Ministère de l'Enseignement
Supérieur et de la Recherche
Scientifique
Université de Carthage
Ecole Polytechnique de Tunisie

وزارة التعليم العالي و البحث العلمي
جامعة قرطاج
المدرسة التونسية للتقنيات

Engineering Internship Report

Anomaly detection in Videos

1^{st} July - 31^{st} August, 2021

Elaborated by:
Hassen MNEJJA
Third Year Student-SISY

Within:

UNIVERSITÉ Concordia UNIVERSITY

Supervised by: **Ms. Fatma NAJAR** **Research assistant**

Academic Year
2021 - 2022

Rue Elkhawarezmi BP 743 La Marsa 2078
Tel: 71 774 611 – 71 774 699 Fax: 71 748 843
Site Web: www.ept.rnu.tn

نهج الخوارزمي صب 743 المرسى 2078
الهاتف: 71 774 611 - 71 774 699 الفاكس: 71 748 843
موقع الواب: www.ept.rnu.tn

ABSTRACT

Video anomalies are generally defined as events or activities which are abnormal and signify abnormal behavior. The purpose of anomaly detection is to localize the anomaly events in video sequences over time or space.

In modern intelligent video surveillance systems, automatic anomaly detection through computer vision analytics plays an important role not only in greatly increase of surveillance efficiency, but also to reduce the burden on real-time surveillance.

In this project, we have implemented a model in which we integrate ConvNet and ConvLSTM with Auto-Encoder (**Spatiotemporal Auto-Encoder**) to learn the regularity of appearance and motion for the ordinary moments so to address challenges and limitations of anomaly detection and localization for real-time CCTV.

The accuracy and robustness of this approach were evaluated using three benchmark data sets.

Keywords : Anomaly Detection, Anomaly Localization, Long and Short Term Memory, Auto-Encoder, Convolution, Deep Learning, Computer Vision.

ACKNOWLEDGEMENTS

I would like to thank my esteemed supervisor **Mrs. Fatma Najjar**, research assistant at Concordia University, for her invaluable supervision, support and tutelage during the internship.

Additionally, I would like to express gratitude to **Dr. Nizar Bouguila**, Professor at Concordia university, for giving me the opportunity to pursue my internship within Concordia University.

I am very thankful also to all the academic staff at **Ecole Polytechnique de Tunisie** for their continous help and valubule training during my study years.

My appreciation also goes out to my family and friends for their encouragement and support all through my internship.

LIST OF ACRONYMS

AE	*Auto-Encoder*
ANN	*Artificial Neural Network*
AUC	*Area Under the Curve*
CCTV	*Closed-Circuit Television Camera Systems*
CNN	*Convolution Neural Network*
DL	*Deep Learning*
LSTM	*Long Short Term Memory*
ML	*Machine Learning*
NMT	*Neural Machine Translation*
PCA	*Principal Component Analysis*
RBM	*Restricted Boltzmann Machine*
RNN	*Recurrent Neural Network*
ROC	*Receiver Operator Characteristic*
SGD	*Stochastic Gradient Descent*

CONTENTS

LIST OF FIGURES

LIST OF TABLES

GENERAL INTRODUCTION

This work comes within the scope of the second year's Engineering Internship project at the Gina Cody School of Engineering and Computer Science at Concordia University. This work has been done under the supervision of Nizar Bouguila, professor at Concordia University, and Fatma Najar, research assistant at Concordia University.

In the first chapter, we will present the hosting company, the project description as well as the technologies and methodology that we have used during the internship. The second chapter contains the theoretical background of the different techniques as well as a brief description of the algorithms used in the next chapter. In the latter, we present the project's pipeline as well as the state-of-the-art algorithms and our algorithm and we will discuss the different results found. In the last chapter, we generate a general conclusion of the internship and some perspectives for future work.

CHAPTER 1

GENERAL FRAMEWORK OF THE PROJECT

Introduction

This chapter begins with a presentation the host internship laboratory. In addition, the issue that we are dealing with in this project, the projected objectives and the methodology followed are discussed. The contributions made are also listed. At the end of this chapter, the thesis plan is presented.

1.1 Host Organization Presentation

1.1.1 Concordia University

Concordia University is a comprehensive public research university in Montreal, Quebec, Canada. Concordia University, founded in 1974 as a merger of Loyola College and Sir George Williams University, is one of three universities in Quebec where English is the primary language of instruction.

