Curve (ROC) scores from UCSD Ped1 and Ped2 datasets were compared. Comparative analysis of the recorded model accuracies was tabulated and a percentage increase in the model accuracy was noted. A sign test was used to test the significance of the improvement and at both 1% and 5% significance levels, empirical evidence of the enhancement was found. This work contributed to the autoencoder design paradigms, improvement of Spatial-Temporal Autoencoder accuracy through depth and regularization tuning and reduction of anomaly detection errors in surveillance videos. The study has shown that the depth of spatial-temporal autoencoder impacts its anomaly prediction accuracy. In future work, integration of continual learning and real-time anomaly detection should be considered.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

3D	3 Dimensions
3DCNN	3D Convolutional Neural Network
AI	Artificial Intelligence
ANN	Artificial Neural Network
AUC	Area Under Curve
CCTV	Closed Circuit Television
CNN	Convolutional Neural Network
ConvLSTM	Convolutional Long-Short Term Memory
DBN	Deep Belief Networks
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DSN	Deep Stacking Networks
FCN	Fully Connected Network
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Model
HD	High Definition
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge
LSTM	Long-Short Term Memory
NVR	Network Video Recorder
RBM	Restricted Boltzmann Machine
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
STAE	Spatial Temporal Autoencoder

UCF	University of Central Florida
UCSD	University of California San Diego
YOLO	You Look Only Once
YOLOV3	You Look Only Once Version 3

CHAPTER ONE

INTRODUCTION

1.1 Background to the Study

Video surveillance has evolved hand in hand with photographic cameras. Rapid improvements in networks, storage and processing have made it possible to capture digital video data which can be processed in real-time [1]. Recently, video surveillance has increased rapidly to enhance security. Technology growth has made video surveillance evolve from analogue cameras to High Definition (HD) digital cameras which are accessible over the internet through Network Video Recorders [2]. The growth of video surveillance has given rise to the integration of security surveillance systems to address the complex operational and security needs of organizations.

Video surveillance systems have become a critical part of the security and protection system of modern cities, homes, and institutions. The video surveillance serves as a distant monitoring tool for management and security forces. It is an important subsystem required to complete a security plan. To achieve the goals of surveillance, closed-circuit television cameras (CCTVs) have been deployed massively which gives rise to a large amount of video data [3].

China and India have been noted to have the largest dense number of CCTV camera systems per square kilometer [4]. A metropolis called Chennai in India has the highest number of CCTVs Installations per square kilometer, it has about 657 surveillance systems deployed per square kilometer, which makes it the number one city in the world

in terms of security installations [4]. The projected trend of video surveillance is that data will continue to grow since more cameras will continue to be deployed. Hence, the introduction of cloud storage for surveillance data and online surveillance cameras like Wi-Fi cameras [4].

Notable improvements in security surveillance include night vision. The night vision feature allows cameras to monitor in low-lit and pitch-dark areas without loss of image quality. Both video and image data collected are clear for analysis and monitoring [1].

The act of surveillance involves observation of sceneries with intention of identifying specific behaviors, which are inappropriate, or in some manner indicate the possibility of the emergence of improper behavior or activity. Traditionally, the effectiveness of video surveillance required human intervention. People need to be on the lookout, in the control room throughout to detect abnormal activities (security threats) [2].

Video surveillance can be applied to security systems and investigations when there is proper monitoring. Proper monitoring involves having human monitors in real-time keenly reviewing the surveillance footage to identify malicious or criminal activities. This task requires an elevated level of commitment since anomalies are rare and can happen within a few seconds. The increase in surveillance video data requires more human monitors to effectively monitor. The human resource comes with additional associated costs [3].

High-Definition Video requires large storage to capture the surveillance footage over a certain period. This introduces the problem of storage since, large sets of data(videos) are

generated every millisecond of the surveillance, this increases the cost of storage. Hence, the cost of storage is among the determinants of the duration in which, the videos are stored [1]. The excessive cost of storage can be associated with the storage of unnecessary footage. Researchers have made numerous attempts to achieve intelligent surveillance. For instance, storage optimization, where the video management system stores the video when any motion subject is sensed [5].

The growth of computing technology has given rise to advanced computing concepts like artificial intelligence which provides complex cognitive processing paradigms like machine learning and deep learning. These technologies have given birth to intelligent surveillance [4].

Early attempts of intelligence surveillance include the integration of contactless solutions to video surveillance like facial recognition, vehicle plate recognition, and thermal camera body temperature screening to perform specific tasks using Internet of Things (IoT) gadgets. Blending video surveillance with artificial intelligence enables the surveillance systems to perform intelligent tasks like recognition of humans, vehicles, objects, and events. The deep learning algorithms make comparisons with a reference object in different postures, angles, positions, and movements [5].

The intelligent surveillance problem has spiked a lot of interest in computer vision algorithms with researchers aiming to have smart models which can monitor the scenes automatically. The act of identifying improper behaviors in surveillance videos can be referred to as anomaly detection. For instance, some of the most popular anomalies include violence, abuse, theft, traffic accidents, explosions, fighting, abuse, shooting,

weapons, stealing, vandalism, and shoplifting. Hence, the need to involve artificial intelligence in this field [3].

Artificial intelligence borrows the intelligence behavior of humans and uses computer systems to solve real-world problems like humans. Machine learning is a branch of Artificial Intelligence (AI), that uses past experiences(data) to solve a given problem [6]. The machine learning algorithms analyses the historical data and uses that knowledge to predict the future. An Artificial Neural Network (ANN) is an example of a machine learning algorithm that borrows heavily from the human brain. The ANN uses parallel networks with non-linear neurons that learn by adjusting the strengths of their connections. To address more complex cognitive intensive problems many layers of neural networks are stuck together. The stacking of many layers to solve complex problems that require complicated internal representations led to a variant of machine learning called deep learning.

The utilization of deep learning in video processing has shown promising results in areas like automatic recognition of temporal and spatial events in videos. Deep Learning has made it possible to train video analysis systems that mimic human behavior. In addition, these systems can identify specific objects in an image and track their path. Therefore, the presence of such technology forms a fundamental part of our abnormal scene detection research [5].

Deep learning can be considered as a subset of machine learning whose algorithms are inspired by the structure and functioning of the neural networks in the human brain. As the name ‘deep’ implies deep learning is all about the scale where larger synthetic neural

networks are trained with a huge amount of data, while their performance and accuracy continue to increase. This is notably different from other machine learning techniques that reach a plateau in their performance [6].

Deep Learning algorithms have achieved exceptional performance in areas like image recognition. For instance, AlexNet, Imagenet Large-Scale Visual Recognition Challenge (ILSVRC-2012). Such achievements have provided the foundation knowledge for video analytics research since a video can be broken down into frames then the frames can be analyzed and processed like images [7]. Convolutional neural networks (CNN) have been used in image processing since it works well for pattern recognition.

Another popular deep learning algorithm is Recurrent neural networks (RNN). This algorithm is quite useful while modelling time-series data. Since video data is spatial-temporal, RNNs preserve weights from sequential data. This network has made video analytics possible [8].

However, recurrent neural networks (RNNs) are not immune to problems faced by artificial neural networks such as vanishing/exploding gradient problems which limits their performance. This led to the development of its variation called Long Short-Term Memory (LSTM), which was developed by two Germany Scholars namely, Sepp Hochreiter and Juergen Schmidhuber [9]. LSTM can preserve, the estimation error that can be backpropagated through time and layers. This allows recurrent nets to learn over many steps (over 1000) thereby opening a channel to link causes and effects remotely.

LSTM networks work well in the processing of sequential data but are limited to one-dimensional data. This makes it hard for LSTM in its initial form to process video data that is three dimensional. The memory cell of LSTM was replaced by a convolutional layer to allow it to process spatial-temporal data hence the introduction of the ConvLSTM algorithm which is widely used for video processing [10].

Deep learning algorithms are stacked together or joined in layers and then compiled to work as a unit in a model. The models are designed using different learning architectures and depending on the job they should do. Deep learning models are based on artificial neural network variants and their design architectures are classified into supervised and unsupervised learning The nature of the problem dictates the model design [11].

Most importantly, a model designed to pinpoint and alert anomalies in surveillance videos can solve the storage problem since only the abnormal behavior footage will be stored and less human power will be needed for a surveillance operation.

1.2 Problem Statement

Many deep learning algorithms have been implemented to detect anomalies in surveillance videos [15]. Examples of the algorithms used include Convolution Neural Networks [16], 3D Convolutional Neural Networks [17], Long- Short Term Memory Autoencoder [18] and Conv-LSTM [19]. Chalapathy & Chwala [20] finds that 37.8% of the deep learning anomaly detection solutions are made of Auto-Encoders (AE), while 29% are made of CNN constructs, both LSTM and RNN had 4% each. These findings

however do not find what model is the best for use while constructing a deep learning solution for anomaly detection.

The challenge is determining the best algorithm among the anomaly detection models in surveillance videos. Little research has been conducted in the evaluation of deep learning models that are used for the detection of anomalies [21].

The second challenge is the high error rate in anomaly detection models. There are errors present in the detection of anomalies. Both false positive and false negative alarms are present in these models. The structure of the deep learning models affects their ability to identify anomalies well making them prone to errors while detecting anomalies. The quality of video representation and complexity scenes affects the performance of the model.

This research, therefore, exists to contribute to two gaps namely the empirical determination of the best deep learning model for the detection of anomalies in surveillance video and the enhancement of the deep learning model to reduce the false alarm rate.

1.3 Objectives of the study

1.3.1 Main Objective

The main objective of the study was to develop an enhanced deep learning model to detect anomalies in surveillance videos.

1.3.2 Specific Objectives

To achieve the main objective, the study was guided by the following specific objectives:

i. To investigate empirically, the selected deep learning models used to detect anomalies in surveillance videos with an aim of determining the best.
ii. To develop deep learning model that would optimize detection of anomalies in surveillance videos.
iii. To validate the developed enhanced deep learning model.

1.4 Research Questions

To fully achieve the stated objectives, the study will seek to answer the following research questions:

i. Which deep learning models are used to detect anomalies in surveillance videos?
ii. How can the selected deep learning model be improved to reduce high error rate?
iii. How valid is the developed enhanced model for anomaly detection in surveillance videos?

1.5 Justification

Surveillance videos are an important part of our security systems, ranging from the domestic, corporate and national levels. Surveillance equipment generates a vast amount

of video data at a remarkably high velocity. The generated video data require a large amount of storage space so that in case of any incidents, the footage can be referenced.

The overwhelming abundance of surveillance video data expresses the need for intelligent surveillance systems. Researchers have made numerous attempts to solve this problem. For instance, Chong and Tay [19], Sultani et al. [20], Bansod [21] and many others [22], [23] [17]. These solutions suffered from false alarm errors, and they need improvement.

This research sought to improve an existing model by modifying its architecture. Through improvement the study reduced the error rate in the model. Reduction of the error rate goes a long way to achievement of intelligent surveillance.

The study can ease the work of surveillance operators and be a significant contribution to the field of computer vision. With the application of the research to optimize the storage footage, if only the abnormal scenes are stored then a large amount of storage space can be saved.

1.6 Scope of the Study

The study focused on improvement of the deep anomaly detection model. Anomalies were taken as activities that deviated from the others. Anomalies included strange activities like having bikes and motor vehicles which are prohibited in a park. Other real-world threats to public safety such as Abuse, assault, traffic accidents, burglary, explosion, fighting, robbery, shooting, weapons, stealing, shoplifting and vandalism were also considered anomalies.

The study evaluated selected deep learning models used in anomaly detection and one was selected for improvement. The selected model architecture was improved by addition of more layers and optimization of the regularization functions. The enhancement process was focused on improvement of the model structure through depth optimization. The validation scope included comparison of the old model to the new enhanced model accuracy. The accuracy was validated further through test of significance.

The study was confined within Deep learning technology that is a subset of Machine Learning. Within deep learning the study focused on self-supervised models. Autoencoder model was the selected model for improvement. Internal structure of the autoencoder was optimized to increase the model accuracy.

1.7 Significance of the study

The importance of physical security in the real world cannot be underestimated. Video surveillance has become a key component of any comprehensive security system. Traditionally security surveillance provided video footage whether live or recorded. The advancement of technology has introduced the concept of smart surveillance that intends to provide knowledgeable insights from the video.

This study contributes to smart surveillance by providing means to improve the anomaly detection models. The study focused on improving a deep learning model that detects unusual activities by modifying the structure of the deep learning model to allow the extraction of more features.

Therefore, the greatest beneficiaries of the study will be the security personnel since the successful identification of anomalies will ease their work. Other contributions expected from this study are to the field of computer vision where unmanned surveillance can borrow heavily from anomaly detection.

Such contribution will improve the quality and efficiency of smart surveillance. Hence, the improvement of the security by making it even easier for enforcers to identify incidents in time and even deter crime.

1.8 Limitations of the study

The study focused on model improvement only. A model was chosen, and its architecture was altered to improve its anomaly prediction accuracy. The study was limited to depth tuning and model regularization enhancements. The study did not focus on the real-time anomaly detection for live surveillance videos. The study did not focus on optimizing the data structure to reduce its dimensions. The improvements were done to the model structure and that's what the study focused on.

1.9 Contributions of the Thesis

The research work made several contributions to the field of deep learning and smart surveillance by enhancement of intelligent surveillance. These contributions are as follows:

Improvement of anomaly detection accuracy by modification of the autoencoder model structure. Depth tuning of the autoencoder increased the model capacity to draw features

from complex surveillance scenes that in turn increased the model accuracy to optimize anomalies. The depth of the model was increased from 15 layers to 29 layers. Trainable parameters increased from 1,958,209 to 3,710,157 parameters.

Autoencoder model refinement and rapid improvement of the architecture of spatial-temporal autoencoders is another important theoretical knowledge contributed by this research.

Another contribution is in the autoencoder models troubleshooting, debugging and incremental design paradigm. This work takes an incremental stepwise deep learning model troubleshooting approach to continuously refine the model and select the best hyperparameters. The debugging knowledge can be applied in the designing and refinement of autoencoders.

1.10 Organization of the Thesis

This thesis is organized into five chapters. The first chapter contains the introduction to the thesis by highlighting the background information and the technology that has led to the growth and development of this field. The initial chapter also describes the problem found in the field, and how the researcher planned to address it, through the formulation of objectives and research questions, this chapter goes further to establish the delimitations of the study and it adjourns by providing a list of the contributions the study made to its field.

Chapter two describes relevant literature to the field of smart surveillance and the topic at hand. Chapter three details how the study was conducted through experimentation and

systematic review methodologies. Chapter four introduces the results and their interpretations and finally, the last chapter summarizes the study, gives recommendations, and gives suggestions for future works.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This chapter will explore other related work to the study. It will start by investigating available literature on popular deep learning models used in the detection of anomalies in surveillance videos. The available literature on deep learning model enhancement shall be explored as well. Most importantly, the datasets in this area will be considered, as well as model improvement and validation methods.

2.2 Overview of Anomaly Detection Deep Learning Models

This section describes the technology behind the deep learning technology and the terms encountered in anomaly detection. The section expounds on the theoretical framework from which the research originated.

2.2.1 Understanding Deep Anomaly Detection

It is paramount to describe the meaning of anomaly detection in video surveillance since it was the main subject of this study. [22] describes anomaly detection as a prominent level of video understanding which sieves out the abnormal events from the normal sequence of events [22].

The study is inclined to deep learning models that are used to detect anomalies. Therefore, it is important to understand deep learning as a subdivision of machine learning and artificial intelligence.

Ian Goodfellow gently defines deep learning as, "a form of machine learning that enables computers to learn from experience and understand the world in terms of a hierarchy of concepts. The hierarchy of concepts allows the system to learn sophisticated concepts by constructing them out of simple representations*"* [23].

Deep Learning is a subcategory of machine learning which is encouraged by the design and working of artificial neural networks (ANN) [12]. Deep Learning is sometimes referred to as Deep Neural Networks (due to the depth of the neural networks used) and is capable of learning features from structured and unstructured without supervision [8]. The inspiration for Neural networks is the way the human brain filters information. In a way, these neural networks mimic the human brain. The said networks are capable of learning unsupervised from both structured and unstructured data. [12] For Instance, in our brains, a neuron is composed of a body, axon and dendrites. The signal(dendrites) is transferred from one neuron to the other through the axon [8].

Therefore, the idea behind deep learning algorithms is that inputs are fed to the input layer whose output is fed as input to the next layer and so on for several hidden layers until the final output is achieved. As the name 'deep' implies larger neural networks with many hidden layers are trained with a huge amount of data, while their performance and accuracy continue to increase. However, this is notably different from other machine learning techniques that reach a plateau in performance [8].

Deep learning utilizes several layers of non-linear processing elements for transformation and feature extraction. The learning process creates a hierarchy of ideas where each level acquires a more abstract and composite representation from input vectors [13]. Processing at deep network nodes takes in the numerical data form, considering that each node has a number that aids in determining its activation value. The activation value is calculated from the connection weights and transfer functions and then it is passed to the next node. These weights are how ANN decides where to pass the signals. Activation runs throughout the network nodes until the output node is reached. Finally, the output node transforms the information in a way we can understand [13].

The model performance can be calculated from the cost function since it associates the estimated output and the actual output. To minimize the cost function, weight adjustment is done which is referred to as backpropagation. Forward propagation alternatively, uses information entered in input layers and moves forward through the different nodes to produce output. Weight values are calculated and circulated backwards to update weighted nodes and finally, the network is trained [24]. A feedforward network is composed of input, hidden and output layers where the signal can only move in a single direction (forward). These kinds of networks are utilized in data mining. On the other hand, a feedback network allows signals to move in both directions. An example of such a network is the recurrent neural network (RNN) [11].

Inside the network node, we have an activation function. An activation function determines the output of a node. It can also be referred to as a transfer function since it translates the input vector to the output vector [24]. The function transforms the output

values in ranges of 0 to 1or -1 to 1. In simple terms, the activation function is the rate of action potential firing in the cell [25]. Common examples of activation functions include threshold activation function, sigmoid activation function, hyperbolic Tangent Function, and rectifier function [11]. The most popular models are selected using the criteria of the most used model and with the best outcomes from various published research like the review of deep learning models by Shrestha [8].

2.2.2 Selected Deep Learning Models in Video Anomaly Detection

The most popular deep learning models used to detect anomalies in surveillance videos can be identified from the frequency of use by researchers. For instance, a survey by [20] on Deep Learning for Anomaly Detection, identifies the following models as the most popular; Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), Auto Encoders, and Convolutional Auto Encoders [20]. The table below shows the models used in different research.

Deep Learning Model Used	**No. of Times Used**	**Referred use cases**
CNN	11	Dongetal.[2016],Andrewsaetal.,Sabokrouetal.[2016a],Sabokrouetal.[2017],Munawar et al. [2017],Li et al. [2017b],Qiao et al. [2017],Tripathi et al. [2018],Nogas et al. [2018],Christiansen et al. [2016],Li et al. [2017b],

AE-CNN-LSTM	3	Chong and Tay [2017], Qiao et al. [2017], Khaleghi and Moin [2018]
AE	14	Qiao et al. [2017],Yang et al. [2015],Chen et al. [2015],Gutoski et al.,D'Avino et al. [2017],Dotti et al. [2017],Yang et al. [2015],Chen et al. [2015],Sabokrou et al. [2016b],Tran and Hogg [2017],Chen et al. [2015] ,D'Avino et al. [2017],Hasan et al. [2016],Yang et al. [2015],Cinelli [2017],Sultani et al. [2018]
LSTM-AE	1	D'Avino et al. [2017]
LSTM	4	MedelandSavakis[2016],Luoetal.[2017a],Ben-AriandShwartz-Ziv[2018],Singh [2017]
RNN	4	Luo et al. [2017b], Zhou and Zhang [2015] ,Hu et al. [2016], Chong and Tay [2015]

Table 2.1: A table showing the usage of deep learning models used in video anomaly detection adapted from Chalpathy and Chwala.

The concept of anomaly detection encompasses other areas like fraud detection, cyber intrusion detection and other areas. This study however will only focus on anomaly detection in surveillance videos.

2.2.3 Convolutional Neural Networks – ConvNet

Convolution can be defined as a category of linear operation that is used for feature extraction. Convolutional Neural Network is a deep learning algorithm whose architecture borrows heavily from the human visual cortex, where neurons are arranged in patterns that overlay to span over the entire pictorial/graphic area. A ConvNet uses filters to acquire pattern dependencies in images. The architecture of ConvNet reduces the images

into small matrices that are easier to process but maintains the distinctive features that are important for acquiring a good prediction. A kernel is composed of a trivial array of numbers, which is used across the input array also known as a tensor. Element-wise multiplication between all parts of the kernel and the input array is computed at all parts of the tensor, then summed to obtain the output value. The output value is also known as the feature map. [25].

The building blocks of a CNN include the convolutional layer, Pooling Layer and Fully Connected Layer. The usual architecture of CNN consists of the replication of a stack of numerous convolutional layers and a pooling layer followed by a fully connected layer [26].

The convolution layer is the foundation block of CNN which performs the biggest portion of the CNN's computational load. The pooling layer reduces the size of the representation, therefore, decreasing the amount of computation and number of weights required. The max-pooling reports the maximum output from the neighborhood. Pooling offers some transformation invariance, which makes an object to be detectable irrespective of its location on the frame. The fully connected layer (FC): The nodes in this layer are fully linked with all the nodes in the previous and the next layer, which is like a regular feed-forward neural network. Hence, it can be calculated normally using matrix multiplication, followed by a bias effect. The FC layer is particularly useful in mapping the image from the input to the output [26].

This technology has been applied extensively in image processing and anomaly detection. For instance: a deep learning application known as AI Guardman uses pose estimation to

detect shoplifting. This application was developed by a tech startup called NTT East [27]. The application uses the OpenPose Technology from Carnegie Mellon University to estimate the pose of a person and suspicious behavior is derived from the pose. OpenPose was established by scientists at Carnegie Mellon University to approximate the posture of a person instantaneously. This model can identify a person's facial points, body and hands from 3D and 2D images as well. Open Pose technology utilizes multitask learning, using a Convolutional Neural Network. [28].

With the integration of Openpose technology, AI Guardman can identify and trail an object's motion and behavior. This application is trained to identify suspicious behavior like nervous customers and then it alerts the store clerk's smartphone and sends a mugshot and location [27]. Some of the challenges faced and still present in the system are the high error rates. The resultant application failed to distinguish shoplifters from indecisive shoppers or widow shoppers. On several occasions, the system had flagged, customers who pick up and put-back items, and salesclerks who restock shelves as potential shoplifters.

Sabokrou et al. [29] developed a method of detecting and localizing anomalies in videos aimed at detecting an anomaly in crowded scenes [29]. This model used Fully convolutional Neural Networks (FCN) and sequential data. Their model utilized a supervised pre-trained FCN to ensure the detection of anomalies in the scene. FCN extracts distinctive features present in the video regions. Initial layers of the FCN borrowed from a pre-trained Alexnet Model-which is a CNN model developed to classify

images and trained using the ImageNet (MIT Places Dataset). The extracted features are discriminative enough to pinpoint anomalies in video surveillance data [29]

The shortcoming of the model includes the error rate margin, where the model labels the motion of people going in different directions as abnormal behaviour [29].

2.2.4 3D Convolutional Neural Networks (3D- CNN)

Convolutions are filters (matrix/vectors) composed of learnable parameters that extract low-dimensional features from input data. Instinctively, convolution is the step of using the idea of sliding window (a filter with learnable weights) across the input and producing a biased sum (of weights and input) as the output [30]. The weighted sum is the feature space that is used as the input for the next layers. 3D-CNN is a type of CNN that utilizes 3D convolutions.

3D Convolutions applies a 3-dimensional filter across the input as it moves in 3-directions (x,y,z) to compute the depiction of the low feature. The output matrix is a 3D volume space, like a cube or cuboid. The 3D convolutions apply to video analytics like event detection in videos [30].

Kushwaha et al developed another smart surveillance system to detect intruders [31] This system implemented convolutional neural networks to detect motion and alert the homeowners. Depending on the level of motion, the system flags intrusion and then notifies the homeowner. This system pulls data/ live surveillance from a pi camera, then

it is forwarded to CNN to detect motion. Their system used Raspberry PI to process the footage using the TensorFlow library of deep learning [31]

CNN is used to detect motion, thereafter they flag intrusion and alert the homeowner. Although the use of deep learning improved the accuracy of the system. It is still vulnerable to false alarms which can be raised by non-threat intrusion [31]. Another challenge faced by the system is that motion detection is dependent on motion exceeding a certain limit, otherwise, the system cannot raise an alarm. With the rapid growth in complexity and increase of theft actions, their model could not effectively distinguish threats from non-threats.

Sabokrou et al. [32] developed a dependable method to detect anomalies in crowded scenes [32]. This model was composed of cascading classifiers with two main stages: light-deep 3-D spatial autoencoder used to identify the initial cubic normal patches and remaining video data is evaluated by a more complex deeper 3D-CNN. This model achieves comparable performance to the other similar hi-tech models. Although this model is only limited to crowded scenes rendering it less effective in generalized video surveillance and flags people moving in different directions as an anomaly [32].

2.2.5 Recurrent Neural Networks – (RNN)

RNNs are part of deep learning algorithms that are crafted to extract patterns in chronological(sequential) data. This type of network is appropriate for images that are broken down into a group of patches that are treated as a sequence. Within recurrent networks, decisions arrived at time step (t-1), affect the decision reached moments later

at time step t. The major shortcoming facing this kind of neural network is the vanishing gradient problem, which limits its performance. This led to the development of an RNN variation called, Long Short-Term Memory (LSTM) [10].

2.2.6 Long Short-Term Memory (LSTM)

Two German Scholars developed LSTM network namely: Sepp Hochreiter and Juergen Schmidhuber to solve the problem of exploding gradient. They introduced forget gates which preserve the backpropagated error through time and layers. This allows the LSTM network to learn the features over many cycles (over 1000), thereby LSTM opens a conduit to connect the causes and the effects remotely [9].

LSTM has gated memory cell where data is written and read from the cell, it is like how a computer's main memory works. The cells with help of forget gates make decisions on weights to store and when access the reads, writes and erasures, through gates that close and open. The forget gates were implemented through selective multiplication via sigmoid functions which range from 0 to 1. Moreover, these gates acted on signals they received and just like the neural network's nodes; they pass or block the signal depending on its strength and acquire features using their own set of weights [9].

The limitation of LSTM as to be applied in this thesis is that it only accepts 1-D data. For purposes of video analytics where a sequence of images exists, both feature extractor and sequence data modelling are needed. Hence, need to explore another variation of RNN called Conv-LSTM, which combines the capabilities of Convolutional Networks and the Sequential data processing capability of LSTM networks [10].

2.2.7 Conv-LSTM

Contrary to the LSTM Cell state, convolutional LSTM internal matrix multiplications are exchanged with convolutions operations. Hence, the data that flows through the conv-LSTM cell keeps the input dimension (3D in our case) instead of the 1D vector. This algorithm considers both spatial features and temporal features from input data. This characteristic of the Conv-LSTM makes it more suited for video analytics. Among the ways of processing sequential images, the best approach is using Conv-LSTM layers [14].

Conv-LSTM varies from LSTM since the number of input dimensions in a Conv-LSTM is different from LSTM. As input data accepted by LSTM is one-dimensional, which makes LSTM not suitable for 3-D chronological data such as radar image data, satellite, and video sets. Conv-LSTM on the other hand is crafted for spatial (3-D) data input and processing [14].

2.2.8 Autoencoders

Medel and Savakis [19], implemented a Convolutional Long-Short Term Memory Autoencoder Network (Conv-LSTM-AE) to predict the rate of change of video sequence from several input frames [19]. They applied that technique to measure regularity scores derived from the reconstruction errors of a set of predicted frames with abnormal video sequences. The abnormal video sequences yield low regularity scores since they diverge further from actual scores sequences over time [19].

The amalgamated Conv-LSTM-AE was composed of an encoder and a decoder which learnt the regularity of videos from the non-overlapping patches of the frames from an

input segment. The network was trained to predict accurately normal actions like those found in training videos. Hence, the prediction of abnormal videos diverges away from the ground truth with each time stamp. Reconstruction errors are measured using a regularity score which is used to determine when anomalies occur [19].

The architecture of the model is constituted of encoder and decoder parts. The encoding part accepts the input sequence and reshapes it to a stack of non-overlapping patches. The decoder part is composed of two decoders that is one reconstructs the past input video sequences and the other predicts the future frames. Both decoders are initialized with encoded input. The trained model with normal videos. Anomalies are identified by inspection of the reconstructed and predicted since anomalous events are more likely to stand out since the trained model was only trained with normal videos [19]. The challenge faced by this model is the rate of false-positive errors.

2.3 Other technologies used in Anomaly detection

This section highlights the use of motion sensors and behaviour tracking to detect anomalies. Other technologies that have been used to detect anomalies in surveillance videos are discussed in this section.

2.3.1 Use of Motion Sensors

Kushwaha's pioneer work on the anti-theft system used motion sensors and a video camera to point out cases of home intrusion [31] Their system used a video feed which was broken down to images, which were preprocessed through background estimation, background subtraction, outlier rejection, frame referencing and segmentation [31].

Several frames were used to estimate the background, then subsequent frames were subtracted from the reference frame to detect anomalies in the objects. The nearest neighbor search was then carried out to match previous and current images to pinpoint anomalies. This system utilized a simple decision-making algorithm based on feature matching. The first captured image was stored as the reference image, then if the current image did not match the referenced image, the system generated an alert message, where the user must determine the credibility of the alert if it is true or false and whether to take any action. This system is limited to one camera only and the motion generated from other objects other than intruders are picked as alerts. To address the shortcomings of this paper Kushwaha developed another antitheft system that uses Convolutional Neural Networks to detect motion [31].

2.3.2 Behavior Tracking based anomaly detection

[33] is a classic case of behavior analysis. Farooq et al. [33] implemented a system that utilizes unsupervised learning to detect anomalies in street traffic. This system puts together various algorithms and models. Gaussian Mixture Model (GMM) was utilized for the identification and tracking of objects. GMM was used to subtract background scenes and identification of foreground objects. In addition, Kalman Filter is applied to indicate each track. Features extracted from the live feed, include the position of the object, orientation, trajectory, visible age in the scene and invisible account [33].

The model also utilizes the Euclidean distance to analyze the trajectory and calculate the speed of the vehicles [33] From the trajectory's dataset, clusters are calculated. Thereafter Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm is

used to cluster events based on features like age and angle. For instance, anomalous activity like a sharp turn is identified when the object turning angle is greater than 56 degrees. Some of the contributions form the foundation for object tracking and detection of behavioral analysis. One of the challenges faced is that traffic is more complicated. Other than trajectories, enforcement of other traffic rules and continuous tracking of vehicle behavior is not addressed [19].

2.4 Model Development Strategies

This section explores the literature related to model development and enhancement. Different model design techniques are influenced by the nature of the problem. The art of combining different algorithms to work as a single unit is widely explored.

To understand the strategies used in the model development process, it is critical to understand what constitutes a model and the relationship between algorithms, models and the training data. A deep learning model can be described as the outcome of training a deep learning algorithm with the training data. The model is the product of the training process [34].

On the other hand, deep learning algorithms are the general approaches to problem-solving that borrows heavily from artificial neural networks for instance: CNN, LSTM and ConvLSTM. The algorithms can be combined and implemented to have different models. The model becomes the mathematical representation of the solution using a specific pattern that can be applied to solve real-world problems. Many models can be created from the same algorithm using different sets of training data [34].

2.4.1 Model Development Process

The development of deep learning models involves a myriad of activities which can be summed up into several steps. The deep learning model development process is somehow like the software development process. The development process involves; the initiation of the deep learning project, preparation of the datasets, design of the model, visualization of the model and metrics, debugging of the model and finally the improvement of the model [34].

Initiation of a deep learning project involves understanding the problem at hand and defining the expected solution. At this stage, researchers define a plan on how to achieve the project objectives by defining the acceptable solution, ethical reservations, the acceptable precision/accuracy score, and the nature of the problem i.e., classification, regression, and clustering among others. The platforms and deep learning frameworks are selected and optimized according to the project requirement and budget constraints. For instance, for deep learning, a platform with a graphics processing unit (GPU) is required. Cloud infrastructure provides scalable computing resources that are extremely useful in deep learning projects. For instance, Amazon or Google Cloud [34].

The data preparation phase includes the creation of the training and testing data needed to craft the model. After the problem is understood, the next step is to find the data that will be used. The quality and the quantity of data are important in deep learning. The preparation process involves identification of the data, cleaning of the data, preprocessing and data transformation to the required shape. Data can be collected from public datasets that have created specialized data for use in our problem domain. For instance, the

University of California San Diego (UCSD) has shared a video anomaly dataset called Pedestrian 1 (Ped1) and Pedestrians 2 (Ped2) [35]. Other popular datasets include the Shanghai Tech Video Anomaly dataset and the Avenue video anomaly dataset.

The design of the deep learning model begins with the identification and selection of the software framework or platform to be used. The deep learning frameworks include TensorFlow, Keras, Caffe, PyTorch, Caffe2, Apache MXNet and Microsoft Cognitive Framework (CNTK) [34]. The most popular frameworks include TensorFlow and PyTorch. The popularity of PyTorch can be credited to its user-centered design and user-friendliness features. Its internal organization makes pre-trained models and popular datasets easily accessible. On the other hand, TensorFlow is more popular due to its big developer community that offers ready support and answers that guides the deep learning researchers to avoid pitfalls. TensorFlow blends well with other intuitive Application Programming Interfaces (APIs) and frameworks like Keras making it a popular choice for many [36].

The next step is to visualize the model and the metrics. It involves a graphical representation of the model performance and the input and output data to understand the training process and easily debug it [37]. Deep learning requires video data to be scaled from -1 to 1 by dividing the pixel values. A plot of such values is applied in counter-checking that the data is between the range of -1 to 1. Scaled data ensures the network, does not suffer from exploding gradient problems. Loss and accuracy of the training and validation process are plotted to aid in the tracking of the model performance. The loss

plot is useful in tuning the learning rate, while the accuracy plot is useful in tuning the regularization factors.

Debugging and improvement of the model are important in achieving the goals of the model. Jonathan Hui, advocates for a rigorous debugging process that starts by overfitting the model with a small amount of the training data and monitoring its training loss if it significantly drops after 5000 iterations [38]. If the model loss drops, incremental modifications to the model are suggested to ensure model depth. After model depth is added, training with more data is advised and additional regularizations to control the overfitting of the model.

2.4.2 Model Design Considerations

Although the model design is a part of the model development process, it requires more exploration to uncover separate ways researchers combine and use deep learning algorithms to make well-performing models. Some researchers use simple and incremental design principle that dictates starting simple and building more into the model [38]. Below are some of the design considerations that every researcher considers achieving better performance in deep learning.

2.4.2.1 Use of Cost Functions

The design of deep learning models requires an understanding of the cost functions since they affect the optimality of the solution. Cost functions are used to estimate the error between the predicted values and the expected values. The returned values are called cost, error or loss and can be applied to check optimal model parameters since at minimal error

model performance is high. Some examples of the cost functions include the mean absolute error (MAE), mean squared error (MSE) and Cross Entropy. The MSE has good mathematical properties that make it a better choice over the Mean Absolute Error (MAE) since its derivative is easier to compute [36].

2.4.2.2 Scaling of Inputs

Deep Learning models use the training dataset to map the input and the output. Usually, the training sets have the X and Y variables, with X as the input and Y as the expected output [36]. The weights of the model are initialized to small random values and are adjusted through an optimization algorithm during the training process. The size of the input and output should be aligned to the small weights to lower the error [39].

Unscaled input variables result in an unstable and slow training process, while unscaled output on regression problems can result in exploding gradient which makes the learning process halt at some point. To deal with such design requirements, deep learning researchers use standardization techniques to scale the input and output between 0 and 1 [36].

2.4.2.3 Batch Normalization

The problem of unbalanced nodes at the layer output before the activation function. To smoothen the training process, it is advisable to normalize the output of the nodes. Batch normalization is therefore applied to the Convolutional Neural Networks (CNN) to normalize the outputs [36]. Batch Normalization computes the mean and the variance of the spatial location, and it uses the same mean and the variance to normalize the node

output at each location. In Recurrent Neural Networks (RNN) layer normalization is used instead [40]. Layer normalization is different in the sense that it calculates the mean and the variance at every layer, which is applied to normalize the outputs of layer nodes.

2.4.2.4 Activation functions

Choice of activation functions has been noted to affect the performance of the models hence, activation functions should be part of the interest in designing the deep learning models. Activation functions take the output signal from the previous cell as input and convert it to a form to be used as input to the next cell. For instance, Rectified Linear Unit (ReLu) introduces non-linearity and Leaky ReLu replaces zero values with some predefined value [38].

It has been found that non-linear activation functions are preferred in the deep learning design since they limit their values to some range, hence preventing computational overload. The most desirable feature of the activation functions in deep learning is the introduction of non-linearity. Complex problems require a higher degree of complexity to learn non-liner patterns [38].

Good activation functions should not shift the gradient to zero in deep layers to avoid the vanishing gradient problem. These activation functions are zero centred and symmetrical at zero to avoid shifting of gradients. The activation functions should be computational inexpensive and differentiable in the gradient descent process.

Some examples of non-linear activation functions include sigmoid, softmax, tanh, ReLu and LeakyReLu. Tanh and Sigmoid functions have been found to cause enormous

vanishing gradient problems and should not be used in deep learning [38]. On the other hand, ReLu is good for a start, and it can be replaced with LeakyReLu if the dying ReLu problem (model stops learning) is encountered [38].

2.4.2.5 Checkpoints Design

The incorporation of checkpoints in deep learning model design is an important design consideration to enhance model scalability. The trained model epochs are saved to be reloaded later. The saved checkpoints can be compared and the best load. This model design paradigm allows the model to be trained continually even after the initial training [36].

2.4.2.6 Addition of Custom Layers

In some cases of deep learning, custom made layers can be added to the model. Some reasons that influence the creation of user-defined layers include unit testing of the forward pass and backpropagation processes. Researchers, with intention of introducing custom computational operations to the model, can add custom layers to the model [36].

2.4.2.7 Optimizers

Optimizers are the algorithms used to alter the attributes of the deep learning network. Optimizers alter attributes like the learning rate and the learning loss in the deep learning network. Optimizers can change the weights in the neural network. Some examples of optimizers include the Gradient descent that is used in linear regression, Stochastic

Gradient Descent, Mini-Batch Gradient Descent, Momentum, Nesterov Accelerated Gradient, Adagrad, AdaDelta and Adam [39].

Adam is the best optimizer for use in deep learning since it maintains a learning rate for all parameters and adapts them separately as learning unfolds. Adam optimizer has four parameters i.e., learning rate, the exponential decay rate for the first moment estimation, the exponential decay rate for the second moment estimation and finally € denotes a small value that replaces zeros to avoid division by zero [39].

2.4.3 Model Improvement Strategies

Deep learning models do not achieve perfect accuracy, but they can improve through rigorous tuning and debugging processes to achieve comparable performance to the state-of-the-art models or baseline models. The improvement process involves the systematic analysis of the model structure to find areas of weakness. The model structure can have areas of weakness in the model depth, dataset quality, model regularization and activation functions.

2.4.3.1 Improvement of model capacity

Deep learning networks can be added with more layers to increase the learnable parameters which in turn makes the model extract more features from the data. Some of the considerations while model depth is being increased include the addition of small filters since the small filters like 3x3 and 5x5 work better than larger filters [41].

The process of tuning models is purely empirical. That means the improvement is open experiments with minimal known outcomes. The experiment is aimed at overfitting the model by having, a deeper network that extracts more features from the model. Later the overfitted models are toned down through regularization and dropout. The regularization involves the introduction of layer normalization and dropout functions. This process is repeated until the model improves its accuracy [41].

2.4.3.2 Dataset Collection and Clean-up

When analyzing the model with the intention of improvement, analysis of the dataset is important. It has been noted that analysis of the false errors and true errors (bad predictions) can be traced back to the low-quality dataset. If bad predictions are caused by the dataset, it is advisable to preprocess the data or use a variety of datasets [41].

The collection of samples has serious effects on the model accuracy. For instance, if an image dataset was to be used in a deep learning model, high-quality images are recommended and filtering out the unwanted data in those images will increase the model accuracy.

Complicated scenes in imagery and video datasets call for deeper convolutional networks with smaller filters to untangle scene complexity. More data should be used to train the model since deeper models have more trainable parameters which require a lot of data. Data size threshold can be achieved through, the collection of data with variety and the creation of variations of the same data through transformations such as reflections, zooming and others. Therefore, the use of data augmentation is advised [41].

2.4.3.3 Learning Rate Tuning

While debugging the learning rate of the model, the non-critical hyperparameter can be turned off or initialized to zero. In some cases of deep learning, the default learning rate works well. However, depending on the nature of the data and the model, the learning rate might need some tweaking [41].

For instance, the Adam optimizer has a default learning rate that gives rise to high model performance. Learning rate should be among the last model improvement attempts after the other parts of the model have been perfected and the model training loss has failed to drop [39].

The usual learning rate is from 1 to $1e^{-7.}$ Therefore, the best practice for tuning the learning rate is reduction or increment of the rate by factors rate of 10. Deep learning engineers recommend dropping the rate gradually with close monitoring of the model loss. It is notable that when the learning rate increases the training loss goes up consistently and vice versa [39].

Other hyperparameters that can be tuned during model improvement include mini-batch size, regularization factors and layer-specific hyperparameters like the dropout. Mostly, mini-batch size assumes either 8, 16, 32, or 64 values. However, it has been noted that a small batch size has the smoothest gradient descent. Therefore, for models that take a long training time, small-batch size should be used to ensure that, learning oscillations are shorter and there is less training loss [11].

2.4.3.4 Model Regularization

The gap between the validation and training accuracy can be minimized through the improvement of the validation accuracy which is usually achieved by tuning down the overfitting within the model [42]. Model regularization mechanisms reduce the overfitting within the model. Some examples of the model regularization include Dropout, Sparsity, $L^{1,}$ L^2 Activation Functions, Layer Normalization, and others.

Model regularization seeks to solve the problem of model generalization, which is the ability of the model to perform well on new input test data. Regularization methods put constraints on the model intending to introduce restrictions to the parameter values to limit the capacity of the model [43]. A group of regularization mechanisms use parameter restrictions while others add objective functions ($\Omega(\theta)$) which introduces a soft constraint on the parameter values. The objective function can be introduced in neural networks.

Commonly used forms of regularization include L^2 and L^1 regularization. L^2 is commonly known as the weight decay since it pushes the weights toward the origin by addition of the regularization term $\Omega(\theta) = \frac{1}{2}||w||^2_2$ [42]. L^2 makes the model use all its inputs rather than leaving some of the inputs. It shrinks the weight vector by a constant factor before gradient update. On the other hand, L^1 introduces the sparsity property by making the weight vectors awfully close to zero. At zero weight vector, it implies that the corresponding features were discarded. Hence, reduction of overfitting [42].

Other than tuning the model, the improvement process can involve the use of different deep learning algorithm variants or the introduction of new algorithms within the model.

A model that has Convolution Neural Network as the building block can replace the ConvNet with 2DConvNets or 3DConvNet to extract more features depending on the nature of the data. For instance, a Convolutional Neural Network-based image generator can incorporate time sequence by adding the recurrent neural network to the model i.e., LSTM or ConvLSTM [41].

2.5 Deep Learning Models Evaluation and Validation

In similar research work, researchers use the systematic review to determine the best deep learning models in anomaly detection. Different models are analyzed, and their accuracy scores are tabulated and compared to reveal underlying patterns. Experiments are set up using the selected deep learning algorithms then several parameters like training and performance are evaluated. For instance, an empirical study that evaluates deep learning networks used in anomaly detection utilizes various metrics. Some of the measures used in these papers include training time and model accuracy to evaluate training complexity, and F-measure (F1-Scores), which are used to estimate the performance of the model. F-measure combines precision and recall which are particularly useful in measuring the accuracy of classification. Another important metric used is Mattew Correlation Coefficient (MCC)which is used to estimate the quality of binary classification. The coefficient value is interpreted -1.0 (poor), 0.0 (random) and 1.0 (good) [44].

[45] describes how to evaluate the performance of the deep learning models through confusion matrix, accuracy, precision, specificity, F1 Score, Precision-Recall or PR curve and Receiver Operating Characteristic (ROC) curve. Terms introduced here include, True Positive (TP) which indicates the predicted positive is positive, and False Positive (FP)

which indicates that the predicted positives are negative. In addition, True Negatives mean that the projected negative is negative and finally the False Negatives indicate projected negative is positive [45]. Limited research has been conducted on the assessment of deep learning models used in the identification of anomalies, specifically in surveillance videos.

Better evaluation models as indicated by Nighania [49] include **Precision** which is calculated by the percentage of the total predicted positive instances. It answers the question, '*What percentage is the model right when it is saying is right*'. It is given by True Positives divided by the total of all positives. **Recall/ Sensitivity/ True Positive Rate** measure describes the percentage of positive instances (TP) out of the total actual positive instances (TP+FN) in the dataset. The measure determines, *'the number of right outcomes the model missed.* **Specificity** is another measure that calculates the percentage of negative instances (TN) out of the actual negative instances (TN+FP) [45].

Recommended metrics by Nighania include the PR (Precision and Recall) Curve, which is a plot of precision and recall for various threshold values. The top right part of the curve portrays an ideal space where we get high precision and recall. The choice of predictor and threshold values are dictated by the type of application. ROC Curve (Receiver operating characteristic) is plotted against True Positive Rate and False Positive Rate. ROC AUC is the area under the curve and the higher its numerical value the better. True Positive Rate (TPR)= Recall = while false positive rate (FPR) =1- Specificity = [45]. This literature provides ways to evaluate and validate the models which played a major part in our research.

2.6 Conceptual Framework

The literature review found the following relationships among the ideas within the research study. The reviewed theories revolved around the model architecture, dataset quantity, quality of data preprocessing and the model prediction accuracy. Figure 2.1 below illustrates the conceptual framework.

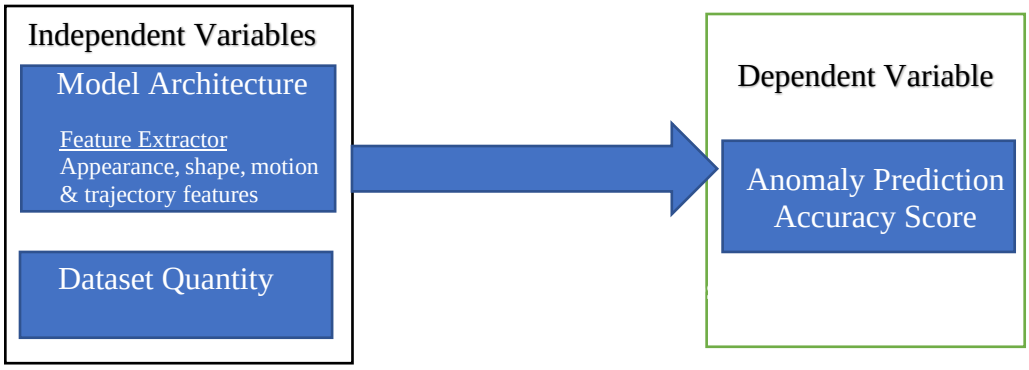


Figure 2.1: Illustration of the conceptual framework

2.7 Summary

In conclusion, the literature review identified the main gaps: First the literature has shown immense growth in the detection and identification of abnormal scenes like violent scenes within surveillance videos. Deep learning models have been used to detect anomalies. Yet there is still a challenge in identifying which is the best model among all those implementations. The research gap exists in finding out empirically the best deep learning model for use in anomaly detection specifically in surveillance videos.

Secondly, although the best performing deep learning models used to identify and localize anomalies include 3D-CNN, Conv-LSTM (Composite Neural Network) and Conv-LSTM (auto-encoder). Although they have achieved quite commendable performance, we have seen that there are still errors present while detecting anomalies in surveillance videos. A research challenge exists in improving the accuracy of the deep learning models.

CHAPTER THREE

METHODOLOGY

3.1 Introduction

This chapter describes how the research itself was conducted. Research methodology links theory and the data for purposes of analysis thereby gaining insights into the research problem. Strategy and methods are well aligned for the research study.