Figure 1.1: Logo of the University of Concordia

1.1.2 Gina Cody School of Engineering and Computer Science

The Gina Cody School of Engineering and Computer Science is one of Canada's largest engineering faculties, with seven units:

- Building, Civil and Environmental Engineering (BCEE)

- Centre for Engineering in Society
- Computer Science and Software Engineering (CSSE)
- Concordia Institute for Information Systems Engineering (CIISE)
- Chemical and Materials Engineering (CME)
- Electrical and Computer Engineering (ECE)
- Mechanical, Industrial and Aerospace Engineering (MIAE)

In this internship, I worked within the Concordia Institute for Information Systems Engineering (CIISE) research team. As the name of the department indicates, its main activities are centered around cybersecurity, internet of things, artificial intelligence, and industry 4.0.

1.2 Project Description

1.2.1 Problem Statement

Video surveillance is a major consideration in the design, operation, and long-term viability of modern industrial and urban environments. It contributes to the efficiency, safety, security, and optimality of the locality, infrastructure, individuals, operations, and activities. Autonomous machinery, cyber-physical systems, and energy-efficient layouts are becoming more common in industrial settings. With the increased use of multilevel buildings and increased vehicular, pedestrian, and crowd movements, urban environments are becoming more densely populated.

Because of the vertical and horizontal expansion of asset and area utilization in both industrial and urban environments, the deployment of closed-circuit television camera systems (CCTV) has increased exponentially. Human observers, on the other hand, would be unrealistic and impossible to monitor and analyze every video stream with high precision. The current literature on artificial intelligence (AI) techniques for autonomous video surveillance is divided into four categories: video summarization, object detection and re-identification, activity/behavior detection, and anomaly detection.

1.2.2 Anomalies Detection in Videos

Anomaly detection is a critical task in autonomous video surveillance because it contributes to the success of other categories like video summarization, object detection, and

re-identification. It is also a difficult task because the anomalies to be detected are unknown ahead of time, making it difficult even for a human observer. A general definition of anomaly detection is the identification of behaviors that differ from expected and accepted behavior.

1.3 State-of-the-Art Algorithms

1.3.1 Incremental Spatiotemporal Learner (ISTL)

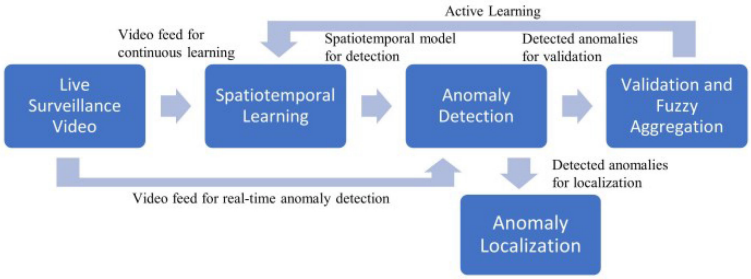


Figure 1.2: Overview of the proposed ISTL approach [1]

This algorithm is developed by Nawaratne Rashmika et al. [1] "Spatiotemporal Anomaly Detection Using Deep Learning for Real-Time Video Surveillance".

They propose the incremental spatiotemporal learner (ISTL) in this work to solve the constraints and limitations of anomaly detection and localisation in real-time video surveillance. ISTL is an unsupervised deep-learning strategy that uses active learning with fuzzy aggregation to continuously update and differentiate between new anomalies and normality that grow over time. ISTL is demonstrated and tested on three benchmark datasets for accuracy, robustness, computing overhead, and contextual indicators.

1.3.2 Deep Convolution Feed-forward Autoencoder (Conv-AE)

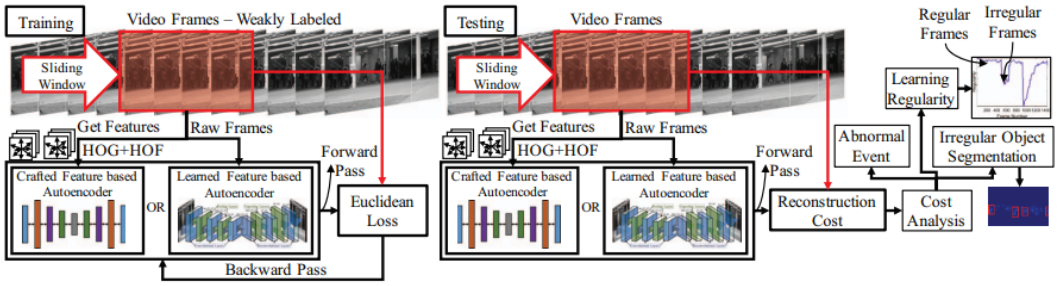


Figure 1.3: Overview of the proposed Conv-AE approach [2]

This algorithm is developed by Mahmudul Hasan et al. [2] "Learning Temporal Regularity in Video Sequences".