3.2 Research Design

The research design that was used to conduct the research was a systematic literature review and experimentation. This chapter explores the two research methods extensively.

3.2.1 Systematic Literature Review

To gain full insight into the research problem and form a basis for a solution. A systematic Literature Review was used to gather the relevant knowledge to conduct the study. In the first objective, a review of the deep learning models used in video anomaly detection was conducted. Opensource papers published since 2016 in this area were analyzed and the available deep learning models were identified and ranked. The review discovered the underlying technologies and trends in video anomaly detection using deep learning.

3.2.2 Experimentation

Experimental Research Design applies a scientific approach to the research problem by allowing variables to be manipulated and their effects on other variables to be measured

[46]. This research strategy is common in deep learning research due to its nature of allowing the comparison of different models with a forthright logic.

True experimental research design was adopted. True experimental research design relies on statistical evaluations to approve or disapprove research hypothesis. True experimental design established a cause-and-effect relationship within the study. This research design required the study be formed with a control group which was taken as the model before enhancement. True experimental design also required an independent variable. In the study, the model depth (learnable parameters) is taken as the independent variable.

A pre-test posttest-only control group design was used. The model's accuracy was tested before and after the enhancement. The control group was composed of the test cases before the model was enhanced while the experimental group was composed of the enhanced model test cases. The accuracy of both control and experimental groups were tabulated and their differences are analyzed further.

Our research problem required a comparison of the model before and after experiment to point out the impact of the improvement done to the model. The experimental design was best suited for the problem at hand. Other researchers have applied the same methodology to similar problems. For instance, [47], [48], [22], [49].

Two classes of the experiments were set up. The first set of experiments was used to investigate 3 chosen models from the reviewed models. The second set of experiments was used to improve the selected model and validate the improved model. Experimentation allows the comparison of the models by feeding the different models

using the same dataset and running the model using similar computing resources. That allowed the performance and effectiveness of the models to be compared. Similarly, while improving the model, experimentation was used where model parameters of the selected model were optimized to increase accuracy while the existing model was used as the control group to establish the improvement done.

To identify the most effective deep learning model for the identification of anomalies in surveillance videos, Multiple Instance Learning [22], Spatial-Temporal Autoencoder(STAE) [50] and Generative Adversarial Network [51] were implemented to determine the most appropriate for use in anomaly detection of surveillance videos.

A comparative empirical study was conducted to analyze the effectiveness models in identifying anomalous activities. The models indicated above were then trained and tested while various metrics like precision, specificity, ROC curve and PS curves were used to provide empirical evidence of the best model.

The selected deep learning model was enhanced through various improvements to the model. The enhancements included optimization of the model hyperparameters and systematic code review.

The final part of the study involved validation of the improved model, a comparative study through the ROC curve and Precision and Recall (PR) curve. The plot of both models, old and improved can provide enough validation evidence of the improvement by running the model across 2 validated datasets.

The datasets were downloaded from public datasets like the UCF database of real-world anomalies [22], UCSD database of crowded scenes on a sidewalk [35], Shangai Tech database of staged sidewalk surveillance and some of the codes borrowed from GitHub.

3.3 Design of the Experiment

Experiments were set up in Google Collab and Google Cloud due to the computation power required. Google Collab offered ready to deploy platform which was especially useful for writing code and debugging the code before large-scale deployment. Since Google Collab is free it allowed stepwise debugging and incremental enhancement of the model. For lengthy training and testing, the Google Cloud platform was used due to its stability and infrastructure scalability. Google cloud offers infrastructure as a service and scalable computing resources on-demand, which are quite suitable for the running of deep learning experiments.

Python-based frameworks and libraries were used for the experiments due to their wide documentation and suitability in deep learning libraries like Keras and TensorFlow. In addition, it has shown the best results while solving data science problems.

The model improvement was done using Python 3.7. The hybrid model was crafted under Anaconda Development Environment, which puts together all frameworks needed like OpenCV. TensorFlow, Keras. Matplot, FFmpeg and sckit-learn libraries. The libraries were configured to work in jupyter-notebook for interactive coding in both Google Collab and Google Cloud.

The projects were set up and run through the Jupyter notebook integrated development environment (IDE). Divergent functions were integrated from input, processing and output programs. Both training and anomaly prediction accuracies were recorded for comparative study. A common dataset was selected from real-world anomaly datasets which was used for training and testing the selected models and then the most suitable model was selected for enhancement.

The data needed during the research was surveillance videos that had diverse types of real-world anomalies like violence, burglary, weapons, running, crowding and others which are unusual. Since a comparison among multiple models was done, a large, diverse, and balanced dataset to reach a convincing conclusion was used. Purposive sampling was used to select existing datasets from public deep learning research centers which are available online [52]. Purposive sampling was adopted to ensure the datasets used by the old models were the ones used to train the improved model. Purposive sampling was considered due to its non-random criteria. Purposive sampling is referred as judgement sampling was due to its ability to base sampling on the researcher judgement. Researcher had the power to select the most information rich datasets [52] that benefited his study. Homogeneous sampling technique was used to acquire a homogeneous sample of dataset that contains only surveillance videos. The sampling technique focused on the datasets of that were used in analyzed papers and the improved model. A video dataset that was large enough for deep learning and surveillance anomalies rich videos were considered. For instance, for deep learning models' a big dataset is required for training and testing.

Some of the tools which were used to extract frames from the videos include the FFmpeg video library. FFmpeg library is a code library that allows manipulation of the videos and breaking down of the videos to get images. Only a computer with the internet was used to download and prepare the dataset for use in model training and validation.

Recently released large-scale real-world anomaly detection benchmark, UCF (University of Central Florida) Crime, UCSD (University of California San Diego) Ped1 and Ped2 [39] were used to evaluate our model improvement [55]. These datasets consist of 1900 real-world surveillance videos, half of which contain anomalous events and the other half normal activities. For the anomalous videos, violent scenes, intruders in the park, commotion, and running were the majority anomalies. The training split and testing split were based on the model architecture since some models required to be trained with only normal videos and tested with anomalous videos.

3.4 Research Procedure

Implementation of study can be categorized into three main phases namely: investigation of chosen deep learning models to determine the most effective, development of an improved deep learning anomaly detection model and finally validation of the developed deep learning model.

The research procedure can be summarized in a simple diagram. Figure 3.1 above, illustrates the research procedure. The arrows indicate the flow of activities between the objectives and the activities. Figure 3.1 indicates major portions of this research.

3.5 Data Analysis

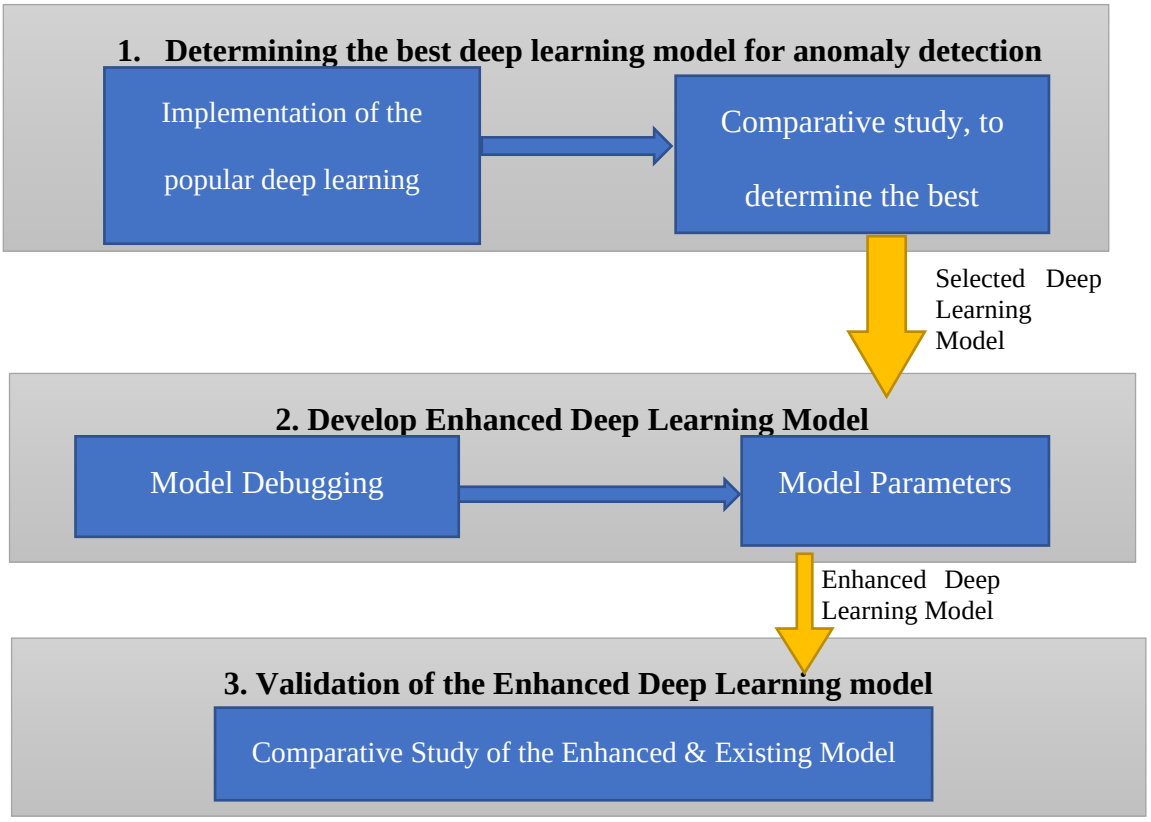


Figure 3.1: A Summary of the Research Procedure

Data analysis describes how the data was collected from the experiments and how the data was processed to gain some insights into the research problem.

3.5.1 Deep Learning Models Evaluation Experiment

Evaluation of the deep learning model's accuracy utilized the following measures: PR (Precision-Recall) Curve, which is a plot of precision and recall for various threshold values. The top right part of the curve portrays an ideal space where we get the highest accuracy and ability of the model to remember.

ROC (Receiver operating characteristic) will be plotted against valid positives TP Rate and invalid positives FP Rate. ROC AUC is the area under the curve and the higher its numerical value the better.

True Positive Rate (TPR)= Recall while,

False positive Rate (FPR) =1- Specificity [45].

The accuracies scores of the evaluated models were recorded and tabulated. Their accuracy score in different datasets was averaged for ease of comparison.

3.5.2 Validation of Enhanced Model Experiment

The best-selected model was improved, then trained and validated. Then, both training and testing accuracies were recorded for the enhanced model and control model. The accuracy values were captured in a table for purposes of comparison. To test the model's effectiveness, the video-based ROC curve and corresponding area under the curve (AUC) were calculated to evaluate the accuracy of the model [53].

The area under the ROC curve (AUC-ROC) is an independent metric of the model accuracy [53]. The Receiver Operating Characteristic (ROC) curve is the plot of sensitivity versus specificity. Specificity can also be referred to as the false positive rate while sensitivity is the rate of the true positive. Then to find a single value to indicate the model performance, the area beneath the curve commonly known as (AUC) is calculated. AUC is the ratio of the area underneath the curve and the total area. The AUCs of the ROC curve shall be tabulated together for comparison.

After accuracy was tabulated, a test of significance was conducted to establish if the enhancement was statistically significant.

3.6 Summary

This chapter has described the procedure that was followed to conduct the research. In summary, it involved evaluation of the popular deep learning model, implementation, and improvement of the selected model. It also described the model evaluation metrics that were used to validate the enhanced model.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Introduction

This chapter is dedicated to the implementation of the study objectives. The first objective was to investigate empirically the deep learning models used in anomaly detection in videos to determine the best. The chapter starts by introducing the deep learning model architectures, published models since 2016 that were reviewed and tabulated. The considered models are implemented, and their performance is compared with the published accuracies and the best pure deep learning solutions are selected. The second objective was to improve the selected deep learning model for the discovery of anomalies in surveillance videos. The third objective was to validate the improved model. This chapter also describes the data preparation process, the improvement process, the outcome and finally the validation process.

4.2 Deep Learning Models

The main purpose of the study is to enhance deep learning anomaly detection models using depth tuning. It is important to understand what a deep learning model entails and the distinctive design architectures. Deep learning models are composed of neural network variants that are multilayered. The architecture of deep learning models is extremely flexible since the models can differ in the number of layers, filter size and dimension, as well as the basic constructs. Models used in the detection of anomalies in videos have Convnets, ConvLSTM and 3DCNN as the basic building blocks [20].

A layer within a deep learning model is composed of interconnected nodes(neurons). A node may be connected to all other nodes in the adjacent layers or not. Data fed through the model goes through each layer and is transformed into an abstract representation also known as extracted features. The training process sets the weights across different transformation functions. Then the model modifies the weights using backpropagation where the output is traced back to the input modifying the weights.

Researchers have utilized this knowledge to design models using deep learning frameworks like PyTorch, Keras, and TensorFlow. A model can be composed of different deep learning algorithms. Different deep learning algorithms are stacked together to produce a model. For instance, a model may have Conv2D, and ConvLSTM, layers combine to get a model that works for sequential problems using high dimensional data like videos. Problem nature inspires the model design [54].

Hybrid models combine several deep learning models, which are optimized to work together. In some cases, like Sultani multiple instances learning model [22], it combines a pre-trained feature extractor and another model to rank an anomaly score of the bagged normal and abnormal scenes. Some of the pre-trained models that have been used extensively to extract features from videos include the Facebook C3D Model [22], Inception v3 [55], YOLOV3 and open pose technology from Carnegie Melon university [27]. These models are used to extract features like, motion, objects and even body posture. Those features are transferred to another model that is trained to identify anomalies in the features.

4.2.1 Review of the Deep Learning Models in Anomaly Detection

A review of deep learning solutions for anomaly detection in surveillance videos, published since 2016 was conducted. Open-source publications were considered, due to their unlimited availability. The systematic review was conducted in Google Scholar, Science Direct, Elsevier Journal Finder and ACM digital library due to their extensive ability to list the publications from peer-reviewed journals.

The review aimed to provide a holistic map of the published deep video anomaly solutions, track their growth and the trends in this active research area to rank the best models in the area. Intelligence surveillance is still an active research area due to its extensive application in security.

The nature of the reviewed models includes supervised, weakly-supervised and unsupervised learning. Supervised models are somehow limited to specific anomalies since anomalies are many and are subjective. For instance, some models are limited to traffic anomalies [33], home intrusion and violence [31]. Supervised solutions have failed to generalize the anomalies since the definition of anomalies is subjective and labelling of all anomalies is complex since anomalies are rare within videos. For instance, within a video clip, only a few frames have anomalies.

The weakly supervised class of the models utilizes small, labelled data while the rest of the data is unlabeled. This is a very common trend in deep learning especially in hybrid solutions [47] [56] [57]. The hybrid models contain two or more deep learning models. In some cases, the hybrid solutions are typical cases of transfer Learning.

4.2.1.1 Transfer Learning

Transfer Learning describes the handover of the knowledge from one model to another. This approach uses an already pre-trained model to solve a different task. Transfer learning is useful when there is a scarcity of data or computational resources since it allows the models to use less data by re-using the learned weights from the pre-trained model.

Transfer Learning was found as a growing trend in video anomaly detection and deep learning. One strategy of implementation of transfer learning through feature extraction. Pretrained models were used to extract features from labelled video and imagery data. The pre-trained models used mostly in the reviewed models include Facebook C3D Model [22], I3D [55], and YOLOV3.

Facebook C3D borrows from BVLC Caffee which was modified to support 3D Convolution and pooling [17]. C3D model was trained by Facebook Researchers on sports videos to extract features from videos. Which can be useful in down sampling the dataset for effective processing. The pre-trained model has been used in various models. For instance, [22] utilizes C3D for feature extraction in their paper. The model is set to input a video and then it extracts a tensor of 4096 features.

The use of pre-trained models to extract features from videos appears as a growing trend in anomaly detection research. Other feature extractor models that were found include Inflated 3D, which is a pre-trained model on the Kinetics-400 dataset [58]. It extracts

features from videos and gives an output of shape 1024. By default, frames fed should be of size 224x224 and video to be recorded at 25 frames per second (fps).

Another important feature extractor used by researchers [59], [60] is You Look Only Once Version3 (YOLOv3) which is a deep convolutional neural network that identifies specific objects in videos or images. YOLOv3 is an improved version of YOLOv2 that borrows heavily from the DarkNet model that was trained on Imagenet. YOLOv3 combines two 53 layers of Darknets to form a deep 106-layer network [61]. Object detection in the model happens within three separate locations. First Detection happens at the 82^{nd} layer that uses a 1x1 kernel, the second detection happens at the 94^{th} layer that uses a 2x2 kernel and the third detection occurs at the 106^{th} layer that uses a 2x2 kernel. The model also predicts bounding boxes on the objects and draws them around the objects and labels the objects. This detector was used to extract objects from videos which were used to define anomalies and normal scenes, on which anomaly detection was based on.

Transfer learning was identified in the following papers, Sultani [22], that used C3D pre-trained model combined with a light classifier to assign a ranking score for the normal and abnormal instances. The C3D model was used for feature extraction. Motion and trajectory features were extracted from the real-world UCF crime dataset.

[62] used pre-trained CNNs in anomaly detection. Nazare [62] explored several CNN networks including VGG-16, ResNet-50, Xception and DenseNet-121.Their paper [62] investigated the role of pre-trained image classifiers in feature extraction to solve the problem of anomaly detection. The paper found that the Xception model outperforms its counterparts, and it can be used for features extraction even though the whole idea

performs poorly compared to other anomaly detection methods. Other examples of transfer learning found in the review include [63], [48], [60], [59], [58], [64], [65], [51].

4.2.1.2 Autoencoders

Autoencoders are a substantial part of the survey, out of 30 papers reviewed, 11 papers were found to have used the autoencoder model design paradigm. Which is around 36% which is significant statistically. Thus, autoencoders can be considered a growing trend in video anomaly detection. Autoencoders are widely used due to their unsupervised nature, and ability to learn without human supervision or labelled data. The golden idea behind autoencoders is the reconstruction error that arises after when reconstructing the abnormal frames. The reconstruction error of the irregular videos is larger than regular videos. This idea is applied in designing models that detect anomalies in videos.

The autoencoders found in the review have different architectures and deep learning algorithms. For instance, [66] integrates a Conv-AE and Inception Module to form a deep autoencoder that detects the appearance and motion features from the videos. The decoder part of the model has two units that are dedicated to motion and appearance.

Duman and Erdem autoencoder [56] is composed of Convolutional Autoencoder and Convolutional LSTM. This framework uses Optical Flow to extract features of speed and trajectory from the videos. The optical flow output is fed to the autoencoder which returns the reconstructed optical flow map. The reconstructed output is subtracted from the input to acquire the mean squared error that is used to calculate the regularity score that indicates the abnormality level of every frame. [67] implemented an unsupervised

solution for anomaly detection in crowded scenes that was based on autoencoder design. The model was constituted of Conv-LSTM. Raw image sequences and edge image sequences were used to train the model.

Spatial-Temporal autoencoder [68] is another variation of the autoencoders encountered in the review. This model was made by [69] in their paper named Spatio-Temporal AutoEncoder for Video Anomaly Detection. Their model is composed of 3D convolutional layers. The architecture of the network is made up of an encoder and two decoder branches. The decoder branches consist of the prediction branch and reconstruction branch. The two branches are used to create the prediction loss function and reconstruction loss function that are used to estimate the regularity score for anomalies locating.

Pawar and Attar autoencoder is hybrid of convolutional autoencoder and LSTM autoencoder [21]. This presents another design paradigm of combining two different autoencoders to create a seamless model. The convolutional part takes care of the image part while the LSTM preserves the sequence. Reconstruction error is used to model the regularity score.

Variational Autoencoder is an improvement of autoencoders that employs the use of probabilistic modelling to select the best reconstruction from the latent space. Unlike the normal autoencoders that encode the latent space as a single point, variational autoencoders generate their latent space as a distribution. [70] exploited this architecture to create a two-stream variational autoencoder to detect anomalies in both local and streaming videos.

Other unique autoencoders found include Bhakat and Ramakrishnan [71], Mahmudul Hasan et al. [72], Sabokrou and Fathy [49] and another case of Spatio-temporal autoencoder by Chong and Tay [50]. The Spatio-temporal autoencoder is different due to its building constructs. It employs time-distributed layers wrapped in conv2d layers for the spatial part and convlstm2d for the temporal part.

4.2.1.3 Ensemble Learning

Other reviewed models are random cases of ensemble learning that combine multiple learning algorithms to get better predictive performance than the constituent learning algorithms alone. For instance, Zahid et al. [58] is a typical case of ensemble and transfer learning. The model combines a 3D convolutional network and a Fully Connected (FC) Network [58]. [73] is another case of ensemble learning that combines Conditional Generative Adversarial Networks, R-CNN and Support Vector Machines (SVM).

4.2.1.4 A Summary of the Deep Learning Solutions

A summary table of the models reviewed their learning technique and underlying deep learning algorithms are illustrated in table 3:

Table 4.1: A Summary Table showing the models reviewed in the review.

Publication	Learning Technique	Deep Learning Algorithm/Models	Datasets	Overall Accuracy
Chong and Tay [50]	Auto-encoder	ConvLSTMAE	UCSD Ped1, UCSD Ped2	87%
Sabokrou et al. [49]	Auto-encoder	Sparse AE & Non-Sparse AE	UCSD Ped2 & UMN	90.8%
Hasan et al, [72]	Fully Conv Feed Forward Auto-encoder	FC Convnet AE	UCSD Ped1 UCSD Ped2	83.18%

			CHUK Avenue	
Chalapathy et al. 2017 [74]	Robust PCA	PCA	Cifar10	89%
Sultani et al. [22]	Multiple Instance Learning (MIL)	C3D, FC Convnet SVM Classifier	UCF Crime Dataset	75.41%
Nguyen [75]	Generative Adversarial Network	GAN- Generative Adversarial Network	AI City Challenge	91%
Xu et al [70]	Variation Auto-encoder	2 stream Variational Autoencoder (VAE) / GAN	-	-
Doshi and Yilmaz [76]	Continual Learning	YOLOv3 KNN -K- Nearest Neighbors	UCSD, Avenue, Shangai Tech	85%
Kavikuil and Amudha [77]	Feature Learning	CNN	-	-
Liu et al. [57]	Transfer Learning	Binary Networks, 3DCNN	citySCENE	94.6%
Ullah et al. [64]	Ensemble/Transfer Learning/	CNN, Residual LSTM	UCF, UMN,Avenue	98.3%
Vu et al. [73]	Ensemble Learning	R-CNN, SVM, CGAN	Avenue, UCSD Ped1, Ped2, Shangai Tech	91.7%
Cinelli et al. [65]	Residual Network	ConvNet	CDNET2014	84.9%
Bhakat and Ramakrishnan [71]	Auto-encoder	ConvLSTM	Avenue, Surveillance Office, Police	73.6%
Ullah et al. [78]	Transfer Learning	Pre-trained CNN, BD-LSTM	UCF Crime	89.05%
Zahid et al. [58]	Transfer & Ensemble Learning	Fully Connected Network, Inception V3,	UCF Crime	-
Murugesan and Thilagamani [79]	Ensemble Learning	MLP-RNN	-	-
Nazare et al. [62]	Transfer Learning	Pre-trained CNNs	UCSD Ped2	76%

Aberkane and Elarbi [47]	Reinforcement Learning	Deep Q Learning Network (DQN),	UCF Crime	-
Bansod and Nandedkar [63]	Transfer Learning	Pre-trained CNN (VGG16)	UCSD, UMN	
Cinelli [48]	Transfer Learning	Pre-trained CNN ResNet	CDNET2014	85%
Pawar and Attar 2021 [80]	Auto-Encoder	ConvAE, LSTM AE	-	-
Zhao et al. [69]	Spatial Temporal Auto-encoder (STAE)	ConvLSTM	UCSD Ped1 & Ped2, CUHK Avenue	86.8%
Ramchandran and Sangaiah [67]	Auto-Encoder	ConvLSTM	UCSD Ped1 & Ped2	-
Duman and Erdem [56]	Auto-Encoder	OF-ConvAE-ConvLSTM	Avenue, UCSD Ped1, Ped2	91.53%
Doshi and Yilmaz [60]	Transfer Learning	Pre-trained Convnet (YOLOV3) & Least Square Generative Adversarial Network LS-GAN	CUHK, UCSD Ped2 & Avenue	84.83%
Nasaruddin [81]	Transfer Learning	3D-CNN	UCF Crime	95.4%
[82] Khaleghi and Moin	-	CNN	UCSD	-
Doshi and Yilmaz [59]	Transfer Learning	Pre-trained Convnet (YOLOV3) & GAN	UCSD PED2, CUHK, Shanghai Tech	84.87%
Nguyen [66]	Auto-encoder Hybrid	Conv-Net, GAN	UCSD Ped2, CHUK Avenue, Subway Entrance, Exit	91%
Liu et al. [52]	GAN	GAN	CHUK, UCSD Ped1 & Ped2, Shanghai Tech	83.76%

From the summary table, it can be noted that the best performing model in terms of accuracy is Ullah et al.is a typical case of transfer and ensemble learning that combines a CNN feature extractor with a residual LSTM network [78]. This model yields an overall accuracy of 98% per cent while detecting anomalies. This can be ranked among the best models. Unfortunately, more than 80% of the reviewed papers have not published their implementation code for further investigation or even improvement.

4.2.2 Experimental Investigation of the Reviewed Models

The researcher found out that, many of the published papers in this area, failed to share a complete implementation code. This factor is among the greatest challenges faced by other researchers who want to build enhancements on the existing models. Less than 20% of the papers had published their implementation code and some publishing only some part of the implementation code.

Three models had shared their complete code on GitHub, the Sultani et al Multiple Instance model [22], Chong and Tay Autoencoder [50] and Prediction based Anomaly Detector that uses Generative Adversarial Network(GAN) [51].

These models were implemented and further investigated to unruffle their internal working and learn ways of combing algorithms as well as the training, testing and evaluation techniques. Each model was investigated, and its model architecture was studied.

4.2.2.1 Transfer Learning: Sultani Multiple Instance Model

Sultani Multiple Instance Learning model [22], can be considered a breakthrough in this area due to its contributions. This model was among the pioneer works on anomaly detection that used transfer learning by utilizing a C3D feature extractor to extract features from the videos and multiple ranking algorithms that used Convnet and SVM. This paper

introduced the UCF Crime dataset, which has been widely accepted and used for research purposes within this domain.

During the development of the Sultani Multiple Instance Learning model [22], researchers introduced the UCF Crime dataset. The UCF Crime dataset is composed of real-world anomalies. The UCF Crime dataset is 1900 hours long video dataset that was introduced by Sultani et al. [22] and is composed of real-life anomalies like Arrest, Arson, Abuse and many others [22]. The training set has both abnormal and normal videos as well as the testing set. Although usage of both classes is dependent upon the nature of the model to be trained. This dataset has been re-used widely by other academicians and is immensely popular. For example, [64], [58], [47] and [81] have used it.

The Sultani et al. model [22] did not use the videos in their raw form, instead, the researchers used a C3D feature extractor to extract motion and appearance features. Figure 4.1 is an illustration of the C3D feature extractor. C3D is made up of deep 3-dimensional convolutional networks [17]. Its architecture contains 3x3x3 convolutional kernels, followed by 2x2x2 convolutional kernels. In total, the model contains 8 convolutional, 5 pooling layers and 2 fully connected layers. C3D feature extractor is important since it can extract motion and temporal features.

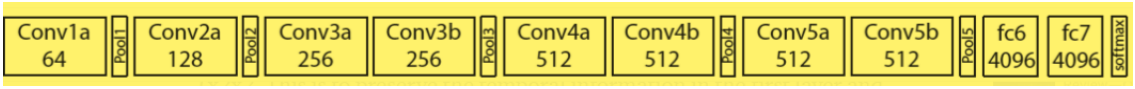


Figure 4.1: C3D Feature Extractor internal architecture adapted from [17]

C3D feature extractor was borrowed from Caffe and Facebook research [17]. It was released with an open-source license. During the model investigation, the command illustrated in figure 4.2 was used to extract features from the videos and it outputs a vector of shape 4096.

```
# extract feature
%cd /content/gdrive/My\ Drive/C3D2/C3D/C3D-v1.0/examples/c3d_feature_extraction
!sudo bash c3d_sport1m_feature_extraction_video.sh

/content/gdrive/My Drive/C3D2/C3D/C3D-v1.0/examples/c3d_feature_extraction
WARNING: Logging before InitGoogleLogging() is written to STDERR
```

Figure 4.2: Illustration of C3D feature extraction command

C3D feature extractor is compiled and run through a command illustrated in Figure 4.2. A video is passed to the command and the model extracts the features and it returns a vector of size 4096 as the output.

The extracted features were vector files with 4096 dimensions. The outputs are text files for each video. These features are fed into the fully connected neural network that is used to get a ranking value for the normal and abnormal videos. Figure 4.3 shows the structure of the model that Sultani et al used [22].

```
[ ] print("Create Model")
    model = Sequential()
    model.add(Dense(512, input_dim=4096,kernel_initializer='glorot_normal',kernel_regularizer=l2(0.001),activation='relu'))
    model.add(Dropout(0.6))
    model.add(Dense(32,kernel_initializer='glorot_normal',kernel_regularizer=l2(0.001)))
    model.add(Dropout(0.6))
    model.add(Dense(1,kernel_initializer='glorot_normal',kernel_regularizer=l2(0.001),activation='sigmoid'))
```

Figure 4.3: Fully Connected Sultani Ranking model

Figure 4.3 shows the structure of the Sultani ranking model that accepts the extracted features in vectors of size 4096 and then outputs anomaly status in the output layer. The

sigmoid activation function is used to extrapolate the anomaly ranking score. Figure 4.3 shows the structure of the fully convolutional model used by Sultani to detect anomalies

```
All_class_files= listdir(AllClassPath)
All_class_files.sort()
loss_graph =[]
num_iters = 20000
total_iterations = 0
batchsize=60
time_before = datetime.now()

for it_num in range(num_iters):

    AbnormalPath = os.path.join(AllClassPath, All_class_files[0])  # Path of abnormal already computed C3D features
    NormalPath = os.path.join(AllClassPath, All_class_files[1])    # Path of Normal already computed C3D features
    inputs, targets=load_dataset_Train_batch(AbnormalPath, NormalPath)  # Load normal and abnormal video C3D features
    batch_loss =model.train_on_batch(inputs, targets)
    loss_graph = np.hstack((loss_graph, batch_loss))
    total_iterations += 1
    if total_iterations % 20 == 1:
        print ('These iteration='+ str(total_iterations) + ') took:' + str(datetime.now() - time_before) + ', with loss of' + str(batch_loss))
        iteration_path = output_dir + 'Iterations_graph_' + str(total_iterations) + '.mat'
        savemat(iteration_path, dict(loss_graph=loss_graph))
    if total_iterations % 1000 == 0:  # Save the model at every 1000th iterations.
       weights_path = output_dir + 'weightsAnomalyL1L2_' + str(total_iterations) + '.mat'
       save_model(model, model_path, weights_path)


save_model(model, model_path, weights_path)
```

Figure 4.4: Illustration of Sultani Model training on batch process [22]

by using the output value of the model to rank anomalies in videos [22] The model was trained using both normal and abnormal features. The model was trained in batches. Figure 4.4 illustrates the training of the Sultani et al model and the saving of the training weights.

Testing was done on the published weights and the saved model. To get exactly the accuracy published in the Sultani et al of 75.41% [22]. Figure 4.5 displays an illustration of the output of the testing process of the model. It shows the output of the model by

printing the summary of the model and the time taken to run a successful training cycle. The accuracy scores were recorded and tabulated.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               2097664
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                16416
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 2114113
_________________________________________________________________
None
Total Time took: 0:00:01.278439
Total Time took: 0:00:02.268541
Total Time took: 0:00:03.076233
Total Time took: 0:00:04.185732
Total Time took: 0:00:05.319500
Total Time took: 0:00:06.264620
Total Time took: 0:00:07.428194
Total Time took: 0:00:08.494271
```

Figure 4.5: Output of the Sultani Model Testing [22]

4.2.2.2 Future Frame Prediction based Anomaly Detector

Generative Adversarial Networks (GAN) have shown immense potential in generating videos and images. The GANs contain the discriminative and the generator network part, which is trained to generate video frames [51]. Future video frames are predicted based on the video frames and then compared with the ground truth frames to identify anomalies. This model used the CUHK Avenue dataset, UCSD Pedestrian Dataset and ShangaiTech Campus Dataset. This model is also a case of transfer learning since it uses Flownet to estimate the optical flow in the videos. Complete code of this model was shared on Github. The code was obtained, and the experiment was redone, and the accuracy published matches the experiment that was conducted.

Figure 4.6 shows the structure of the Generative Adversarial Network used by Liu [51], the image portrays the discriminator and the generative part of the model. The flow net part is used to calculate motion features within the frames, and it is used to point out abnormalities in the motion feature.

```
def generator(inputs, layers, features_root=64, filter_size=3, pool_size=2, output_channel=3):
    return unet.unet(inputs, layers, features_root, filter_size, pool_size, output_channel)


def discriminator(inputs, num_filers=(128, 256, 512, 512)):
    logits, end_points = pix2pix.pix2pix_discriminator(inputs, num_filers)
    return logits, end_points['predictions']


def flownet(input_a, input_b, height, width, reuse=None):
    net = FlowNetSD(mode=Mode.TEST)
    # train preds flow
    input_a = (input_a + 1.0) / 2.0     # flownet receives image with color space in [0, 1]
    input_b = (input_b + 1.0) / 2.0     # flownet receives image with color space in [0, 1]
    # input size is 384 x 512
    input_a = tf.image.resize_images(input_a, [height, width])
    input_b = tf.image.resize_images(input_b, [height, width])
    flows = net.model(
        inputs={'input_a': input_a, 'input_b': input_b},
        training_schedule=LONG_SCHEDULE,
        trainable=False, reuse=reuse
    )
    return flows['flow']
```

Figure 4.6: Illustration of the GAN [51] generator, discriminator

The model was tested using the weights from the paper and the accuracy tallies with the publication. Figures 4.7, 4.8, and 4.9 below, show images of some tests that were run and their accuracy output.

Figure 4.7 shows the individual test cases being run during the model testing process using both the UCSD Ped1 and Ped2 datasets. The figures the looping of the test cases and their summation.

```
] !python inference.py  --dataset  ped2   \
                        --test_folder  '/content/drive/My Drive/datasets/future_frame_pred/Data/ped2/testing/frames'     \
                        --gpu  1    \
                        --snapshot_dir '/content/drive/My Drive/datasets/future_frame_pred/Codes/checkpoints/pretrains/ped2'
```

```
video = 12 / 12, i = 72 / 180, psnr = 37.641094
video = 12 / 12, i = 73 / 180, psnr = 38.012215
video = 12 / 12, i = 74 / 180, psnr = 38.403305
video = 12 / 12, i = 75 / 180, psnr = 37.845070
video = 12 / 12, i = 76 / 180, psnr = 37.369957
video = 12 / 12, i = 77 / 180, psnr = 37.161648
video = 12 / 12, i = 78 / 180, psnr = 37.294979
video = 12 / 12, i = 79 / 180, psnr = 37.595188
video = 12 / 12, i = 80 / 180, psnr = 37.553898
video = 12 / 12, i = 81 / 180, psnr = 37.925411
video = 12 / 12, i = 82 / 180, psnr = 37.780823
video = 12 / 12, i = 83 / 180, psnr = 37.396690
video = 12 / 12, i = 84 / 180, psnr = 36.761158
video = 12 / 12, i = 85 / 180, psnr = 37.633839
video = 12 / 12, i = 86 / 180, psnr = 37.543343
video = 12 / 12, i = 87 / 180, psnr = 37.189205
video = 12 / 12, i = 88 / 180, psnr = 37.095192
video = 12 / 12, i = 89 / 180, psnr = 37.194172
video = 12 / 12, i = 90 / 180, psnr = 36.950764
video = 12 / 12, i = 91 / 180, psnr = 36.888741
video = 12 / 12, i = 92 / 180, psnr = 37.117737
video = 12 / 12, i = 93 / 180, psnr = 37.215866
video = 12 / 12, i = 94 / 180, psnr = 36.913876
video = 12 / 12, i = 95 / 180, psnr = 36.932812
video = 12 / 12, i = 96 / 180, psnr = 36.632198
video = 12 / 12, i = 97 / 180, psnr = 37.126686
video = 12 / 12, i = 98 / 180, psnr = 37.172829
video = 12 / 12, i = 99 / 180, psnr = 37.128479
video = 12 / 12, i = 100 / 180, psnr = 36.804874
```

Figure 4.7: Testing of the GAN prediction model

Figure 4.8 below, shows the output after all the test cases were run. This figure portrays the output of the ped2 dataset. It goes further to illustrate how the area under the curve (AUC) score was computed. Finally, a score of 0.9539 was obtained.

```
total time = 4910.395835399628, fps = 0.40933563553261243
##### optimal result and model = dataset = ped2, loss file = psnrs/ped2_l_2_alpha_1_lp_1.0_adv_0.05_gdl_1.0_flow_2.0/ped2, auc = 0.9539455634972204
dataset = ped2, loss file = psnrs/ped2_l_2_alpha_1_lp_1.0_adv_0.05_gdl_1.0_flow_2.0/ped2, auc = 0.9539455634972204
```

Figure 4.8: Output of the Liu- GAN [51] model testing using the Ped2 dataset

Figure 4.9 below illustrates the output obtained after all the ped1 dataset videos were run through the Liu-GAN model [51]. The output describes the time take in seconds and the frame per second property of the video. The output gives accuracy score of 0.8315.

```
total time = 16551.81282734871, fps = 0.4349976691437312
##### optimal result and model = dataset = ped1, loss file = psnrs/ped1_l_2_alpha_1_lp_1.0_adv_0.05_gdl_1.0_flow_0.01/ped1, auc = 0.8315324430557308
dataset = ped1, loss file = psnrs/ped1_l_2_alpha_1_lp_1.0_adv_0.05_gdl_1.0_flow_0.01/ped1, auc = 0.8315324430557308
```

Figure 4.9: Output of the Liu-GAN Model [51] testing using Ped1 Dataset

4.2.2.3 Spatial-temporal Autoencoder

Chong and Tay published their complete implementation code, this made it possible to investigate the model since their complete code is accessible as well as the datasets [50]. This model used frames to train and test the model. So, the first step was to extract frames in the videos and create a frames dataset. This model used the reconstruction error to identify anomalies since the normal frames have low reconstruction error compared to the abnormal frames.

Figure 4.10 is an illustration of the code section that calculates the reconstruction error from the original frames and the reconstructed frames. The code calculates the Euclidean distance between the original sequences and the reconstructed sequences per every frame *i*.

```
# 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(0,sz)])
mse2=np.mean([np.power(np.linalg.norm(np.subtract(sequences[i],reconstructed_sequences[i])),2)for i in range(0,sz)])
mse = np.array([np.power(np.linalg.norm(np.subtract(sequences[i],reconstructed_sequences[i])),2)for i in range(0,sz)])
sa = (sequences_reconstruction_cost - np.min(sequences_reconstruction_cost)) / np.max(sequences_reconstruction_cost)
sr = 1.0 - sa
```

Figure 4.10: Illustration of the regularity score computation function

The highlighted section within the Figure 4.10 illustrates the calculation of the regularity score by scaling of the scores between 0 and 1. This scaled the reconstruction error to

values between 0 and 1 and finally the value is subtracted from 1 to get the regularity score that indicates presence of anomaly.

The model is constituted of spatial and temporal parts, with the spatial containing convolutional layers wrapped in time-distributed layers while the temporal part is made up of Convolutional LSTM layers that can preserve the learned weights across the temporal sequence. Below is the illustration of the spatial-temporal autoencoder by [50].

Figure 4.11: Chong and Tay Autoencoder Illustration

Figure 4.10 illustrates the computation of the regularity score from the Euclidean distance while Figure 4.11 demonstrates the spatial and the temporal components of the model. It can be noted the use of convolutional and convolutional LSTM as the underlying deep learning algorithms.

The model accepts a sequence of 10 images of size 256 by 256, which it encodes to the latent space and then decoded back to an output of 10 images of size 256 by 256.

The input and the output size of the autoencoders are usually the same. Therefore, frame width, height, and the number of frames in a sequence are the same for the input and the output.

Figure 4.12 shows a plot of the regularity score per every frame in the short video. After the regularity score was calculated in figure 4.10, the regularity score sr(t) is plotted for every frame in the video to show the anomalies within the video. Anomalies are identified when the regularity score is low.

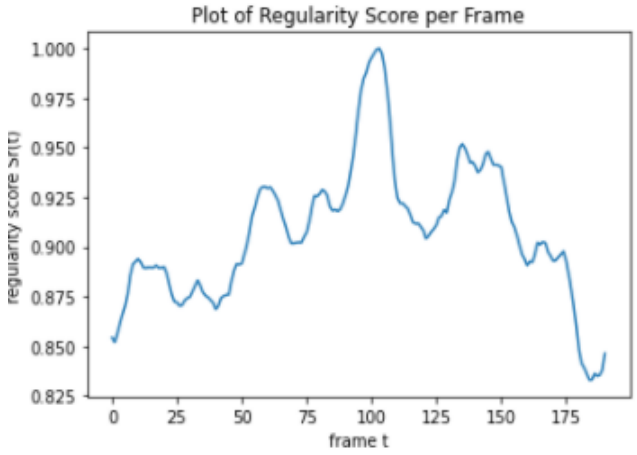


Figure 4.12: Chong and Tay Autoencoder Regularity Score [50]

The model returns a regularity score that is used to identify anomalies at the frame level. The highest picks on the graph indicate, high regularity score and hence that region has normal frames. Low points on the graph indicate the presence of anomalies since anomalies are irregular patterns.

4.2.3 Results of Experimental Investigation

The first objective was to find the popular deep learning models and then select one model to be improved. More than 30 models were reviewed and less than 20% of the models were selected for experimental investigation since they had shared complete implementation code. The biased selection was based on the accessibility of the complete code. Three models were hand-picked due to their availability.

The models and the datasets used in them were studied further and run to identify parameters used in their training and testing. It is important to note that the 3 models selected as seen above are completely different in terms of their internal organization, constitution, and their design. Two of the models can be considered as transfer learning since they build upon knowledge from other previous models i.e., C3D and Flow net. This is advantageous since it improves the performance of the new models constructed but also troublesome since the pre-trained models can transfer errors to the hybrid model.

Table 4.2 is a tabulation of the model accuracies, according to the different datasets used in the empirical investigations.

Table 4.2: Comparison of Frame Level AUC

	Frame Level AUC comparison					
No.	Method	Accuracy per Dataset				
		UCSD Ped1	UCSD Ped2	Avenue	UCF Crime	Average
1.	Multiple Instance Learning, Sultani et al. [22]	n/a	n/a	n/a	75.41%	75.41%
2.	STAE Autoencoder, Chong and Tay [50]	86.14%	90.23%	81.23%	78.23%	83.96%
3.	Generative Adversarial Network Future Frame Prediction, Liu et al. [51]	83.15%	95.31%	84.89%	n/a	87.78%

Table 4.2 above shows the comparison of the ROC curve AUC of three models with distinct learning techniques. All the models have shown high accuracy in anomaly prediction. The first model when compared with the second model does better in terms of accuracy. The GAN Future Frame prediction model [51] outperforms the Spatial-Temporal Autoencoder [50] in some instances.

The Multiple Instance Learning-Sultani Model [22] and the Future Frame Prediction Liu [51] models are complex due to their transfer learning nature where they borrow from other pre-trained models. They are susceptible to transfer of error and hence they are not considered for improvement due to their internal working complexity. On the other hand, Chong, and Tay's [50] autoencoder is simple in its design since it is not a hybrid solution. The Chong and Tay autoencoder can be improved with ease, hence it was selected for enhancement [50]. The Chong and Tay model pays attention to the spatial and temporal

nature of the videos since it has spatial and temporal descriptors dedicated to extracting those features. Hence it was selected.

4.2.4 Selected Model

It was established from the previous objective that, the Chong and Tay Spatial-Temporal Autoencoder [50] was the chosen model to be improved. The improvement was aimed at the reduction of anomaly detection errors. Therefore, it is important to understand the architecture of the model before the improvement process. The design of autoencoders is composed of the encoder, latent space, and the decoder part.

The Chong and Tay Spatial-Temporal Autoencoder [50] model will be referred to as the selected model herein and the improved model will be referred to as the enhanced model. The selected model is composed of spatial and temporal parts in its encoding and decoding parts. To better understand the architecture of the model, a graphical illustration of the model and a model summary printed from the code is shown below.

The autoencoder learns the (normal) regular patterns from the training videos. The model has two parts namely spatial and temporal autoencoder. The spatial part extracts the location-based data. The spatial part encodes the location of objects within an image. The spatial encoder and decoder have two convolutional and deconvolutional layers correspondingly.

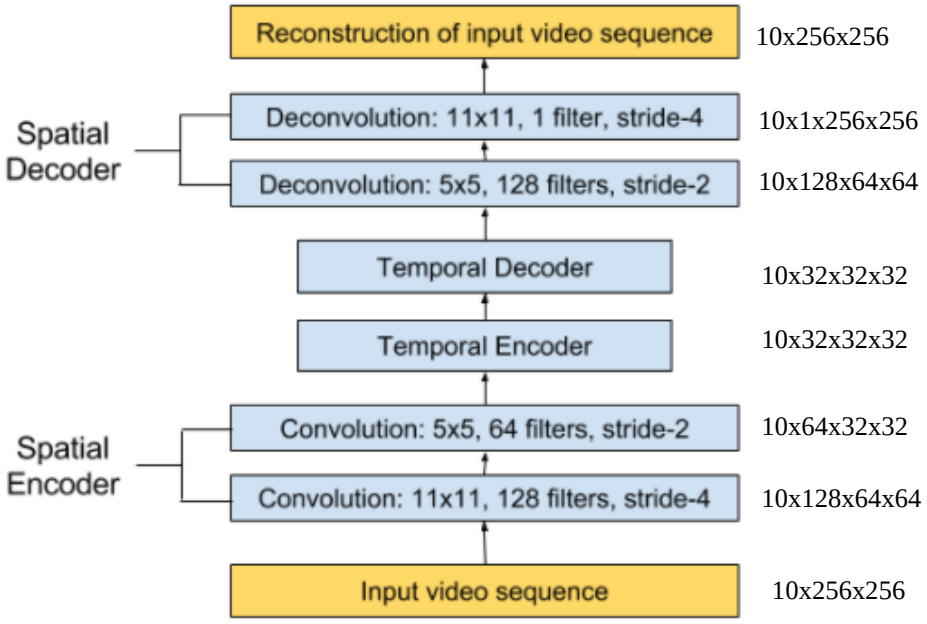


Figure 4.13: Illustration of the Spatial-Temporal Model Architecture [50]

Figure 4.13 shows the design parts of the Spatial-Temporal Model before improvement. The temporal part of the autoencoder is composed of 3 layers of convolutional long short-term memory ConvLSTM. The convolutional part is good at object recognition while the LSTM part performs well at sequence learning and time modelling.

LayerNormalization is used between the layers to normalize the weights. This layer performs similar operations during the training and testing of the model. The *LayerNormalization* layer is used to stabilize the hidden state of the recurrent networks like ConvLSTM. Changes caused by the output of one-layer causes highly correlated changes summed in the inputs to the next layer. These changes cause the covariate shift

problem which can be minimized by estimation of mean μ and the variance σ of the summation of inputs in each layer [40].

LayerNormalization statistics is applied to all hidden units in the same and it ensures that all hidden layers have the same normalization terms of μ and the σ. The mean of the layer is denoted by μ, and the standard deviation of the layer is denoted by σ. Which are calculated as follows [40]:

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$

H denotes the number of hidden units in each layer.

Therefore, the mean (μ) and the standard deviation (σ) of the layer are used to normalize the weights of the layer in a process called layer normalization.

The model is trained using normal videos. The convolutional networks learn the weights of the filters during the training process, and they learn how to reconstruct the normal videos. Parameters like the number of layers, filters and filter size are set. These parameters can be tuned to increase the accuracy of the model. Therefore, model improvement involved tweaking the model parameters. For instance, a larger number of filters, extract more features from images and yield a better network that recognizes patterns in unseen images.

The reconstruction error difference when reconstructing abnormal clips is used to calculate the regularity score that is used to estimate the abnormality score of every frame. It is paramount to understand the calculation of the reconstruction error of all pixel values

I in frame t, which is taken as the Euclidean distance between the input frame and the reconstructed frame.

$$e(t)=||x(t)-f_W(x(t))||_2$$

Figure 4.14: Euclidean distance applied to calculate reconstruction error

Figure 4.14 portrays the reconstruction error calculated as the Euclidean distance. Where f_w is the learned weights of the model, x(t) is the input frame and the $f_w(x(t))$ defines the reconstructed video frame. Their difference is referred to as the Euclidean distance denoted by e(t).

Abnormality score $S_a(t)$ was calculated by scaling the Euclidean distance from 0 and 1. Scaling of the Euclidean distance to obtain the abnormality score involved taking the e(t) Euclidean distance per pixel and subtracting the ratio of the least $e(t)_{min}$ divided by the largest Euclidean distance denoted as $e(t)_{max}$ in figure 17.

Afterwards, the regularity score $S_r(t)$ was derived by subtracting the abnormal score $S_a(t)$ from 1.

$$s_a(t)=e(t)-e(t)_{min}e(t)_{max}$$

$$s_r(t)=1-s_a(t)$$

Figure 4.15: Regularity score and Abnormality score calculation

The selected model code is attached to Appendix IV, the code can be noted that it has 15 layers and a total of 1,958,209 trainable parameters. This was the structure of the model before enhancement.

4.3 Data Preprocessing

The Chong and Tay [50] autoencoder model were selected for improvement. This model uses frames of size 256x256 for training. Therefore, the first step of data preparation is the extraction of frames from the videos and splitting of normal videos as the training set and abnormal videos as the testing set. Opencv library is utilized to extract the frames from the videos.