They solved this problem in the study by learning a generative model for regular motion patterns (referred as regularity) from many sources with very little supervision. They specifically propose two strategies based on autoencoders for their ability to work with little to no supervision. They start with traditional handcrafted spatiotemporal local characteristics and train a fully connected autoencoder on them. Second, as an end-to-end learning system, they construct a fully convolutional feed-forward autoencoder to train both the local features and the classifiers. Their model is capable of capturing regularities from diverse datasets.

1.3.3 Streaming Restricted Boltzmann Machine (S-RBM)

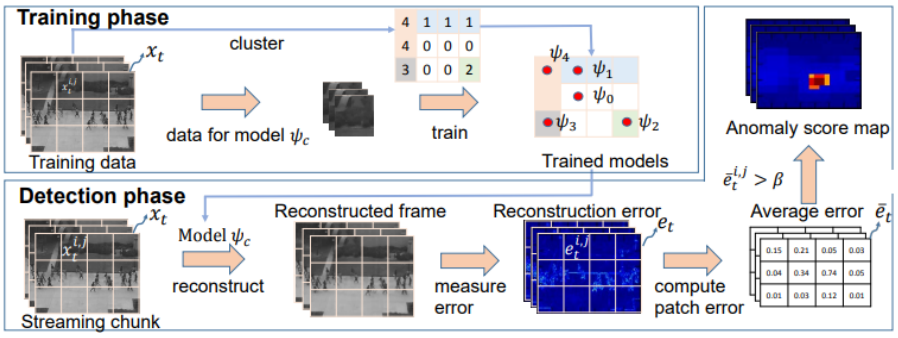


Figure 1.4: Overview of the proposed S-RBM approach [3]

This algorithm is developed by Hung Vu et al. [3] "Deep Abnormality Detection in Video Data".

The restricted Boltzmann machine (RBM) is an undirected generating network with two layers: visible and hidden. In comparison to deep generative networks, the RBM can learn the data distribution more effectively because of its bipartite structure. RBMs are also extended to handle more complex data structures like multiway tensor data. Furthermore, their pretraining responsibilities contribute to the effectiveness of deep network training and signal the growth of deep learning. As a result, they use RBMs and generative networks to detect video anomalies.

1.3.4 Unmasking-late-fusion (Unmasking)

This algorithm is developed by Radu Tudor Ionescu et al. [4] "Unmasking the abnormal events in video".

They present a system for detecting abnormal events in video that does not require any training sequences. This approach is built on unmasking, a technique previously employed for authorship verification in text files that they have adapted for their objective. They iteratively train a binary classifier to discriminate between two successive video sequences, deleting the most discriminant features at each step. Higher training accuracy rates for intermediately acquired classifiers indicate abnormal events.

1.3.5 Deep Learning

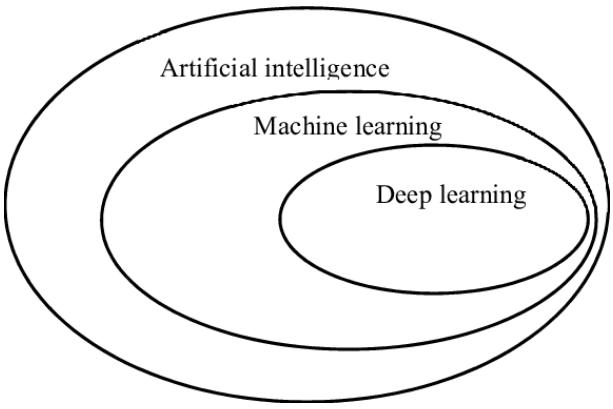


Figure 1.5: A Venn-diagram of artificial intelligence

Deep Learning is a class of algorithms whose fundamentals have been known since the late 1980s, but whose application has just recently gained traction.

Its concept is straightforward: the model is made up of a succession of modules, each representing a processing step. Each module is trainable, with configurable parameters

similar to linear classifier weights. The system is driven from beginning to end: in each instance, all of the parameters of all of the modules are updated to bring the system's output closer to the desired result. The term "deep" refers to the way these modules are arranged in consecutive levels.

To train the system in this way, it is required to know which parameters of each module should be adjusted in which direction and by how much. To do so, we must compute a gradient, which is the amount by which the output error will rise or decrease when the parameter is changed by a particular amount for each adjustable parameter. The back-propagation method, which has been employed since the mid-1980s, is utilized to calculate this gradient.