```
def video_to_frames(video_path, frames_dir, overwrite=False, every=1, chunk_size=1000):
    """
    Extracts the frames from a video using multiprocessing
    :param video_path: path to the video
    :param frames_dir: directory to save the frames
    :param overwrite: overwrite frames if they exist?
    :param every: extract every this many frames
    :param chunk_size: how many frames to split into chunks (one chunk per cpu core process)
    :return: path to the directory where the frames were saved, or None if fails
    """

    video_path = os.path.normpath(video_path)  # make the paths OS (Windows) compatible
    frames_dir = os.path.normpath(frames_dir)  # make the paths OS (Windows) compatible

    video_dir, video_filename = os.path.split(video_path)  # get the video path and filename from the path

    # make directory to save frames, its a sub dir in the frames_dir with the video name
    os.makedirs(os.path.join(frames_dir, video_filename), exist_ok=True)

    print(video_filename)

    capture = cv2.VideoCapture(video_path)  # load the video
    total = int(capture.get(cv2.CAP_PROP_FRAME_COUNT))  # get its total frame count
    capture.release()  # release the capture straight away

    if total < 1:  # if video has no frames, might be and opencv error
        print("Video has no frames. Check your OpenCV + ffmpeg installation")
        return None  # return None

    frame_chunks = [[i, i+chunk_size] for i in range(0, total, chunk_size)]  # split the frames into chunk lists
    frame_chunks[-1][-1] = min(frame_chunks[-1][-1], total-1)  # make sure last chunk has correct end frame, also han

    prefix_str = "Extracting frames from {}".format(video_filename)  # a prefix string to be printed in progress bar

    # execute across multiple cpu cores to speed up processing, get the count automatically
    with ProcessPoolExecutor(max_workers=multiprocessing.cpu_count()) as executor:
        futures = [executor.submit(extract_frames, video_path, frames_dir, overwrite, f[0], f[1], every)
                   for f in frame_chunks]  # submit the processes: extract_frames(...)
```

Figure 4.16: Frame extraction code for video data preparation

Figure 4.16 shows the process of frame extraction and saving of frames within a structured directory to preserve the frames of a video in one directory. The full code for the whole extraction process is attached to Appendix II.

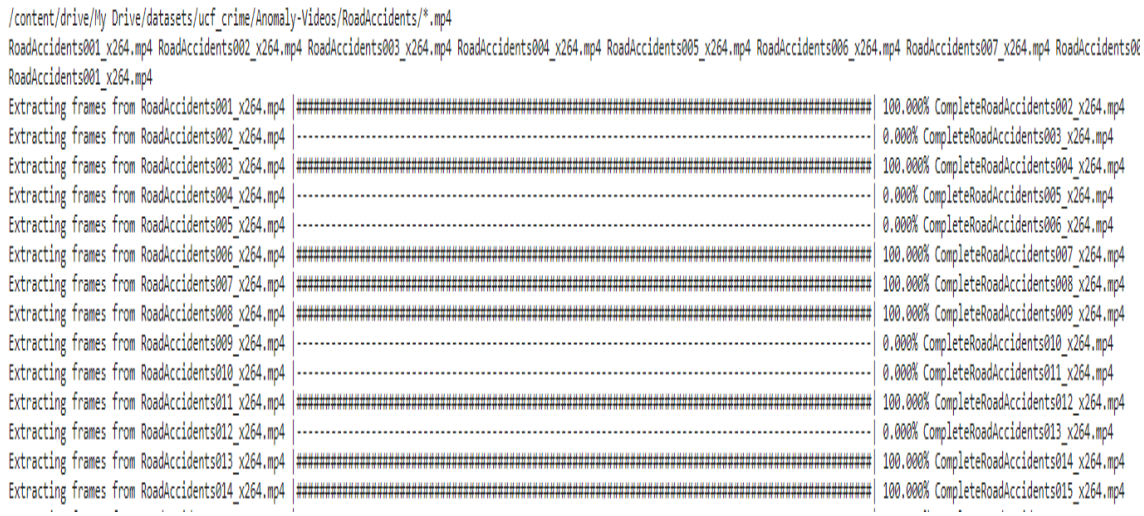


Figure 4.17: Frames Extraction process

Figure 4.17 portrays the frame extraction process from the videos. It shows the output of several videos in the UCF Crime dataset. Other ways to improve the data for effective model training and accuracy improvement included data augmentation. This process increases the amount of data by applying transformations like reflection, rotation and zooming in to expose the model to all angles of the image. It makes the dataset richer and more realistic.

Data augmentation is the process of increasing the size and variety of data [38]. Existing data is transformed through some modifications. Data augmentation is useful when there is a shortage of data and variety. This process has been found to reduce overfitting. In the study, the geometrical transformation was used to perform data augmentation. The

geometrical transformation includes random flip, cropping, rotation, and transformation. The Keras ImageDatagenerator class was used for the augmentation tasks.

The sliding window technique was applied to increase the size of the training dataset. Frames were concatenated through strides to acquire a rich dataset. For instance, a stride could concatenate the even frames while the other one concatenated the odd frames. The sliding window technique has been illustrated in Appendix II.

4.4 Model Enhancement

The model improvement process was conducted as a 3-tier process by applying 3 different treatments to increase model accuracy, reduce overfitting, and reduction of human supervision. The improvements that were done include, the introduction of Max-pooling, the addition of model depth and the addition of a classifier layer.

4.4.1 Max-Pooling Treatment

The pooling operation is the process of sliding a filter(kernel) across each channel of the feature map to summarize the features lying within the area covered by the filter. Pooling can be considered as part of the convolutional layer building block. The max-pooling reduces the size of the spatial representation since it reduces the number of parameters and computations in the network.

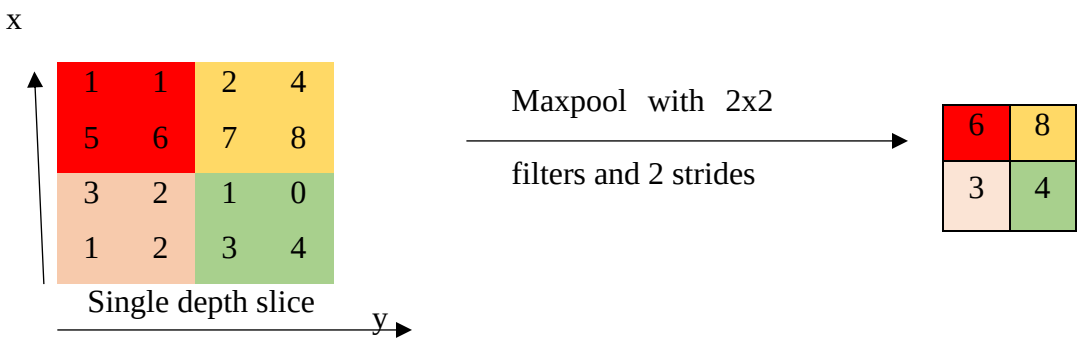


Figure 4.18: High-level illustration of Max-pooling operation

Figure 4.18 illustrates how a max-pooling operation works by taking the maximum value in its filter. The max-pooling operation generates a single value from each filter. The value can be denoted as $Z_f = \max\{S\} = \max\{s_1, s_{2,} s_{3.....}s_n\}$. The pooling operation aggregates together the output of each, and it compacts the output of a layer to a vector. Max-pooling has been found to increase the overall performance of the model by up to 2% [83].

The MaxPooling2D was used due to the nature of the Convolutional Network used in the Spatial encoding and decoding, which was Conv2D. Since the spatial parts were wrapped in TimeDistributed functions to preserve the movements of objects, pooling functions were also wrapped within the TimeDistributed functions. The encoding part of the autoencoder was fitted with the MaxPooling2D while the decoder part was fitted with the UpSampling2D which reverses the max-pooling effect for the spatial reconstruction.

The Spatial Encoder before max-pooling treatment had only the convolutional layer and layer normalization.

```
seq = Sequential()
seq.add(TimeDistributed(Conv2D(128, (11, 11), strides=4, padding="same"), batch_input_shape=(None, 10, 256, 256, 1)))
seq.add(LayerNormalization())
seq.add(TimeDistributed(Conv2D(64, (5, 5), strides=2, padding="same")))
seq.add(LayerNormalization())
# # # # #
```

Figure 4.19: Spatial encoder, before the introduction of max pooling

Figure 4.19 shows the encoding part of the model. It shows that the partial encoding which is consisted of the convolution 2d layers. This part provides the extraction of the

appearance and texture features. Features extraction was guided by the spatial encoders that extracted spatial features from the video frames. The encoding part draws the 256 by 256 features from every image, then it scales it down to 32 by 32 features. In some way this was dimension reduction.

The encoder part after the addition of a max-pooling layer

```
seq = Sequential()
seq.add(TimeDistributed(Conv2D(128, (11, 11), strides=4, padding="same"), batch_input_shape=(None, 10, 256, 256,1)))
seq.add(LayerNormalization())
seq.add(TimeDistributed(MaxPooling2D(pool_size=(4,4),padding="same")))
seq.add(TimeDistributed(Conv2D(64, (5, 5), strides=2, padding="same")))
seq.add(LayerNormalization())
seq.add(TimeDistributed(MaxPooling2D(pool_size=(2,2),padding="same")))
# # # # #
```

Figure 4.20: New encoder after addition of max-pooling2d

Figure 4.20 shows the outcome of the encoding part after the max-pooling function was introduced. Note the addition of the Maxpooling2D function wrapped within TimeDistributed function. The pool size varies according to the scaling of the convolution sizes.

UpSampling2D is the deconvolutional layer pooling function that reverses the Max-pooling operation in the spatial decoder. The work of the decoder part of the autoencoder is to reverse the operations of the encoder part. Therefore, Up Sampling increases the output dimensions, unlike the max-pooling that reduces it. They extend the range of the next kernel by adding the size of the vector.

Figure 4.21 shows the Spatial Decoder before the introduction of the un-pooling operation

```
# # # # #
seq.add(TimeDistributed(Conv2DTranspose(64, (5, 5), strides=2, padding="same")))
seq.add(LayerNormalization())
seq.add(TimeDistributed(Conv2DTranspose(128, (11, 11), strides=4, padding="same")))
seq.add(LayerNormalization())
seq.add(TimeDistributed(Conv2D(1, (11, 11), activation="sigmoid", padding="same")))
```

Figure 4.21: Spatial Decoder before unpooling was introduced

UpSampling2D functions were added by researcher as part of the improvement, these operations were wrapped in the TimeDistributed function to preserve the motion features. See figure 4.22 below.

```
# # # # #
seq.add(TimeDistributed(Conv2DTranspose(64, (5, 5), strides=2, padding="same")))
seq.add(LayerNormalization())
seq.add(TimeDistributed(UpSampling2D(size=(2,2))))
seq.add(TimeDistributed(Conv2DTranspose(128, (11, 11), strides=4, padding="same")))
seq.add(LayerNormalization())
seq.add(TimeDistributed(UpSampling2D(size=(4,4))))
seq.add(TimeDistributed(Conv2D(1, (11, 11), activation="sigmoid", padding="same")))
print(seq.summary())
```

Figure 4.22: Spatial Decoder after the introduction of UpSampling2D

Figure 4.22 depicts the new spatial decoder with Upsampling2D functions that reverse the effect of Maxpooling functions introduced in the encoding part. The size of the Upsampling matrix is dependent on the size of deconvolution layer. The introduction of the pooling operation to the model reduced overfitting and reduced the computation complexity which was expected to cut some of the anomaly prediction errors. It was observed that training time was cut slightly, and the model accuracy had some slight increment in accuracy.

4.4.2 Model Depth Tuning

The idea behind deep learning is that a deeper network performs better. Therefore, state of art models has shown a design trend of stacking layers of the network to enhance model performance. Increasing the depth of the model was considered as part of the model enhancement process. New layers were introduced in the model to increase learnable features. Additional new layers were built upon the previous improvement that added max-pooling operations.

Figure 4.23 shows the addition of the new layers on the model encoding part. It portrays the 98th filter layer added between the 128 and 64 filter layers.

```
seq = Sequential()
seq.add(TimeDistributed(Conv2D(128, (11, 11), strides=2, padding="same",activation='relu'), batch_input_shape=(None, 10, 256, 256,1)))
seq.add(LayerNormalization())
seq.add(TimeDistributed(MaxPooling2D(pool_size=(4,4),padding="same")))
seq.add(TimeDistributed(Conv2D(98, (7, 7), strides=2, padding="same",activation='relu')))
seq.add(LayerNormalization())
seq.add(TimeDistributed(MaxPooling2D(pool_size=(2,2),padding="same")))
seq.add(TimeDistributed(Conv2D(64, (5, 5), strides=2, padding="same",activation='relu')))
seq.add(LayerNormalization())
seq.add(TimeDistributed(MaxPooling2D(pool_size=(2,2),padding="same")))
# # # # #
```

Figure 4.23: Addition of the new layer with 98 filter size

This adds a new convolution layer that extracts the spatial features. It can be seen in figure 26 that the new layers increased the depth of the model. The features extracted include appearance, motion, and trajectory features.

Figure 4.24 shows the new features added to the temporal section of the model. The use of 3 by 3 kernel sizes can be noted since they work better than the kernels with large kernels.

Several layers were added to the temporal section of the autoencoder to have more depth. From three layers to five layers the 48-filter size was introduced to ensure more detailing.

```
# # # # #
seq.add(ConvLSTM2D(64, (3, 3), padding="same", activation="relu", return_sequences=True))
seq.add(LayerNormalization())
seq.add(ConvLSTM2D(48, (3, 3), padding="same", activation="relu", return_sequences=True))
seq.add(LayerNormalization())
seq.add(ConvLSTM2D(32, (3, 3), padding="same",  activation="relu", return_sequences=True))
seq.add(LayerNormalization())
seq.add(ConvLSTM2D(48, (3, 3), padding="same", activation="relu", return_sequences=True))
seq.add(LayerNormalization())
seq.add(ConvLSTM2D(64, (3, 3), padding="same",  activation="relu", return_sequences=True))
seq.add(LayerNormalization())
# # # # #
```

Figure 4.24: Addition of temporal encoder-decoder depth

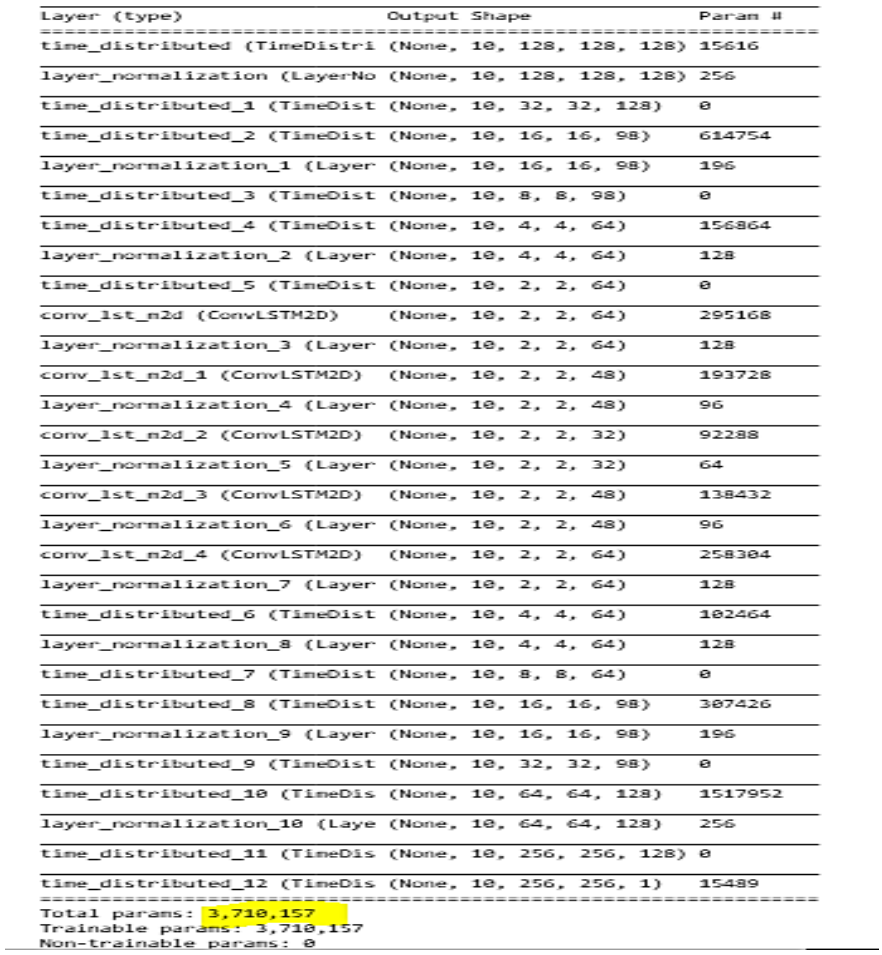
An additional layer was introduced in the spatial decoder part of the model as well to have more depth mirroring the exact structure of the spatial encoder. The decoder part is made of deconvolutional layers that are used for reconstruction. Therefore, a filter of size 98 was introduced to enhance feature reconstruction. Figure 4.25 shows the new decoder after the 98filter layer was introduced. The deconvolutional layer reverses the operation of the convolutional layer hence it is introduced in both parts.

```
# # # # #
seq.add(TimeDistributed(Conv2DTranspose(64, (5, 5), strides=2, padding="same",activation='relu')))
seq.add(LayerNormalization())
seq.add(TimeDistributed(UpSampling2D(size=(2,2))))
seq.add(TimeDistributed(Conv2DTranspose(98, (7, 7), strides=2, padding="same",activation='relu')))
seq.add(LayerNormalization())
seq.add(TimeDistributed(UpSampling2D(size=(2,2))))
seq.add(TimeDistributed(Conv2DTranspose(128, (11, 11), strides=2, padding="same",activation='relu')))
seq.add(LayerNormalization())
seq.add(TimeDistributed(UpSampling2D(size=(4,4))))
seq.add(TimeDistributed(Conv2D(1, (11, 11), activation="sigmoid", padding="same")))
print(seq.summary())
```

Figure 4.25: Addition of Spatial Decoder depth

The model depth increased the trainable parameters from 1,958,209 to 3,710,157. The increase of trainable parameters increased the ability of the model to learn more features from the data and identify objects.

The training data was composed of 10 sequences derived from the frames of the normal videos. The frames were of the same size of 256x256, and they were fit to the model for the training process. The summary of the whole enhanced model after it was executed during the training process is illustrated in figure 4.26.



Layer (type)	Output Shape	Param #
time_distributed (TimeDistri	(None, 10, 128, 128, 128)	15616
layer_normalization (LayerNo	(None, 10, 128, 128, 128)	256
time_distributed_1 (TimeDist	(None, 10, 32, 32, 128)	0
time_distributed_2 (TimeDist	(None, 10, 16, 16, 98)	614754
layer_normalization_1 (Layer	(None, 10, 16, 16, 98)	196
time_distributed_3 (TimeDist	(None, 10, 8, 8, 98)	0
time_distributed_4 (TimeDist	(None, 10, 4, 4, 64)	156864
layer_normalization_2 (Layer	(None, 10, 4, 4, 64)	128
time_distributed_5 (TimeDist	(None, 10, 2, 2, 64)	0
conv_lst_m2d (ConvLSTM2D)	(None, 10, 2, 2, 64)	295168
layer_normalization_3 (Layer	(None, 10, 2, 2, 64)	128
conv_lst_m2d_1 (ConvLSTM2D)	(None, 10, 2, 2, 48)	193728
layer_normalization_4 (Layer	(None, 10, 2, 2, 48)	96
conv_lst_m2d_2 (ConvLSTM2D)	(None, 10, 2, 2, 32)	92288
layer_normalization_5 (Layer	(None, 10, 2, 2, 32)	64
conv_lst_m2d_3 (ConvLSTM2D)	(None, 10, 2, 2, 48)	138432
layer_normalization_6 (Layer	(None, 10, 2, 2, 48)	96
conv_lst_m2d_4 (ConvLSTM2D)	(None, 10, 2, 2, 64)	258304
layer_normalization_7 (Layer	(None, 10, 2, 2, 64)	128
time_distributed_6 (TimeDist	(None, 10, 4, 4, 64)	102464
layer_normalization_8 (Layer	(None, 10, 4, 4, 64)	128
time_distributed_7 (TimeDist	(None, 10, 8, 8, 64)	0
time_distributed_8 (TimeDist	(None, 10, 16, 16, 98)	307426
layer_normalization_9 (Layer	(None, 10, 16, 16, 98)	196
time_distributed_9 (TimeDist	(None, 10, 32, 32, 98)	0
time_distributed_10 (TimeDis	(None, 10, 64, 64, 128)	1517952
layer_normalization_10 (Laye	(None, 10, 64, 64, 128)	256
time_distributed_11 (TimeDis	(None, 10, 256, 256, 128)	0
time_distributed_12 (TimeDis	(None, 10, 256, 256, 1)	15489

Total params: 3,710,157
Trainable params: 3,710,157
Non-trainable params: 0

Figure 4.26: Enhanced model summary

Figure 4.26 illustrates the structure of the improved model. The depth increment and introduction of regularization functions can be visible from the model summary.

4.4.3 Enhanced Autoencoder Model Training

The autoencoder models are trained by fitting the model with the same values of X and Y since the goal is to teach the model how to reconstruct the given input. Therefore, our training dataset of videos was used for training without any labels. A little processing was done to ensure that the frames had the same size of 256x256, and the values of the channels were scaled down to values between 0 and 1. The frames were concatenated in the sliding window technique to acquire sequences of frames cut into bunches of 10. The model takes bunches of 10 frames and then the other frames are fitted progressively in all training epochs. Figure 4.27 shows the data scaling by scaling the images to values between 0 and 1. Scaling of inputs avoids exploding gradient problems. Hence it was applied.

```
for f in sorted(listdir(Config.DATASET_PATH)):
    if isdir(join(Config.DATASET_PATH, f)):
        all_frames = []
        # loop over all the images in the folder (0.tif,1.tif,..,199.tif)
        for c in sorted(listdir(join(Config.DATASET_PATH, f))):
            if str(join(join(Config.DATASET_PATH, f), c))[-3:] == "jpg":
                img = Image.open(join(join(Config.DATASET_PATH, f), c)).resize((256, 256))
                img = img.convert('L')
                img = np.array(img, dtype=np.float32) / 256.0
                all_frames.append(img)
        # get the 10-frames sequences from the list of images after applying data augmentation
        for stride in range(1,2):
            clips.extend(get_clips_by_stride(stride=stride, frames_list=all_frames, sequence_size=10))
return clips
```

Figure 4.27: Training Dataset Preparation

Figure 4.27 takes in the .jpg image then it divides every pixel value with 256 to scale it down to a value range between 0 and 1. The function then returns an array of values that represent the image.

The training data was fit to the autoencoder, to set its internal weights for the sequence reconstruction task as follows. Figure 4.28 shows the training of the enhanced model.