In its most basic form, a deep architecture can be thought of as a multilayer network of basic elements (known as neurons) associated with trainable weights, similar to linear classifiers. This is referred to as a multilayer neural network.

1.3.6 Example of Neural Networks

Artificial Neural Network

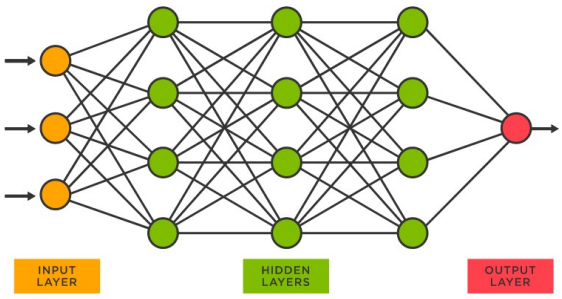


Figure 1.6: General structure of Artificial Neural Network

Artificial Neural Networks (ANNs) are a family of machine learning algorithms inspired by the structure and function of the brain that learn from data and specialize in pattern identification. In fact, it was created in 1943 by a neurophysiologist named Warren McCulloch and a mathematician named Walter Pitts in a paper titled "A logical calculus of the ideas immanent in nervous activity" [5]. Many improvements have been made since then to achieve our present performance of deep learning models.

Artificial neural networks are composed of a network of linked neurons (or Convolutional kernels and many more...) that are structured into different layers that, in most circumstances, receive as input the output of the preceding layer. The above image depicts

a common type of ANN known as a Multi-Layer Perceptron (MLP), which is made up of multiple neurons.

Convolutional Neural Network

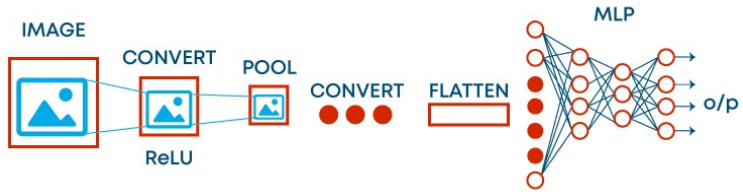


Figure 1.7: General structure of Convolution Neural Network

A convolutional neural network (CNN, or ConvNet) is a type of deep neural network that is most commonly used to analyze visual imagery. It is made up of convolution layers that use filters, also known as kernels, to perform convolution operations by scanning the input image with respect to its dimension. Kernel convolution is employed not just in CNNs, but also in many other Computer Vision techniques. It is a technique in which we take a tiny matrix of numbers (called a kernel or filter), pass it over our picture, and then transform it based on the filter's values.

Recurrent Neural Network

A recurrent neural network (RNN) is a type of artificial neural network in which node connections form a directed graph along a time sequence. As a result, it can display temporal dynamic behavior. RNNs, which are derived from feedforward neural networks, can handle variable length sequences of inputs using their internal state (memory). As a result, they may be used for tasks like unsegmented, linked handwriting recognition or speech recognition. Recurrent neural networks are Turing complete in theory and may execute arbitrary algorithms to handle arbitrary sequences of inputs.

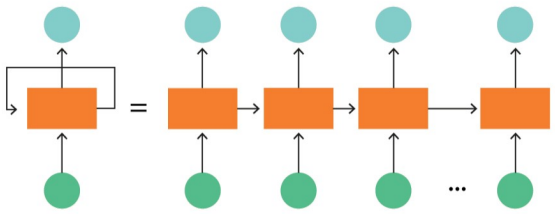


Figure 1.8: General structure of Reccurent Neural Network

The graphic below shows an unrolled RNN. The RNN is unrolled following the equal sign on the left. Because the different time steps are visible and information is transmitted from one time step to the next, there is no cycle after the equal sign. This illustration also demonstrates why an RNN may be thought of as a series of neural networks.

1.4 Technologies Used

1.4.1 Python

Python is a high-level general-purpose programming language that is interpreted. With the usage of considerable indentation, its design philosophy prioritizes code readability. Its language elements and object-oriented approach are intended to assist programmers in writing clear, logical code for small and large-scale projects.

1.4.2 Libraries Used

Tensorflow

Google designed and distributed TensorFlow, a Python library for fast numerical computing. It is a foundation library that can be used to directly develop Deep Learning models or by using wrapper libraries built on top of TensorFlow to simplify the process.

Keras

Keras is a Python-based deep learning API that runs on top of the machine learning platform TensorFlow. It was created with the goal of allowing for quick experimentation. It is critical to be able to move from idea to result as quickly as feasible when conducting research.

OpenCV

OpenCV (Open Source Computer Vision Library) is a free and open source software library for computer vision and machine learning. OpenCV was created to provide a common foundation for computer vision applications and to speed up the incorporation of machine perception into commercial goods. Because OpenCV is a BSD-licensed product, it is simple for businesses to use and change the code.

Matplotlib

Matplotlib is a Python package that allows you to create static, animated, and interactive visualizations. Matplotlib generates publication-quality figures in a range of hardcopy and interactive formats across multiple platforms. Matplotlib is a graphical user interface toolkit that may be used in Python scripts, the Python and IPython shells, web application servers, and a variety of graphical user interface toolkits.

1.4.3 Google Colab

Colaboratory, or "Colab" for short, is a Google Research product. Colab enables anyone to create and execute arbitrary Python code via the internet, and it is particularly well suited to machine learning, data analysis, and education. Colab is a hosted Jupyter notebook service that requires no setup and provides free access to computer resources such as GPUs. It meets the following requirements:

- CPU : 2 Xeon cores running at 2.2GHz
- RAM : 13GB of RAM
- Storage : 33GB HDD for temporary storage
- GPU : NVIDIA Tesla K80 GPU (4992 CUDA cores, 8,73 Tflops)

1.4.4 Google Drive

Google Drive is a cloud-based file storage and syncing service created by Google. Google Drive, which was launched on April 24, 2012, allows users to store files in the cloud (on Google's servers), synchronize files across devices, and share files.

1.5 Methodology

During my internship, I used a multi-step process. First, I began by selecting datasets. Second, I prepped my data for use by employing various preprocessing procedures. Following that, I conducted a search for state-of-the-art algorithms that solve the same problem. So I started building and training my model after that. Finally, I compared my results to the results of the other methods.

Conclusion

In this first chapter, we introduced the host company, the project that we worked on during this internship and both its problem statement and its goals. Later we also explained the basic technical concepts, the software used and the adapted methodology.

In the next chapter, we will talk about the technical and theoretical background and the main concepts of this project.

CHAPTER 2

THEORETICAL BACKGROUND

Introduction

In this chapter, we will present the Mathematical Background behind the models used for Deep Learning. Therefore, we will address some types of layer such as **Transpose Convolution layer**, **Batch Normalization layer** and **LSTM layer**, and the type of Artificial Neural Network used which is **Auto-Encoder**. In addition, I will explain the metric used to evaluate my model which is **AUC-ROC**.

2.1 Transpose Convolution

Transposed Convolutions (also known as deconvolution) are used to upsample an input feature map to a desired output feature map by employing learnable parameters. The basic procedure of a transposed convolution is described below:

- Step 1: Consider a 2x2 encoded feature map that needs to be upsampled to a 3x3 feature map.
- Step 2: We take a kernel of size 2x2 with unit stride and no padding.
- Step 3: Now we multiply the upper left element of the input feature map by each member of the kernel.
- Step 4: We repeat this process for all of the remaining items in the input feature map.
- Step 5: Some of the components in the resultant upsampled feature maps overlap, as seen. We just add the items of the overlapping places to fix this problem.
- Step 6: The final upsampled feature map with the requisite spatial dimensions of 3x3 will be the output.

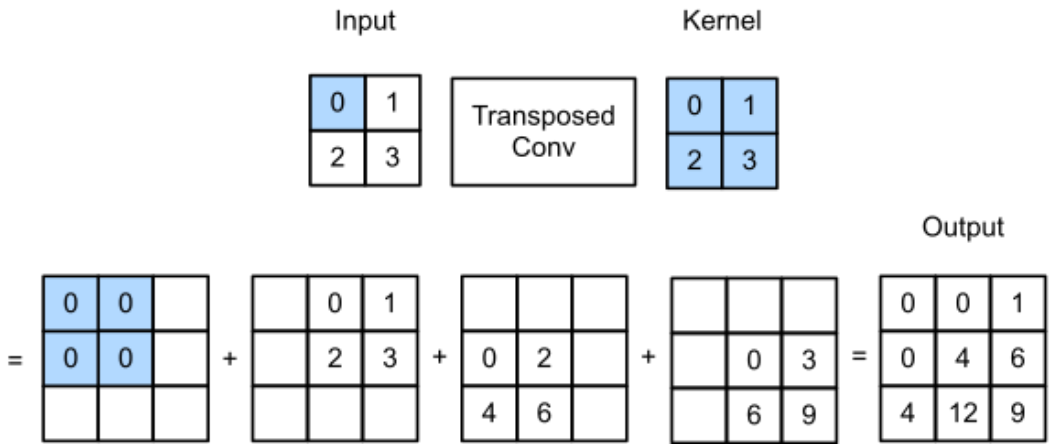


Figure 2.1: Different steps of transposed convolution [6]