```
Total params: 3,710,157
Trainable params: 3,710,157
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/3
170/170 [==============================] - 10072s 59s/step - loss: 0.0116 - accuracy: 1.6096e-05
Epoch 2/3
170/170 [==============================] - 10051s 59s/step - loss: 0.0049 - accuracy: 1.4040e-05
Epoch 3/3
170/170 [==============================] - 10046s 59s/step - loss: 0.0047 - accuracy: 1.3596e-05
```

Figure 4.28: Enhanced autoencoder training process

Figure 4.28 portrays the model training process. The process involved 3 epochs to conserve the available RAM memory while each epoch contained 180 steps. The training process took around 4 to 6 hours when deployed on a CPU environment depending on the size of the dataset.

The training of the model extracted features from the video frames. The features extracted included appearance, motion and trajectory features that define characters within the video. Extracted features were used by the model calculate the reconstruction errors dataset that is used to detect the anomalies within every frame.

4.4.4 Extraction of features and dimensionality reduction

Video frames were processed within the autoencoder by extraction of features that described shapes, edges, and motion. Key features were drawn from the frames to form a vector representation of the video, referred as the latent space. Extraction of the features was guided by the spatial and temporal encoding parts of the model. An input image of size 256 x 256 are fed to the model. The first conv2d layer extracted 65,536 features per every pixel in that frame.

The encoding part reduced the size of the video frames by scaling it down through convolutions of sizes 128, 98, 64 and 32. This process can be considered as reduction of dimensions. Consider a single frame that was reduced from 65,536 features to 1,024 features that translated from size 256x256 to size 32 by 32. The reduced features were combined to a latent space that was used by the autoencoder to reconstruct videos.

The logic of anomaly detection drew from the reconstruction error. The autoencoder was trained on normal videos. The trained model was tested with a video containing anomalies. Larger reconstruction error was realized in every frame that contained an anomaly. Reconstruction error within the frames guided the detection of anomalies per every frame.

4.4.5 Introduction of One-class Support Vector Machine

The existing model did not have a way of calculating the regularity threshold that determines whether a frame has an anomaly or not. To automatically identify the threshold, I introduced an unsupervised clustering tool. One class SVM trained on the

reconstruction error of the normal videos, allows the One-class Support Vector Machine to establish a threshold by identifying the reconstruction error that lies outside the normal class, hence identifying anomalies. One Class Classification entails training the model on the normal data and forecasting whether new data is normal or abnormal.

One class classification technique is used for classification, where the normal is taken as the negative instance (class 0) while the abnormal/anomalies are taken as the positive instance (Class 1). This implies:

Negative Instance: Normal assigned 0

Positive Instance: Anomaly/Outlier assigned 1

One Class SVM captures the density of the majority class, Normal Video Frames are the majority since the anomalies are rare. Anomalies are classified as outliers or extremes of the density function.

Reconstruction errors were used to train the classifier, it was established that the One-class SVM had to be trained on the normal video data only. Therefore, the normal videos were run through the enhanced autoencoder, and the reconstructed sequences were used to acquire the reconstruction error from the Euclidean distance. The reconstruction error of normal videos was used to train the one-class SVM model. For testing purposes, videos with anomalies were run through the enhanced autoencoder and the resultant data was used as a testing set.

4.4.5.1 One class SVM Training dataset

The training dataset was derived from the reconstruction errors of the normal videos run through the autoencoders. The reconstruction error of each frame was accumulated in an array together with other videos to cover the normal videos within the dataset.

```
def svm_training_features():
    # loop over the training folders (Train000,Train001,..)
    model = get_model(False)
    print("got model")
    all_recon_cost=[]
    for f in sorted(listdir(Config.DATASET_PATH)):
        if isdir(join(Config.DATASET_PATH, f)):
            all_frames = []
            test=get_normal_vid(f)
            print(test.shape)
            sz = test.shape[0] - 10 + 1
            sequences = np.zeros((sz, 10, 256, 256, 1))
            # apply the sliding window technique to get the sequences
            for i in range(0, sz):
                clip = np.zeros((10, 256, 256, 1))
                for j in range(0, 10):
                    clip[j] = test[i + j, :, :, :]
                sequences[i] = clip

            print("got data")
            # 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(0,sz)])
            all_recon_cost.append(sequences_reconstruction_cost)
```

Figure 4.29: One class SVM Training Data creation

Figure 4.29 shows, the extraction of the regularity score datasets that were used to train the One-class SVM. Reconstruction cost/error was acquired by calculating the Euclidian distance between the original sequences and the reconstructed sequences of the normal videos. The function within Figure 4.29 returns the reconstruction cost data in *all_recon_cost* variable.

4.4.5.2 One class SVM Training

After the test data was acquired, the next step was to train the One-class SVM classifier with the normal instance's reconstruction cost.

```
def svm_detector_training():
    from sklearn import svm
    import pandas as pd
    import numpy as np
    train_feature=svm_training_features()
    train_feature=pd.DataFrame(train_feature)
    print('Oneclass SVM training')
    oneclass=svm.OneClassSVM(kernel='linear', gamma=0.001, nu=0.95)
    oneclass.fit(train_feature)
    return oneclass
```

Figure 4.30: One-Class SVM training process

Figure 4.30 depicts the training of the classifier. The classifier is trained using the fit () function that uses the reconstruction costs to train the OneClassSVM.

4.4.5.3 One-Class SVM Test Dataset

Abnormal videos were fed to the autoencoder through the *model.predict()* function to acquire the reconstructed sequences that were used to calculate the reconstruction error. This reconstruction error was used to test the One-Class SVM.

The test dataset was used to test the ability of the One-Class SVM to detect anomalies at every frame by returning the class value, where there was an anomaly, it was expected to return 1, otherwise it returns 0. Figure 4.31 shows the testing and validation of the improved model.

```
def svm_testing():

    model = get_model(False)
    print("Got Model")
    #test shape
    test = get_single_test()
    print(test.shape)
    sz = test.shape[0] - 10 + 1
    sequences = np.zeros((sz, 10, 256, 256, 1))
    # apply the sliding window technique to get the sequences
    for i in range(0, sz):
        clip = np.zeros((10, 256, 256, 1))
        for j in range(0, 10):
            clip[j] = test[i + j, :, :, :]
        sequences[i] = clip

    print("got data")
    # 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(0,sz)])
    sa = (sequences_reconstruction_cost - np.min(sequences_reconstruction_cost)) / np.max(sequences_reconstruction_cost)
    sr = 1.0 - sa

    test_feature=sequences_reconstruction_cost
    print ('Got Testing set')
    X_test=pd.DataFrame(test_feature)
    svm_model=svm_detector_training()
    #check outliers predicted on single test case
    fraud_pred = oneclass.predict(X_test)
    return fraud_pred
```

Figure 4.31: Test Dataset Generation

It points out how the regularity scores can be applied to pinpoint anomalies. Anomaly detection relied on the reconstruction error obtained from the calculation of the Euclidean distance between the pixels of the original frame and reconstructed frame. It had been noted by Chong and Tay [19], that the frames with anomalies have higher reconstruction error. The autoencoder model returned the reconstruction error that indicates anomalies. One class SVM expanded that logic by classifying the reconstruction score as either normal or abnormal.

4.5 Experiments

This chapter expounds on the experiments that I conducted, how they were set up and the datasets used in the different test cases. A posttest control experimental setup methodology was followed. The accuracy was measured after the enhancement was done to a random selected videos within the datasets. The experiments were set up in Google cloud. Where the experiments involved model training, testing, and the validation of the enhanced model.

4.5.1 Datasets

The new enhanced model was trained on three of the most used anomaly detection video datasets: Avenue, UCSD Ped1 and Ped2. The training videos were composed of normal events only while the testing videos were composed of both normal and abnormal activities. The University of California San Diego (UCSD) Ped1 and Ped2 datasets were used for training and testing. UCSD Ped1 & Ped2 are composed of 70 videos with 34 as the training set and 36 as the testing set. The video's scenery is a group of people walking in a park. Anomalies included non-pedestrian entities like bikers, skaters, carts, wheelchairs, and people walking in the grass area [35].

The Avenue dataset contains 16 training and 21 testing video clips. A total of 30652 frames are available in the dataset. These videos are captured on a campus street using a still camera. Strange actions like the running of persons and riding a bike in the walkway are the abnormal events presented. The dataset compilation and download code have been attached in Appendix I.

4.5.2 Model Parameters

The aim of training the model was to reduce the reconstruction cost of the input data. The Adam optimizer was utilized to set the learning rate automatically according to the model weights update history. Mini batches of size 4 and 3 epochs were used for training the model until the reconstruction loss stopped decreasing. Rectified Linear Unit (ReLU) activation function was used due to its ability to work well with CNNs and work with floating-point values.

4.6 Model Validation

This section highlights the methods used to ensure that the model achieved the intended purpose. Evaluation methods were deployed to check the performance of the hybrid model. The internal working and the ability to predict the anomalies are measured by established evaluation metrics like the ROC curve and F1 score. Public validated datasets were used for validation of the enhanced model.

4.6.1 Results of the Experiments

The enhancement of the Autoencoder included the addition of max-pooling and an increase of the model depth which increased the trainable features from 1.95 million to around 3.8 million learnable parameters. To effectively monitor the effectiveness of this treatment, various experiments were run using different datasets and model accuracy was recorded.

The autoencoder was used to reconstruct the videos, which were then used to calculate the reconstruction cost/error. The reconstruction cost is scaled between 0 and 1. To get a regularity score which is used to indicate anomalies. Where a low regularity score is considered an anomaly scene and high regularity score is considered normal.

The regularity score, and reconstruction cost graphs were plotted for every experiment run and the corresponding error rate and regularity score can be identified from the plots. The dataset's ground truths were used to calculate the accuracy of the model.

The regularity score indicates the anomaly score of every frame of the video, with a low regularity score indicating frame irregularity, hence indicating anomaly.

Below are sample outputs of the model before and after improvement: -

Test Case 001

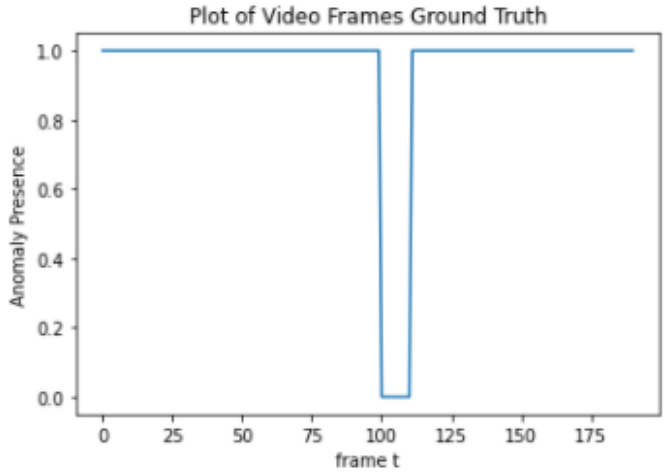


Figure 4.32: A plot of the Case 001 video ground truth

The ground truth is the actual values of anomalies and normal scenes per every frame, it indicates every anomaly present in the test video.

Before enhancement

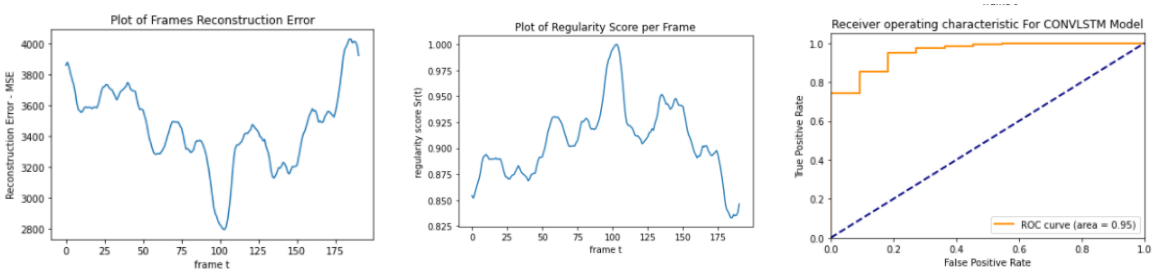


Figure 4.33: Before model improvement

After enhancement

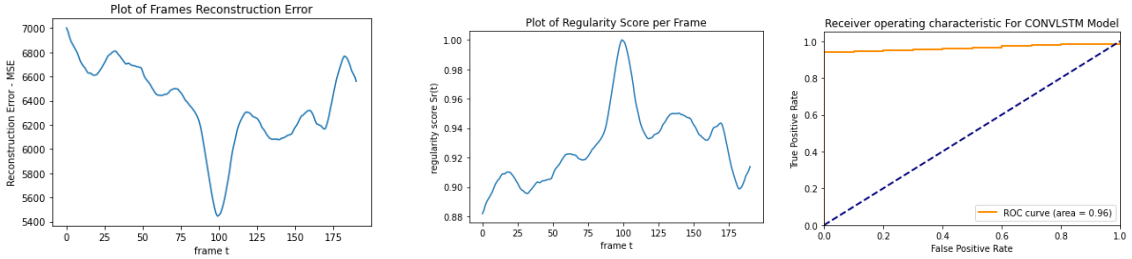


Figure 4.34: Output after enhancement

It can be noted that the graphs have some slight differences when compared with the before and after the model was improved. The improved model graphs have more smooth lines compared with the old model. This difference indicates the ability of the new improved model to deal with local minima and establish a clearer distinction between normal and abnormal scenes in the videos. Both Figures 4.33 and 4.34 show the comparison of the regularity score and mean squared error, before and after the model enhancement.

Test Case 002

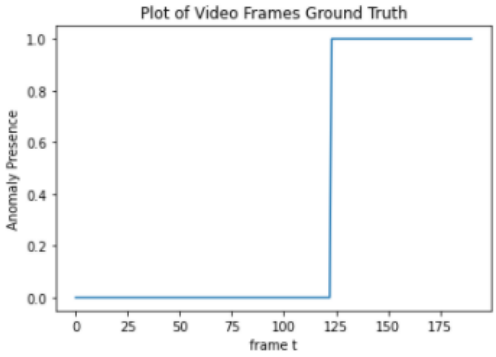


Figure 4.35: A plot of ground truth in test case 002

Figure 4.35 shows a plot of the actual anomalies and normal scenes in the test video. This plot is the true values of anomalies per every frame. The ground truths are verified and published in the dataset. Ground truth is used to measure the accuracy of the model.

Before enhancement

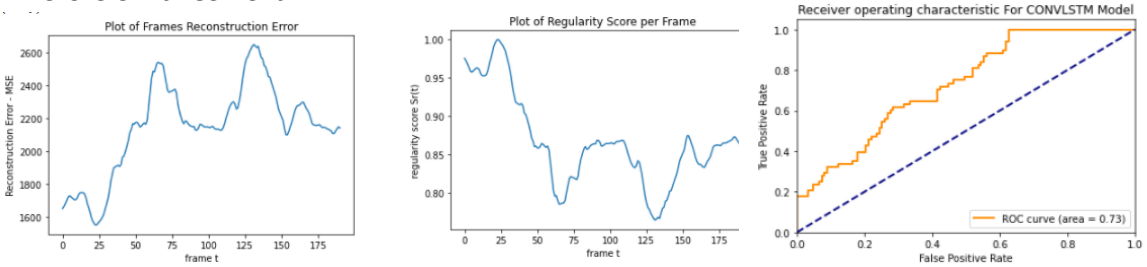


Figure 4.36: Test Case 02: Before enhancement

Figure 4.36 shows the plots of the reconstruction error, regularity score and the RoC Curve. This plot shows that the model performs poorly in this test case. Note the accuracy score. This scene is a bit complicated hence the model performs poorly. The reconstruction error plot on an ideal situation should be close to the ground truth plot in Figure 4.35

After enhancement

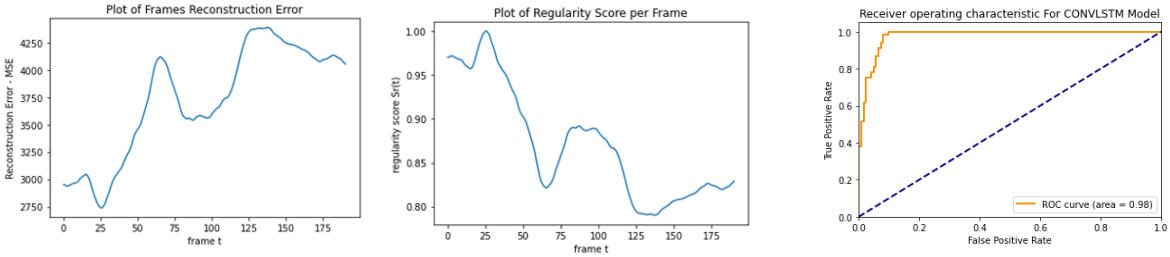


Figure 4.37: Test Case 002: After enhancement output

Figure 4.37 shows the outputs obtained in Test case 002 after the model was improved. Note the increase in the model accuracy and the smoothness of the curve. The model performs better after it was enhanced. It can predict well despite the scene complexity.

Test Case 003

A plot of ground truths

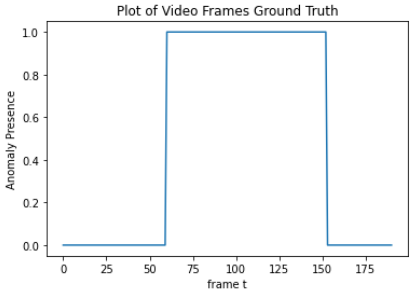


Figure 4.38: Test Case 003 Ground Truth Plot

Before enhancement

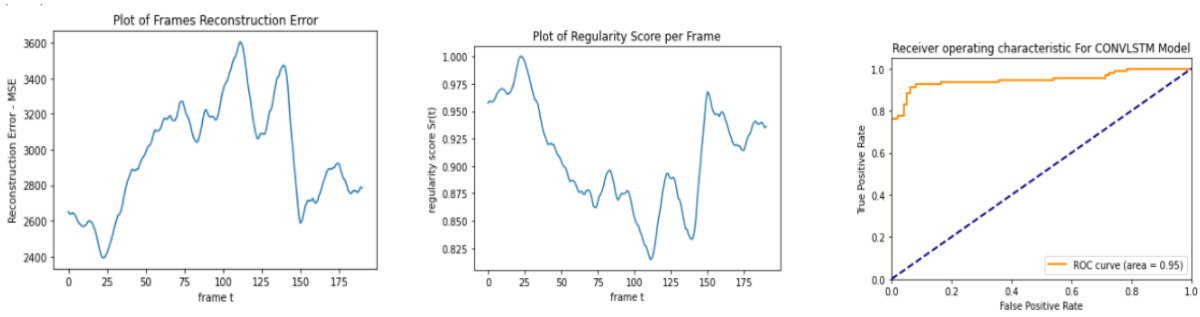


Figure 4.39: Test Case 003: before enhancement

Figure 4.39 shows the reconstruction error, regularity score and the ROC curve. The plot of the reconstruction error shows close resemblance to the ground truth although at the top region with anomalies it has some ramps indicating local minima. After the model was improved as shown in figure 4.40, the plots of reconstruction error and the regularity score smoothens with local minima being flattened. This shows the stability of the model in the prediction of anomalies.

Test Case 003 After enhancement

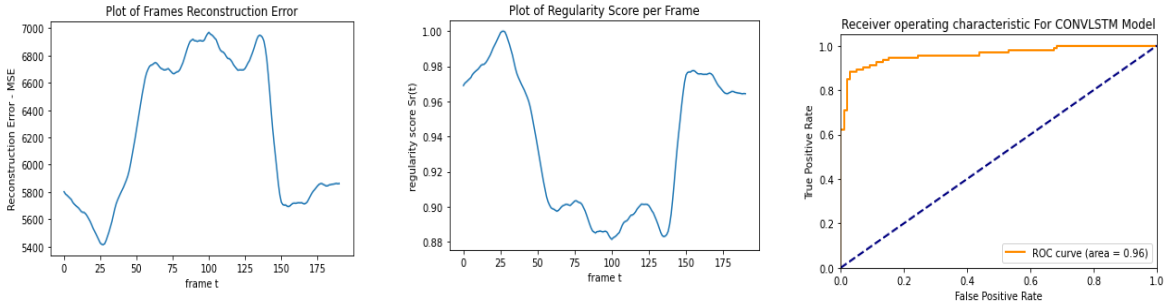


Figure 4.40: Test Case 003 after the enhancement

The low regularity score indicates the presence of anomalies as illustrated below in the Figure 4.41.

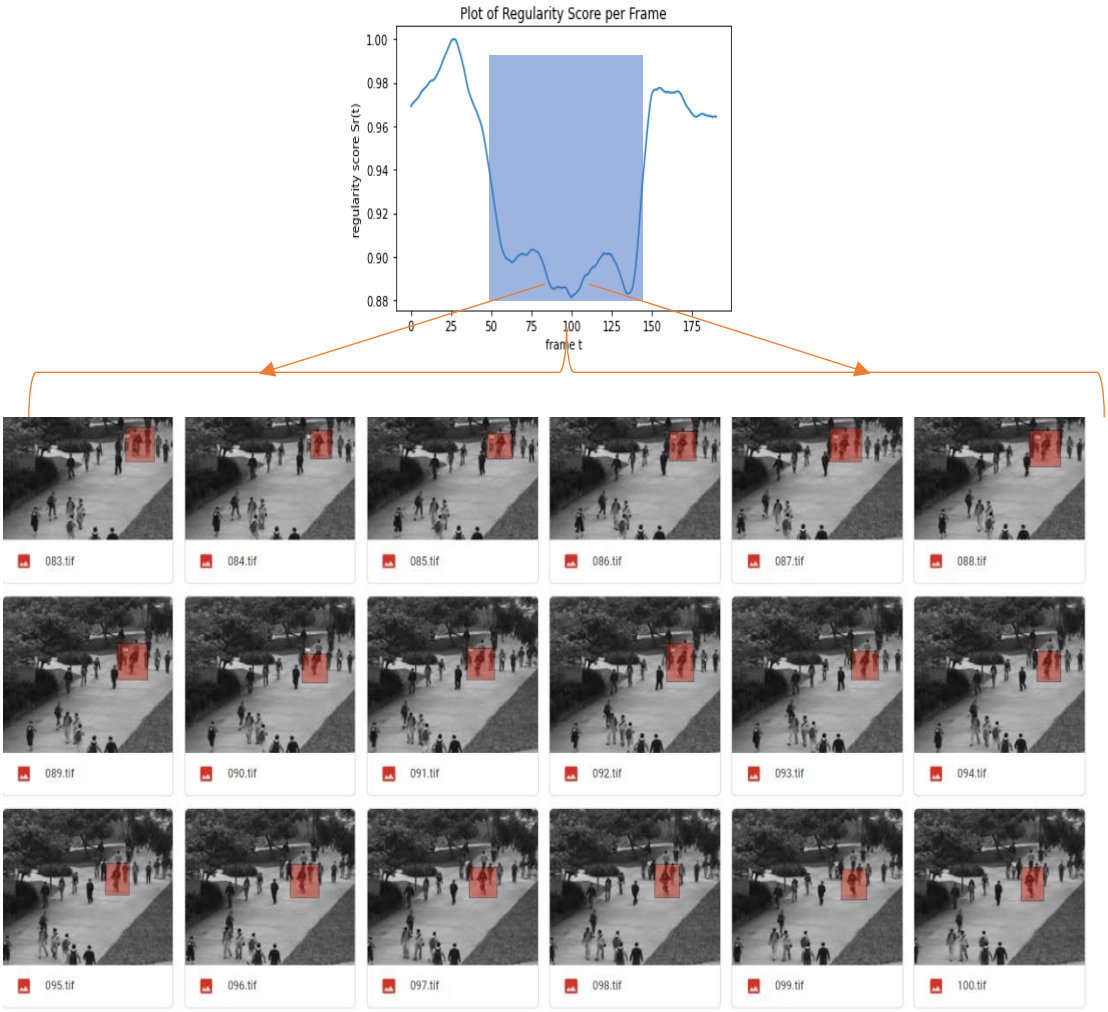


Figure 4.41: Illustration of Anomalies in the test case 003 video

Figure 4.41 illustrates the presence of anomalies in the test video. For instance, the anomaly plot shows that from the 50th to 150th frames there are anomalies present. A part of the frames in between is shown in figure 43. The present anomalies are the presence of non-pedestrian entities in the park. Anomalies have low regularity scores hence the depression in the regularity plot. From figure 4.41, it was established that model improvement addresses local minima by flattening them such that anomalies can be identified with ease.

4.6.2 Comparison of Models Accuracy

Different datasets were used to assess the model and the test cases were run in every dataset and the RoC curve accuracy scores were averaged to have a single accuracy score that was compared to the old model. The accuracy scores were tabulated in the table below for comparison. The model was trained and evaluated on UCSD Ped1 and Ped2 public validated datasets.

Table 4.3: Comparison of the RoC-AUC Scores

Model	UCSD Ped1	UCSD Ped2	Average
Old Autoencoder Model [50]	86.14	90.23	84.86
Enhanced Autoencoder Model	**92.63**	**95.56**	94.10

Table 4.3 shows evidence of improvement in the average percentage accuracy of the test cases per dataset. Notably, the accuracy has been increased by some percentage, due to the improvements made to the model structure. From the average column, we can note that the model accuracy increased by 9.2%. This increase reduces the errors in the anomaly prediction. This contribution reduces false alarm errors and increases anomalies detected during prediction of anomalies.

4.6.3 Test of statistical significance

It is important to evaluate whether the improvements made to the model caused the increase in model accuracy. This test establishes whether the enhancements were significant. The accuracy score is not normally distributed and hence a distribution-free method was used. Sign test method was selected to test for the statistical significance of the improvement. This method does not assume that data is normally distributed, but it treats the data as binomial distribution. The test of significance was conducted at different alpha levels and the following hypothesis was formulated.

Formulated hypothesis

Null Hypothesis

H_0 - No significant accuracy improvement in the enhanced STAE model after depth and pooling tuning.

Research Hypothesis

H_1- There is significant accuracy improvement in the enhanced STAE model after depth and pooling tuning.

The accuracy scores from different datasets were recorded before and after the model was enhanced and were used to test the significance of the improvement using the sign test method as follows:

Test of Significance of the Model Improvement

Test Case	% Accuracy Before Model Improvement	% Accuracy After Model improvement	% Improvement	Sign
1	74	99	25	+
2	100	100	0	0
3	91	99	8	+
4	100	100	0	0
5	84	81	-3	-
6	44	70	26	+
7	99	100	1	+
8	68	100	32	+
9	84	81	-3	-
10	100	100	0	0
11	95	96	1	+
12	94	98	4	+
13	100	100	0	0
14	93	97	4	+
15	48	49	1	+
16	31	61	30	+
17	95	96	1	+
18	94	98	4	+
19	100	100	0	0
20	93	97	4	+
21	48	49	1	+
22	31	68	37	+
23	75	80	5	+

Figure 4.42: Sample data used for test of significance

Figure 4.42 displays, the test cases conducted, that were used to test for the significance of the enhancement. The test cases are the total number of tests conducted from the UCSD Ped1 and Ped 2 test datasets after the test experiments were conducted. These test cases were randomly sampled from the 36 available test subjects at 95% confidence level and a margin error of 10%.

The test statistic for the sign test is the number of positive or negative signs. In this case, we observe 16 positive and 2 negative signs. Zeros are not considered as they do not indicate any significant improvement.

Test statistic for the Sign test is the least number of either positive or negative signs [84]and it follows binomial distribution with the probability of P = 0.5 and n as the number of subjects in the study.

Two-tailed test was used since improvement of the model had both negative and positive effects to the model prediction accuracy [84]. Two-sided test hypothesizes a repetitive behavior like the prediction accuracy before and after the model improvement.

P values were calculated from the binomial distribution formula below: -

$$P(x\ success) = \frac{n!}{x!\,(n-x)!}\ p^{x}(1-p)^{n-x}$$

Since the number of positive is not equal to the negative sign, we proceed to calculate the p-value,

A two-tailed test with a Significance level of 0.05 and,

Sample size n=18[excluding 0], number of successes=2 since the positive outcomes are 16.

When calculating the p-value,

It was assumed that the data follows a binomial distribution, which has a probability of 0.5 such that; Null hypothesis is that there is an equal number of signs (+) and (-).

An assumption of the null hypothesis is made to set up a binomial experiment with 0.5 as probability, 18 as the number of trials and 2 as the success since success is taken as the least positive or negative sign.

At the significance level $\alpha=0.05$

The P-value is 0.000583 and since it is less than the significance level of 0.05, the null hypothesis was rejected.

At the significance level $\alpha=0.01$

The calculated P-Value is 0.000583 and it is still less than the significance level of 0.01, the null hypothesis was rejected.

Therefore, the conclusion was made that, there was evidence of improvement in model accuracy after it was enhanced.

4.7 Summary

This chapter captured the implementation of the study objectives, the results obtained and the interpretation of the results. Deep learning models were reviewed by conducting analysis and a model was selected for improvement. The model was enhanced through the increase of the model depth, regularization, and introduction of the clustering algorithm. Depth of the spatial temporal autoencoder was added from 15 layers to 29 layers. The accuracy of the enhanced model was noted to increase significantly by increase from average of 84.8% to 94.1%. The model accuracy was compared and validated to establish the significance of the improvement.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATIONS

5.1 Summary

This chapter concludes the Thesis, by highlighting the research landmarks and the delivered enhanced deep learning model by shedding light on important highlights. The chapter entails how the model was improved, and the new improved model. The process of model selection, model debugging, and model improvement contributes to autoencoder development and intelligent surveillance.

The first part of the thesis analysis the deep learning models implemented for surveillance videos anomaly detection. Several categories of the models are discovered in the review, and current and emerging trends were also discovered. The implemented models were grouped into several categories namely: transfer learning, autoencoders, ensemble learning and continual learning.

The research study was driven by the need to improve and refine anomaly detection in surveillance videos. This work was focused on the reduction of errors in anomaly detection as identified in the research gap. It can be noted that the model was enhanced by adding its depth, introducing of max-pooling function, and adding an automatic classifier. The new enhanced model extracted more features from the videos after enhancement. Extraction of more features made it possible for the model to reduce scene complexity and make more accurate anomaly predictions. Hence, the overall increase in the model accuracy. The enhanced model can be seen in appendix v.

The last step was to validate the improvement. The enhanced model was trained and tested using the UCSD Ped1 and Ped2 datasets. Anomaly prediction accuracy of the old and the new enhanced models were compared, and significant improvement was noted.

5.2 Conclusion

The enhanced model achieves a significant increase in anomaly prediction accuracy; hence it minimizes the error rate. This study established that the depth of autoencoder models while working in video anomaly detection matters. The spatial parts of the autoencoder model were made deeper to extract appearance, texture, and position features from the imagery data. The depth tuning included addition of regularization layers as well. Application of regularization parameters, reduced the overfitting in the autoencoders, hence fostering model generation capability.

The new enhanced model is deeper with a total of 29 layers compared with the old model which had 15 layers. This model has more trainable parameters with a total of 3,710,157 parameters compared to the old model with 1,958,209 learnable parameters.

5.3 Recommendations

Some of notable recommendations noted within the research can help in shaping the quality of research and the future of intelligence surveillance.

5.3.1 Recommendations to Policy

Other researchers with interest in this area and industries seeking to push further intelligence surveillance should consider video reconstruction error while exploring the

unsupervised anomaly detection. In addition, progressive training should be built on the autoencoder model to allow room for the new data. The problem of novel anomalies can be a future consideration by incorporating continual learning.

Use of Graphics Processing Units (GPUs) as the default computing engine is recommended. GPUs are more efficient while dealing with complex data like videos.

5.3.2 Recommendations for Future Works

In future, more work should be done on real-time anomaly detection and diversification of anomaly detection in satellite surveillance, traffic surveillance and other areas can be considered. Real-time anomaly detection and continually learning models will be a closer step towards deployable intelligent surveillance.

REFERENCES

[1] R. Yadav and M. Rai, "Advanced Intelligent Video Surveillance System (AIVSS): A Future Aspect," *Research Gate,* 2018.

[2] X. Zheng, Y. Zhiguo, M. Lin and Z. Hui, "The Big Data Analysis of the Next Generation Video Surveillance System for Public Security," *Research Gate,* pp. 171-175, 2016.

[3] M. Chowdhury, J. Gao and R. Islam, "Human Surveillance System for Security Application," in *Institute for Computer Sciences, Social Informatics and Telecommunications Engineering,* Bathurst, Australia, 2015.

[4] Surfshark, "Surveillance Cities," Surfshark, November 2020. [Online]. Available: https://surfshark.com/surveillance-cities. [Accessed 25 August 2021].

[5] Tryolabs, "Guide to Video Analytics: Applications and Opportunities," 2020. [Online]. Available: https://tryolabs.com/resources/video-analytics-guide/.

[6] C. Nicholson, "Artificial Intelligence (AI) vs. Machine Learning vs. Deep Learning," 2019. [Online]. Available: https://pathmind.com/wiki/ai-vs-machine-learning-vs-deep-learning.

[7] A. Sarkar, "Human Activity and Behavior Recognition in Videos. A Brief Review," 2014. [Online]. Available: https://www.grin.com/document/276054.

[8] A. Shrestha and A. Mahmood, “Review of Deep Learning Algorithms and Architectures,” *IEEE,* pp. 53040-53065, 2019.

[9] S. Dobilas, “LSTM Recurrent Neural Networks — How to Teach a Network to Remember the Past,” Medium, 6 February 2021. [Online]. Available: https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e. [Accessed 29 June 2022].

[10] C. Nicholson, “A Beginner's Guide to LSTMs and Recurrent Neural Networks,” 2018. [Online]. Available: https://pathmind.com/wiki/lstm.

[11] A. Borner, “What is Deep Learning and How Does it Work? | Content Simplicity,” 2019. [Online]. Available: https://contentsimplicity.com/what-is-deep-learning-and-how-does-it-work/.

[12] M. Hargrave, “Deep Learning,” 30 April 2019. [Online]. Available: https://investopedia.com/terms/d/deep-learning.asp.

[13] J. Brownlee, “What is Deep Learning?,” 16 August 2019. [Online]. Available: https://machinelearningmastery.com/what-is-deep-learning/.

[14] A. Xavier, “An introduction to ConvLSTM,” 25 March 2019. [Online]. Available: https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7.

[15] S. Shimozaki, “Self-Supervised Video Anomaly Detection,” 7 5 2017. [Online]. Available: https://launchpad.ai/blog/video-anomaly-detection.

[16] S. J. Shri and S. Jothilakshmi, “Anomaly Detection in Video Events using Deep Learning,” *International Journal of Innovative Technology and Exploring Engineering (IJITEE),* pp. 1313-1316, 2019.

[17] B. M. Nair, “Deep Dive into Convolutional 3D features for action and activity recognition (C3D),” Medium.com, 23 July 2018. [Online]. Available: https://medium.com/@nair.binum/quick-overview-of-convolutional-3d-features-for-action-and-activity-recognition-c3d-138f96d58d8f. [Accessed 22 August 2021].

[18] H. Selat, “Anomaly Detection in Videos using LSTM Convolutional Autoencoder,” 15 October 2019. [Online]. Available: https://towardsdatascience.com/prototyping-an-anomaly-detection-system-for-videos-step-by-step-using-lstm-convolutional-4e06b7dcdd29.

[19] J. R. Medel and A. Savakis, “Anomaly Detection in Video Using Predictive Convolutional Long Short-Term Memory Networks,” *Research Gate,* 2016.

[20] R. Chalapathy and S. Chwala, “Deep Learning For anomaly Detection: A Survey,” *arxvi,* 2019.

[21] K. V. Pawar and V. Attar, “Deep learning approaches for video-based anomalous activity detection,” *World Wide Web,* p. 22, 27 May 2018.

[22] S. Waqas, C. Chen and S. Mubarak, “Real-world Anomaly Detection in Surveillance Videos,” *Computer Vision Fiundation,* pp. 6479-6488, 2016.

[23] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, Cambridge: MIT Press, 2016.

[24] G. Xie and J. Lai, “An Interpretation of Forward-Propagation and Back-Propagationof DNN,” *Pattern Recognition andComputer Vision,* 2018.

[25] M. A. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

[26] K. O’Shea and R. Nash, “An Introduction to Convolutional Neural Networks,” *Research Gate,* pp. 1-11, 2015.

[27] K. Wiggers, “AI Guardsman uses computer vision to spot shoplifters,” 26 June 2018. [Online]. Available: https://venturebeat.com/2018/06/26/ai-guardsman-uses-computer-vision-to-spot-shoplifters/.

[28] G. H. Martınez, “OpenPose: Whole-Body Pose Estimation,” Carnegie Mellon University, Pittsburgh, Pennsylvania , 2019.

[29] M. Sabokrou, M. Fayyaz, M. Fathy, Z. Moayed and R. Klette, “Deep-Anomaly: Fully Convolutional Neural Network for Fast Anomaly Detection in Crowded Scenes,” *Computer Vision and Image Understanding,* pp. 1-25, 2018.

[30] S. Bansal, “3D Convolutions: Understanding + Use Case,” 2019. [Online]. Available: kaggle.com/shivamb/3d-convolutions-understanding-use-case.

[31] A. Kushwaha, A. Mishra, K. Kamble, R. Janbhare and A. Pokhare, “Theft Detection using Machine Learning,” *IOSR Journal of Engineering (IOSRJEN),* pp. 67-71, 2018.

[32] M. Sabokrou, M. Fayyaz, M. Klette and R. Fathy, “Deep-Cascade: Cascading 3D Deep Neural Networks for Fast Anomaly Detection and Localizaton in Crowded Scenes,” *IEEE Transactions on Image Processing,* pp. 1992-2004, 2017.

[33] M. U. Farooq, N. A. Khan and M. S. Ali, “Unsupervised Video Surveillance for Anomaly Detection of Street Traffic,” *(IJACSA) International Journal of Advanced Computer Science and Applications,,* pp. 270-275, 2017.

[34] J. Hui, “How to start a Deep Learning project?,” Medium, 1 Mar 2018. [Online]. Available: https://jonathan-hui.medium.com/how-to-start-a-deep-learning-project-d9e1db90fa72. [Accessed 28 Sept 2021].

[35] UCSD, “UCSD Anomaly Detection Dataset,” UCSD, 2014. [Online]. Available: http://www.svcl.ucsd.edu/projects/anomaly/dataset.html. [Accessed 10 May 2021].

[36] J. Hui, “Deep Learning designs (Part 3),” Medium, 03 March 2018. [Online]. Available: https://jonathan-hui.medium.com/deep-learning-designs-part-3-e0b15ef09ccc. [Accessed 28 September 2021].

[37] J. Hui, “Visualize Deep Network models and metrics (Part 4),” Medium, 1 March 2018. [Online]. Available: https://jonathan-hui.medium.com/visualize-deep-network-models-and-metrics-part-4-9500fe06e3d0. [Accessed 28 Sept 2021].

[38] J. Hui, “Debug a Deep Learning Network (Part 5),” Medium, 1 March 2018. [Online]. Available: https://jonathan-hui.medium.com/debug-a-deep-learning-network-part-5-1123c20f960d. [Accessed 28 Sept 2021].

[39] S. Doshi, “Various Optimization Algorithms For Training Neural Network,” Towards Data Science, 13 Jan 2019. [Online]. Available: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6. [Accessed 28 Sept 2021].

[40] J. L. Ba, J. R. Kiros and G. E. Hinton, “Layer Normalization,” *arXiv preprint* , vol. arXiv:1607.06450, 2016.

[41] J. Hui, “Improve Deep Learning Models performance & deep network tuning (Part 6),” Medium, 2 March 2018. [Online]. Available: https://jonathan-hui.medium.com/improve-deep-learning-models-performance-network-tuning-part-6-29bf90df6d2d. [Accessed 28 Sept 2021].

[42] T. Gupta, “Deep Learning: Regularization Notes,” Towards Data Science, 20 August 2017. [Online]. Available: https://towardsdatascience.com/deep-learning-regularization-notes-29df9cb90779. [Accessed 28 Sept 2021].

[43] S. N. Ravi, T. Dinh, V. S. Lokhande and V. Singh, “Explicitly Imposing Constraints in Deep Networks via Conditional Gradients Gives Improved Generalization and Faster Convergence,” in *The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, Honolulu, 2019.

[44] R. K. Malaiya, D. Kwon, J. Kim, S. C. Suh, H. Kim and I. Kim, “An Empirical Evaluation of Deep Learning for Network Anomaly Detection,” *ResearchGate*, 2018.

[45] K. Nighania, “Various Ways to Evaluate a Machine Learning Models Performance,” 6 December 2018. [Online]. Available: http://www.towardsdatascience.com/variou-waays-to-evaluate-a-machine-learning-models-performance-230446055f15.

[46] FormPlus, “Experimental Research Designs: Types, Examples & Methods,” FormPlus, 23 September 2021. [Online]. Available: https://www.formpl.us/blog/experimental-research. [Accessed 25 September 2021].

[47] S. Aberkane and M. Elarbi, “Deep Reinforcement Learning for Real-world Anomaly Detection in Surveillance Videos,” in *2019 6th International Conference*

on Image and Signal Processing and their Applications (ISPA), Mostaganem, Algeria, 2019.

[48] L. P. Cinelli, "Anomaly Detection in Surveillance Videos using Deep Residual Networks," Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2017.

[49] M. F. M. M. Sabokrou, "Video anomaly detection and localisation based on the sparsity and reconstruction error of auto-encoder," *Electronic Letters,* vol. 52, no. 13, pp. 1122-1124, 2016.

[50] Y. S. Chong and Y. H. Tay, "Abnormal Event Detection in Videos Using Spatiotemporal Autoencoder," *arxiv,* vol. 1701, no. 01546v1, 2017.

[51] W. Liu, W. Luo, D. Lian and S. Gao, "Future Frame Prediction for Anomaly Detection – A New Baseline," *Computer Vision Foundation,* vol. abs/1712.09867, 2017.

[52] Business Research Methodology, "Purposive sampling," Business Research Methodology, 01 Jan 2021. [Online]. Available: https://research-methodology.net/sampling-in-primary-data-collection/purposive-sampling/. [Accessed 25 June 2022].

[53] T. Srivastava, "11 Important Model Evaluation Metrics for Machine Learning Everyone should know," 6 August 2019. [Online]. Available:

https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/.

[54] Y. S. Chong and Y. H. Tay, “Modeling Representation of Videos for Anomaly Detection using Deep Learning: A Review,” *Research Gate,* 2015.

[55] A. Ullah, K. Muhammad and S. W. Baik, “Action Recognition in Video Sequences using Deep Bi-Directional LSTM With CNN Features,” *IEEE Access,* pp. 1155-1166, 2017.

[56] E. Duman and O. A. Erdem, “Anomaly Detection in Videos Using Optical Flow and Convolutional Autoencoder,” *IEEE Access*, vol. 7, pp. 183914 - 183923, 2019.

[57] K. Liu, M. Zhu, H. Fu, H. Ma and T.-S. Chua, “Enhancing Anomaly Detection in Surveillance Videos with Transfer Learning from Action Recognition,” *Proceedings of the 28th ACM International Conference on Multimedia,* pp. 4664-4668, 2020.

[58] Y. Zahid, M. A. Tahir and M. N. Durrani, “Ensemble Learning Using Bagging And Inception-V3 For Anomaly Detection In Surveillance Videos,” in *2020 IEEE International Conference on Image Processing (ICIP)*, Abu Dhabi, United Arab Emirates, 2020.

[59] K. Doshi and Y. Yilmaz, “Any-Shot Sequential Anomaly Detection in Surveillance Videos,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops,* pp. 934-935, 2020.

[60] K. Doshi and Y. Yilmaz, “Online anomaly detection in surveillance videos with asymptotic bound on false alarm rate,” *Pattern Recognition,* vol. 114, p. 107865, 2021.

[61] A. R. Pathak, M. Pandey and S. Rautaray, “Application of Deep Learning for object detection,” *International Conference on Computational Intelligence and Data Science (ICCDS 2018),* pp. 1706-1717, 2018.

[62] T. S. Nazare, R. F. de Mello and M. A. Ponti, “Are pre-trained CNNs good feature extractors for anomaly detection in surveillance videos?,” *eprint arXiv,* no. 1811.08495v1, 2018.

[63] S. Bansod and A. Nandedkar, “Transfer learning for video anomaly detection,” *Journal of Intelligent & Fuzzy Systems,* vol. 36, no. 3, pp. 1967-1975, 2019.

[64] W. Ullah, A. Ullah, T. Hussain, Z. A. Khan and S. W. Baik, “An Efficient Anomaly Recognition Framework Using an Attention Residual LSTM in Surveillance Videos,” *AI-Enabled Advanced Sensing for Human Action and Activity Recognition,* vol. 21, 2021.

[65] L. P. Cinelli, L. A. Thomaz, A. F. d. Silva, E. A. B. d. Silva and S. L. Netto, "Foreground Segmentation for Anomaly Detection in Surveillance Videos Using Deep Residual Networks," *XXXV SIMPOSIO BRASILEIRO DE TELECOMUNICAC¸ ´OES E PROCESSAMENTO DE SINAIS,* pp. 3-6, 2017.

[66] T.-N. Nguyen and J. Meunier, "Anomaly Detection in Video Sequence with Appearance-Motion Correspondence," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV),* 2019.

[67] A. Ramchandran and A. K. Sangaiah, "Unsupervised deep learning system for local anomaly event detection in crowded scenes," *Multimedia Tools and Applications,* vol. 79, no. 47/48, p. 35275–35295, 2020.

[68] Y. Zhao, B. Deng, C. Shen, Y. Liu, H. Lu and X.-S. Hua, "Spatio-Temporal AutoEncoder for Video Anomaly Detection," *Proceedings of the 25th ACM international conference on Multimedia,* pp. 1933-1941, 2017.

[69] Y. Zhao, B. Deng, C. Shen, Y. Liu, H. Lu and X.-S. Hua, "Spatio-Temporal AutoEncoder for Video Anomaly Detection," *Proceedings of the 25th ACM international conference on Multimedia,* pp. 1933-1941, 2017.

[70] H. Wu, J. Shao, X. Xu, F. Shen and H. Shen, "A System for Spatiotemporal Anomaly Localization in Surveillance Videos," *Proceedings of the 25th ACM international conference on Multimedia,* pp. 1225-1226, 2017.

[71] S. Bhakat and G. Ramakrishnan, “Anomaly Detection in Surveillance Videos,” *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data,* p. 252–255, 2019.

[72] M. Hasan, J. Choi, J. Neumann, A. K. Roy-Chowdhury and L. S. Davis, “Learning Temporal Regularity in Video Sequences,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 733-742, 2016.

[73] T.-H. Vu, J. Boonaert, S. Ambellouis and A. Taleb-Ahmed, “Multi-Channel Generative Framework and Supervised Learning for Anomaly Detection in Surveillance Videos,” *Human Activity Recognition Based on Image Sensors and Deep Learning,* vol. 21, no. 9, p. 3179, 2021.

[74] R. Chalapathy, A. K. Menon and S. Chawla, “Robust, Deep and Inductive Anomaly Detection,” *Machine Learning and Knowledge Discovery in Databases,* vol. 10534, 2017.

[75] K. T. Nguyen, D. T. Dinh, M. N. Do and M. T. Tran, “Anomaly Detection in Traffic Surveillance Videos with GAN-based Future Frame Prediction,” *Proceedings of the 2020 International Conference on Multimedia,* pp. 457-463, 2020.

[76] K. Doshi and Y. Yilmaz, “Continual Learning for Anomaly Detection in Surveillance Videos,” in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops,* Seattle, WA, USA, 2020.

[77] K. Kavikuil and J. Amudha, “Leveraging Deep Learning for Anomaly Detection in Video Surveillance,” *First International Conference on Artificial Intelligence and Cognitive Computing,* vol. 815, no. I, pp. 239-247, 2018.

[78] W. Ullah, A. Ullah, I. U. Haq, K. Muhammad, M. Sajjad and S. W. Baik, “CNN features with bi-directional LSTM for real-time anomaly detection in surveillance networks,” *Multimedia Tools and Applications ,* p. 16979–16995, 2021.

[79] M.Murugesan and S.Thilagamani, “Efficient anomaly detection in surveillance videos based on multi layer perception recurrent neural network,” in *Microprocessors and Microsystems*, 2020.

[80] V. A. Karishma Pawar, “Application of Deep Learning for Crowd Anomaly Detection from Surveillance Videos,” in *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, Noida, India, 2021.

[81] N. Nasaruddin, K. Muchtar, A. Afdhal and A. P. J. Dwiyantoro, “Deep anomaly detection through visual attention in surveillance videos,” *Journal of Big Data,* vol. 7, no. 87, 2020.

[82] A. Khaleghi and M. S. Moin, “Improved anomaly detection in surveillance videos based on a deep learning method,” in *2018 8th Conference of AI & Robotics and 10th RoboCup Iranopen International Symposium (IRANOPEN),* Qazvin, Iran, 2018.

[83] I. S.-B. Víctor Suárez-Paniagua, “Evaluation of pooling operations in convolutional architectures for drug-drug interaction extraction,” in *BMC Bioinformatics 19, 209 (2018).*, 2018.

[84] W. W. LaMorte, “Nonparametric Tests,” Boston University , 4 May 2017. [Online]. Available: https://sphweb.bumc.bu.edu/otlt/mph-modules/bs/bs704_nonparametric/BS704_Nonparametric5.html#:~:text=The%20test%20statistic%20for%20the%20Sign%20Test%20is%20the%20smaller,details%20on%20the%20binomial%20distribution).. [Accessed 20 7 2022].

[85] N. Rasheed, Khan, Shoab and A. Khalid, “Tracking and Abnormal Behavior Detection in Video Surveillance Using Optical Flow and Neural Networks,” in *Research Gate*, 2014.

[86] A. Mordvinster and K. Abdi, “Background Subtraction,” 2013. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py-video/py_bg_subtraction/py_bg_subtraction.hml.

[87] A. Kappeler, R. D. Morris, A. R. Kamat, N. Rasiwasia and G. Aggarval, “Combining Deep Learning and Unsupervised Clustering to Improve Scene Recognition Performance,” NorthWestern University, Evanston Illnois, 2015.

[88] C. Viroli and G. J. McLachlan, “Deep Gaussian Mixture Models,” 21 Nov 2017. [Online]. Available: https://arxiv.org/pdf/1711.06929.pdf.

[89] S. S. Filho, P. Drews-Jr and S. Botelho, “Detecting Changes in 3D Maps using Gaussian distribution,” *Intelligent Robotics and Automation Group (NAUTEC),* 2016.

[90] Y. Zhu and S. Newsman, “Motion-Aware Feature for Improved Video Anomaly Detection,” University of California, Merced, USA, 2019.

[91] R. Revathi and M. Hemalatha, “An Emerging Trend Of Feature Extraction Method In Video Processing,” CS & IT-CSCP 2012, TamilNadu, India, 2012.

[92] E. Variani, E. McDermott and G. Heigold, “A Gaussian Mixture Model Layer Jointly Optimized With Discriminative Features Within A Deep Neural Network Architecture.,” Google Inc., California, 2014.

[93] H. M. Kun Liu, “Exploring Background-bias for Anomaly Detection in Surveillance Videos,” *Proceedings of the 27th ACM International Conference on Multimedia,* pp. 1490-1499, 2019.

[94] R. V. H. M. Colque, C. Caetano and M. T. L. d. Andrade, “Histograms of Optical Flow Orientation and Magnitude and Entropy to Detect Anomalous Events in Videos,” *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY,* vol. 27, no. 3, pp. 673-682, 2017.

[95] C. Lu, J. Shi and J. Jia, “Avenue Dataset for Abnormal Event Detection,” The Chinese Univeristy of Hong Kong, 2013. [Online]. Available:

http://www.cse.cuhk.edu.hk/leojia/projects/detectabnormal/dataset.html. [Accessed 10 May 2021].

[96] W. Badr, “Auto-Encoder: What Is It? And What Is It Used For? (Part 1),” towards data science, 22 April 2019. [Online]. Available: https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726. [Accessed 10 May 2021].

[97] V. Jain, “Everything you need to know about “Activation Functions” in Deep learning models,” Towards Data Science, 30 December 2019. [Online]. Available: https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253. [Accessed 28 Sept 2021].

[98] Y. Ding, “Some Strategies for Machine Learning Projects,” Towards Data Science, 20 April 2020. [Online]. Available: https://towardsdatascience.com/some-strategies-for-machine-learning-projects-5f2f32c34635. [Accessed 28 Sept 2020].

[99] R. Schmelzer, “The Five Ways To Build Machine Learning Models,” Forbes, 30 May 2021. [Online]. Available: https://www.forbes.com/sites/cognitiveworld/2021/05/30/the-five-ways-to-build-machine-learning-models/?sh=1b7014f311a8. [Accessed 28 Sept 2021].

APPENDICES

APPENDIX I:

DATASET COMPILATION CODE