For a given size of the input (i), kernel (k), padding (p), and stride (s), this table below represent the main difference between standard and transposed convolution :

Conv Type	**Operation**	**Output Size**
Standard	Downsampling	$\lfloor (i - k + 2 * p)/s \rfloor + 1$
Transposed	Upsampling	$(i - 1) * s + k - 2 * p$

Tableau 2.1: Comparison between standard and transposed convolution

2.2 Area Under Curve

2.2.1 Introduction

Confusion matrix : A confusion matrix is a summary of classification problem prediction outcomes. The number of right and wrong predictions is summarized with count values and divided by class. The confusion matrix looks like this:

		True Class	
		Positive	**Negative**
Predicted Class	**Positive**	TP	FP
	Negative	FN	TN

Tableau 2.2: Confusion matrix

Sensitivity : It informs us how many members in the positive class were properly

identified.

$$Sensitivity = \frac{TP}{TP + FN} \tag{2.1}$$

False Negative Rate (FNR) : It informs us how much of the positive class was misclassified by the classifier.

$$FNR = \frac{FN}{TP + FN} = 1 - Sensitivity \tag{2.2}$$

Specificity : It informs us how many members in the negative class were properly identified.

$$Specificity = \frac{TN}{TN + FP} \tag{2.3}$$

False Positive Rate (FPR) : It indicates how much of the negative class was wrongly categorized by the classifier.

$$FPR = \frac{FP}{TN + FP} = 1 - Specificity \tag{2.4}$$

2.2.2 Definition

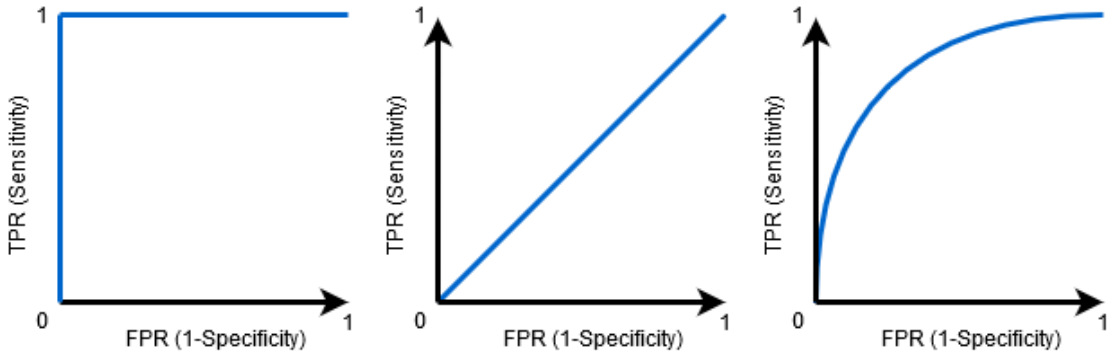
The Receiver Operator Characteristic (ROC) curve is an evaluation metric for binary classification problems. It is a probability curve that shows the TPR vs the FPR at various threshold levels, separating the 'signal' from the 'noise.' The Area Under the Curve (AUC) is a measure of a classifier's ability to discriminate between classes and is used to summarize the ROC curve.

AUC is desirable for the following two reasons:

- AUC is **scale-invariant**. It evaluates how well predictions are ranked, rather than their absolute values.
- AUC is **classification-threshold-invariant**. It assesses the quality of the model's predictions irrespective of what classification threshold is chosen.

2.2.3 Performance evaluation

The greater the AUC, the better the model's ability in differentiating between positive and negative classes.



(a) AUC = 1 (b) 0.5 < AUC < 1 (c) AUC = 0.5

Figure 2.2: Different AUC-ROC cases [7]

- When AUC = 1, the classifier is capable of successfully distinguishing between all Positive and Negative class points. If, on the other hand, the AUC was 0, the classifier would forecast all Negatives as Positives and all Positives as Negatives.

- When 0.5<AUC<1, there is a good possibility that the classifier will be able to discriminate between positive and negative class values. This is due to the classifier's ability to recognize more True positives and True negatives than False negatives and False positives.

- When AUC = 0.5, the classifier is unable to differentiate between Positive and Negative class points. That is, the classifier predicts either a random class or a fixed class for all observations.

2.3 Layer Normalization

2.3.1 Definition

Layer normalization (LayerNorm) is a technique to normalize the distributions of intermediate layers. It allows for smoother gradients, quicker training, and higher generalization accuracy.

2.3.2 Technique Description

In General Case

The layer normalization statistics are computed as follows for all hidden units in the same layer:

$$\begin{aligned} \mu_i &= \frac{1}{K}\sum_{k=1}^{K} x_{i,k} \\ \sigma_i^2 &= \frac{1}{K}\sum_{k=1}^{K} (x_{i,k} - \mu_i)^2 \\ \hat{x}_{i,k} &= \frac{x_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \end{aligned} \tag{2.5}$$

where H is the number of hidden units in a layer. Layer normalization uses the same normalizing terms μ and σ for all hidden units in a layer, but different training instances use different normalization terms. Layer normalization, unlike batch normalization, has no restrictions on the size of the mini-batch and may be employed in the pure online regime with batch size 1.

Finally, there is a scaling and shifting step. γ and β are learnable parameters.

$$y_i = \gamma \hat{x}_i + \beta \equiv \mathrm{LN}_{\gamma,\beta}(x_i) \tag{2.6}$$

Layer Normalization for Convolutional Neural Network

If layer normalization is applied to the outputs of a convolution layer, the math must be significantly adjusted since it is not logical to combine all the items from separate channels together and compute the mean and variance. Each channel is treated as a "independent" sample, with all normalization performed just for that channel within the sample.

Assume the input tensor has shape $[m, H, W, C]$, for each channel $c \in \{1, 2, ..., C\}$

$$\begin{aligned} \mu_{i,c} &= \frac{1}{HW}\sum_{j=1}^{H}\sum_{k=1}^{W} x_{i,j,k,c} \\ \sigma_{i,c}^2 &= \frac{1}{HW}\sum_{j=1}^{H}\sum_{k=1}^{W}\left(x_{i,j,k,c}-\mu_{i,c}\right)^2 \\ \hat{x}_{i,j,k,c} &= \frac{x_{i,j,k,c}-\mu_{i,c}}{\sqrt{\sigma_{i,c}^2+\epsilon}} \end{aligned} \tag{2.7}$$

Specifically for each channel, we have learnable parameters γ_c and β_c , such that :

$$y_{i,r,c} = \gamma_c \hat{x}_{i,:,c} + \beta_c \equiv \mathrm{LN}_{\gamma_c,\beta_c}\left(x_{i,r,c}\right) \tag{2.8}$$

2.3.3 Layer Normalization VS Batch Normalization

Unlike batch normalization, Layer Normalization calculates the normalization statistics directly from the summed inputs to the neurons inside a hidden layer, resulting in no additional dependencies between training instances. It is effective for RNNs and increases both training time and generalization performance of numerous current RNN models. It has lately been employed with Transformer models.

2.4 Long Short Term Memory Networks

2.4.1 Definition

Long Short-Term Memory (LSTM) networks are a sort of **Recurrent Neural Network (RNN)** that are able of learning order dependence in sequence prediction tasks. Unlike traditional feed-forward neural networks, LSTM has feedback connections. This is a necessary behavior in complicated problem areas like as machine translation, speech recognition, and others.

2.4.2 Short-term Memory Problem

Short-term memory is a problem for Recurrent Neural Networks. They will have difficulty transporting information from earlier time steps to later ones if the sequence is lengthy enough.

The vanishing gradient problem affects recurrent neural networks during back propagation. Gradients are values that are used to update the weights of a neural network. The vanishing gradient problem occurs when the gradient declines as it propagates backward in time. When a gradient value becomes incredibly tiny, it does not contribute much to learning.

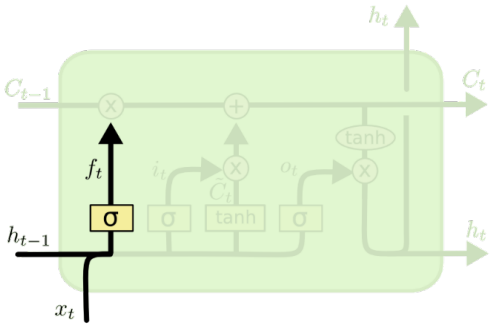
2.4.3 Core Concept

The cell state and other different gates are the key ideas of LSTM. The cell state serves as a conduit for relative information to be sent all the way down the sequence chain. You might think of it as the network's "memory." In principle, the cell state can carry meaningful information throughout the sequence's processing. As a result, information from earlier time steps can traverse to later time steps, minimizing the impact of short-term memory. As the cell state advances, information is added or deleted from the cell state via gates. The gates are several neural networks that determine whether information on the cell state is permitted. During training, the gates might learn which information is important to remember and which to discard.