```
!wget -O ped2.tar.gz http://101.32.75.151:8181/dataset/ped2.tar.gz
```

```
--2021-06-12 15:21:29--  http://101.32.75.151:8181/dataset/ped2.tar.gz
Connecting to 101.32.75.151:8181... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://public.bn.files.1drv.com/y4mJVqs0pGeVMUhtF3RWGKHmWxSS0KeHPdy6BBizIIv3nfRd97sF_CP4C0
--2021-06-12 15:21:29--  https://public.bn.files.1drv.com/y4mJVqs0pGeVMUhtF3RWGKHmWxSS0KeHPdy6BBizIIv
Resolving public.bn.files.1drv.com (public.bn.files.1drv.com)... 13.107.42.12
Connecting to public.bn.files.1drv.com (public.bn.files.1drv.com)|13.107.42.12|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 101645892 (97M) [application/x-gzip]
Saving to: 'ped2.tar.gz'

ped2.tar.gz         100%[===================>]  96.94M  8.83MB/s    in 12s

2021-06-12 15:21:42 (8.32 MB/s) - 'ped2.tar.gz' saved [101645892/101645892]
```

```
!wget -O ped1.tar.gz http://101.32.75.151:8181/dataset/ped1.tar.gz
```

```
--2021-06-12 15:25:26--  http://101.32.75.151:8181/dataset/ped1.tar.gz
Connecting to 101.32.75.151:8181... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://public.bn.files.1drv.com/y4mfL-OKhITsO3ng7c4U4th_sCLUsCXV-sacGzyoFkGs3lK1ixKpupcGxqu
--2021-06-12 15:25:26--  https://public.bn.files.1drv.com/y4mfL-OKhITsO3ng7c4U4th_sCLUsCXV-sacGzyoFkGs
Resolving public.bn.files.1drv.com (public.bn.files.1drv.com)... 13.107.42.12
Connecting to public.bn.files.1drv.com (public.bn.files.1drv.com)|13.107.42.12|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 244566523 (233M) [application/x-gzip]
Saving to: 'ped1.tar.gz'

ped1.tar.gz         100%[===================>] 233.24M  13.3MB/s    in 23s

2021-06-12 15:25:50 (10.1 MB/s) - 'ped1.tar.gz' saved [244566523/244566523]
```

The above code section portrays the collection of the datasets from the public libraries of validated datasets. Data were downloaded and stored in the cloud for easy access.

APPENDIX II:

VIDEO FRAMES EXTRACTION CODE

```
from concurrent.futures import ProcessPoolExecutor, as_comp
leted
import cv2
import multiprocessing
import os
import sys
import os
from os import listdir
import skimage.transform
from skimage import color
from os.path import isfile, join
import numpy as np
import numpy
from datetime import datetime
from pathlib import Path
from os.path import basename
import glob
from random import sample

videopath='/content/drive/My Drive/datasets/ucf_crime/Anoma
ly-Videos/RoadAccidents'
framespath='/content/drive/My Drive/datasets/ucf_crime/test
ing_anomaly_videos/'

def print_progress(iteration, total, prefix='', suffix='',
decimals=3, bar_length=100):
    """
    Call in a loop to create standard out progress bar

    """

    format_str = "{0:." + str(decimals) + "f}"  # format th
e % done number string
    if total==0:
      total=1
    percents = format_str.format(100 * (iteration / float(t
otal)))  # calculate the % done
    filled_length = int(round(bar_length * iteration / floa
t(total)))  # calculate the filled bar length
    bar = '#' * filled_length + '-
' * (bar_length - filled_length)  # generate the bar string
```

```
    sys.stdout.write('\r%s |%s| %s%s %s' % (prefix, bar, pe
rcents, '%', suffix)),  # write out the bar
    sys.stdout.flush()  # flush to stdout

def extract_frames(video_path, frames_dir, overwrite=False,
 start=-1, end=-1, every=1):
    """
    Extract frames from a video using OpenCVs VideoCapture
    :param video_path: path of the video
    :param frames_dir: the directory to save the frames
    :param overwrite: to overwrite frames that already exis
t?
    :param start: start frame
    :param end: end frame
    :param every: frame spacing
    :return: count of images saved
    """

    video_path = os.path.normpath(video_path)  # make the p
aths OS (Windows) compatible
    frames_dir = os.path.normpath(frames_dir)  # make the p
aths OS (Windows) compatible

    video_dir, video_filename = os.path.split(video_path)
# get the video path and filename from the path

    assert os.path.exists(video_path)  # assert the video f
ile exists

    capture = cv2.VideoCapture(video_path)  # open the vide
o using OpenCV

    if start < 0:  # if start isn't specified lets assume 0
        start = 0
    if end < 0:  # if end isn't specified assume the end of
 the video
        end = int(capture.get(cv2.CAP_PROP_FRAME_COUNT))

    capture.set(1, start)  # set the starting frame of the
capture
    frame = start  # keep track of which frame we are up to
, starting from start
    while_safety = 0  # a safety counter to ensure we don't
 enter an infinite while loop (hopefully we won't need it)
```

```
    saved_count = 0  # a count of how many frames we have s
aved

    while frame < end:  # lets loop through the frames unti
l the end

        _, image = capture.read()  # read an image from the
 capture

        if while_safety > 500:  # break the while if our sa
fety maxs out at 500
            break

        # sometimes OpenCV reads None's during a video, in
which case we want to just skip
        if image is None:  # if we get a bad return flag or
 the image we read is None, lets not save
            while_safety += 1  # add 1 to our while safety,
 since we skip before incrementing our frame variable
            continue  # skip

        if frame % every == 0:  # if this is a frame we wan
t to write out based on the 'every' argument
            while_safety = 0  # reset the safety count
            save_path = os.path.join(frames_dir, video_file
name, "{:010d}.jpg".format(frame))  # create the save path
            if not os.path.exists(save_path) or overwrite:
 # if it doesn't exist or we want to overwrite anyways
                image=cv2.resize(image,(256,256))
                image=cv2.cvtColor(image, cv2.COLOR_BGR2GRA
Y)
                cv2.imwrite(save_path, image)  # save the e
xtracted image
                saved_count += 1  # increment our counter b
y one

        frame += 1  # increment our frame count

    capture.release()  # after the while has finished close
 the capture

    return saved_count  # and return the count of the image
s we saved
```

```
def video_to_frames(video_path, frames_dir, overwrite=False
, every=1, chunk_size=1000):
    """
    Extracts the frames from a video using multiprocessing
    """

    video_path = os.path.normpath(video_path)  # make the p
aths OS (Windows) compatible
    frames_dir = os.path.normpath(frames_dir)  # make the p
aths OS (Windows) compatible

    video_dir, video_filename = os.path.split(video_path)
# get the video path and filename from the path

    # make directory to save frames, its a sub dir in the f
rames_dir with the video name
    os.makedirs(os.path.join(frames_dir, video_filename), e
xist_ok=True)

    print(video_filename)

    capture = cv2.VideoCapture(video_path)  # load the vide
o
    total = int(capture.get(cv2.CAP_PROP_FRAME_COUNT))  # g
et its total frame count
    capture.release()  # release the capture straight away

    if total < 1:  # if video has no frames, might be and o
pencv error
        print("Video has no frames. Check your OpenCV + ffm
peg installation")
        return None  # return None

    frame_chunks = [[i, i+chunk_size] for i in range(0, tot
al, chunk_size)]  # split the frames into chunk lists
    frame_chunks[-1][-1] = min(frame_chunks[-1][-1], total-
1)  # make sure last chunk has correct end frame, also hand
les case chunk_size < total

    prefix_str = "Extracting frames from {}".format(video_f
ilename)  # a prefix string to be printed in progress bar

    # execute across multiple cpu cores to speed up process
ing, get the count automatically
```

```
    with ProcessPoolExecutor(max_workers=multiprocessing.cp
u_count()) as executor:
        futures = [executor.submit(extract_frames, video_pa
th, frames_dir, overwrite, f[0], f[1], every)
                   for f in frame_chunks]  # submit the pro
cesses: extract_frames(...)

        for i, f in enumerate(as_completed(futures)):  # as
 each process completes
            print_progress(i, len(frame_chunks)-
1, prefix=prefix_str, suffix='Complete')  # print it's prog
ress

    return os.path.join(frames_dir, video_filename)  # when
 done return the directory containing the frames

def listdir_nohidden(AllVideos_Path):  # To ignore hidden f
iles
        file_dir_extension = os.path.join(AllVideos_Path, '
*.mp4')
        print(file_dir_extension)
        for f in glob.glob(file_dir_extension):
            if not f.startswith('.'):
                yield os.path.basename(f)
if __name__ == '__main__':
    # test it
    AllVideos_Path = videopath
    All_Videos=sorted(listdir_nohidden(AllVideos_Path))
    print(*All_Videos)
    All_Videos.sort()
    for iv in range(len(All_Videos)):
        VideofilePath = os.path.join(AllVideos_Path, All_Vi
deos[iv])
        video_to_frames(video_path=VideofilePath, frames_di
r=framespath, overwrite=True, every=5, chunk_size=1000)
```

The above code was used to extract frames from the videos. A video file is loaded from the dataset, then frames are extracted at 24 frames per second, and they are saved in a directory in the name of the video. It was applied in the data preparation process.

APPENDIX III:

DATA AUGMENTATION CODE

During the model development process, the sliding window technique was applied to link the frames and make sure more image sequences were generated bypassing the window in a sequence on the extracted frames.

```
Def get_clips_by_stride(stride, frames_list, sequence_size):
    """ For data augmenting purposes.
    Parameters
    ----------
    stride : int
        The desired distance between two consecutive frames
    frames_list : list
        A list of sorted frames of shape 256 X 256
    sequence_size: int
        The size of the desired LSTM sequence
    Returns
    -------
    list
        A list of clips , 10 frames each
    """
    clips = []
    sz = len(frames_list)
    clip = np.zeros(shape=(sequence_size, 256, 256, 1))
    cnt = 0
    for start in range(0, stride):
        for i in range(start, sz, stride):
            clip[cnt, :, :, 0] = frames_list[i]
            cnt = cnt + 1
            if cnt == sequence_size:
                clips.append(np.copy(clip))
                cnt = 0
    return clips
```

APPENDIX IV:

SELECTED STAE MODEL

```
def get_model(reload_model=True):
    """
    Parameters
    ----------
    reload_model : bool
        Load saved model or retrain it
    """
    if not reload_model:
        return load_model(Config.MODEL_PATH,custom_objects={'LayerNormalization': LayerNormalization})
    training_set = get_training_set()
    training_set = np.array(training_set)
    training_set = training_set.reshape(-1,10,256,256,1)
    seq = Sequential()
    seq.add(TimeDistributed(Conv2D(128, (11, 11), strides=4, padding="same"), batch_input_shape=(None, 10, 256, 256, 1)))
    seq.add(LayerNormalization())
    seq.add(TimeDistributed(Conv2D(64, (5, 5), strides=2, padding="same")))
    seq.add(LayerNormalization())
    # # # # #
    seq.add(ConvLSTM2D(64, (3, 3), padding="same", return_sequences=True))
    seq.add(LayerNormalization())
    seq.add(ConvLSTM2D(32, (3, 3), padding="same", return_sequences=True))
    seq.add(LayerNormalization())
    seq.add(ConvLSTM2D(64, (3, 3), padding="same", return_sequences=True))
    seq.add(LayerNormalization())
    # # # # #
    seq.add(TimeDistributed(Conv2DTranspose(64, (5, 5), strides=2, padding="same")))
    seq.add(LayerNormalization())
    seq.add(TimeDistributed(Conv2DTranspose(128, (11, 11), strides=4, padding="same")))
    seq.add(LayerNormalization())
    seq.add(TimeDistributed(Conv2D(1, (11, 11), activation="sigmoid", padding="same")))
    print(seq.summary())
    seq.compile(loss='mse', optimizer=keras.optimizers.Adam(learning_rate=1e-4, decay=1e-5, epsilon=1e-6))
    seq.fit(training_set, training_set,
            batch_size=Config.BATCH_SIZE, epochs=Config.EPOCHS, shuffle=False)
    seq.save(Config.MODEL_PATH)
    return seq
```

Figure above is the actual model that was selected for improvement. This the model before it was enhanced. It is important to note its structure before additional of more layers and addition of regularization functions.

APPENDIX V:

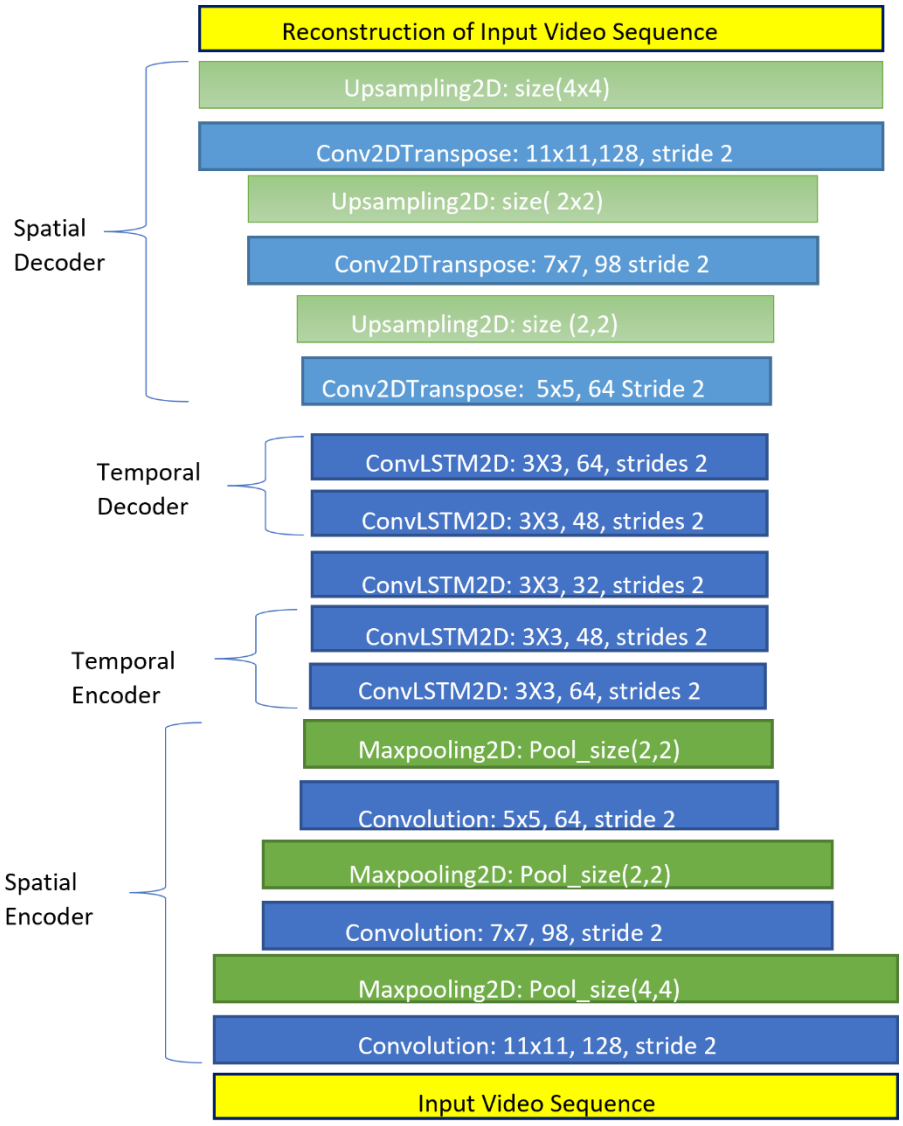
ENHANCED STAE MODEL

```
seq = Sequential()
seq.add(TimeDistributed(Conv2D(128, (11, 11), strides=2, padding="same",activation='relu'), batch_input_shape=(None, 10, 256, 256,1)))
seq.add(LayerNormalization())
seq.add(TimeDistributed(MaxPooling2D(pool_size=(4,4),padding="same")))
seq.add(TimeDistributed(Conv2D(98, (7, 7), strides=2, padding="same",activation='relu')))
seq.add(LayerNormalization())
seq.add(TimeDistributed(MaxPooling2D(pool_size=(2,2),padding="same")))
seq.add(TimeDistributed(Conv2D(64, (5, 5), strides=2, padding="same",activation='relu')))
seq.add(LayerNormalization())
seq.add(TimeDistributed(MaxPooling2D(pool_size=(2,2),padding="same")))
# # # # #
seq.add(ConvLSTM2D(64, (3, 3), padding="same", activation="relu", return_sequences=True))
seq.add(LayerNormalization())
seq.add(ConvLSTM2D(48, (3, 3), padding="same", activation="relu", return_sequences=True))
seq.add(LayerNormalization())
seq.add(ConvLSTM2D(32, (3, 3), padding="same",  activation="relu", return_sequences=True))
seq.add(LayerNormalization())
seq.add(ConvLSTM2D(48, (3, 3), padding="same", activation="relu", return_sequences=True))
seq.add(LayerNormalization())
seq.add(ConvLSTM2D(64, (3, 3), padding="same",  activation="relu", return_sequences=True))
seq.add(LayerNormalization())
# # # # #
seq.add(TimeDistributed(Conv2DTranspose(64, (5, 5), strides=2, padding="same",activation='relu')))
seq.add(LayerNormalization())
seq.add(TimeDistributed(UpSampling2D(size=(2,2))))
seq.add(TimeDistributed(Conv2DTranspose(98, (7, 7), strides=2, padding="same",activation='relu')))
seq.add(LayerNormalization())
seq.add(TimeDistributed(UpSampling2D(size=(2,2))))
seq.add(TimeDistributed(Conv2DTranspose(128, (11, 11), strides=2, padding="same",activation='relu')))
seq.add(LayerNormalization())
seq.add(TimeDistributed(UpSampling2D(size=(4,4))))
seq.add(TimeDistributed(Conv2D(1, (11, 11), activation="sigmoid", padding="same")))
print(seq.summary())
seq.compile(loss='mse', optimizer = Adam(learning_rate=1e-4, decay=1e-5, epsilon=1e-6), metrics=['accuracy'])
#seq.fit(training_set, training_set,batch_size=Config.BATCH_SIZE,epochs=Config.EPOCHS, shuffle=False,use_multiprocessing=True,callbacks=[CustomCallback()])
seq.fit(training_set, training_set,batch_size=Config.BATCH_SIZE,epochs=Config.EPOCHS, shuffle=False)
seq.save(Config.MODEL_PATH)
```

Figure above illustrates the enhanced model structure after an increase of the model depth and introduction of pooling functions in the spatial parts of the encoder and decoder. More layers can be noted from a comparison of the old model.

APPENDIX VI:

ILLUSTRATION OF THE ENHANCED STAE MODEL



The figure above is a graphical illustration of the enhanced model structure. It shows the size of different layers and transitioning between layers using pooling and un-pooling function. There were notable changes in the spatial and temporal parts of the autoencoder after improvement since their depth was increased.

APPENDIX VII:

PUBLICATION

The following publication was published from this work.

Munyua, J.G., Wambugu, G.M., & Njenga, S.T. (2021). A Survey of Deep Learning Solutions for Anomaly Detection in Surveillance Videos. International Journal of Computer and Information Technology (2279-0764).

https://www.ijcit.com/index.php/ijcit/article/view/166