2.4.4 LSTM's Gates

Forget Gate

First, there's the forget gate. This gate determines whether information should be discarded or saved. The sigmoid function is used to process information from the prior hidden state as well as information from the current input. The values range from 0 to 1. The closer to 0 means to forget, and the closer to 1 means to keep.

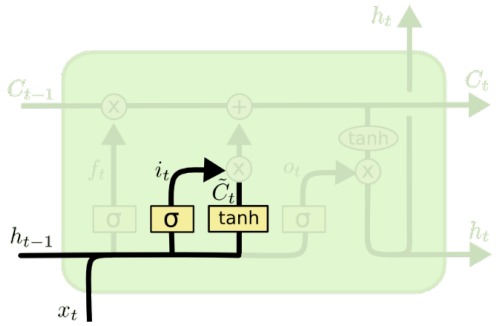


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 2.3: Forget gate operations [8]

Input Gate

The input gate is used to update the cell state. First, we use a sigmoid function to combine the prior hidden state and the current input. This determines which values are changed by converting them to be between 0 and 1. A value of 0 indicates that it is unimportant, whereas a value of 1 indicates that it is very significant. To regulate the network, you also pass the hidden state and current input into the tanh function, which squishes values between -1 and 1. The sigmoid result is then multiplied by the tanh output. The sigmoid output will determine which information from the tanh output should be kept.

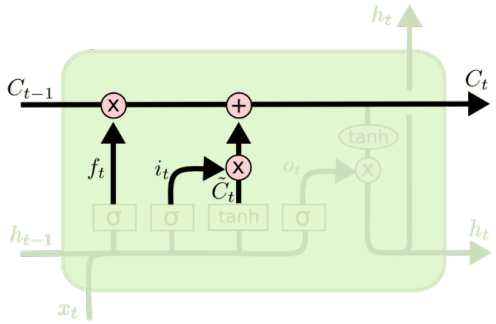


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 2.4: Input gate operations [8]

2.4.5 Cell State

We should now have enough information to compute the cell state. First, the cell state gets pointwise multiplied by the forget vector. If multiplied by values close to zero, this has the potential to drop values in the cell state. The output of the input gate is then used to perform a pointwise addition, which changes the cell state to new values that the neural network considers important. This results in our new cell state.

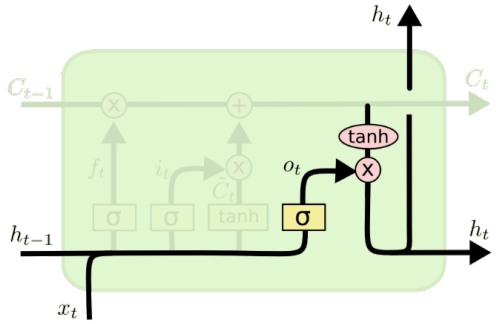


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2.5: Calculating cell state [8]

Output Gate

Finally, we have the output gate. The output gate determines the next hidden state. Keep in mind that the hidden state comprises information from prior inputs. Predictions are also made using the hidden state. First, we use a sigmoid function to combine the prior hidden state and the current input. The newly changed cell state is then sent to the tanh function. To determine what information the hidden state should contain, we multiply the tanh output by the sigmoid output. The hidden state is the result. The changed cell state and hidden state are then passed forward to the next time step.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2.6: Output gate operations [8]

2.4.6 ConvLSTM

ConvLSTM is a type of recurrent neural network for spatio-temporal prediction that has convolutional structures in both the input-to-state and state-to-state transitions. The ConvLSTM determines the future state of a certain cell in the grid by the inputs and past states of its local neighbors. This can easily be achieved by using a convolution operator in the state-to-state and input-to-state transitions (see Figure). The key equations of ConvLSTM are shown below, where '$*$' denotes the convolution operator and '$\odot$' the Hadamard product:

$$\begin{aligned}
i_t &= \sigma\left(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \odot \mathcal{C}_{t-1} + b_i\right) \\
f_t &= \sigma\left(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \odot \mathcal{C}_{t-1} + b_f\right) \\
\mathcal{C}_t &= f_t \odot \mathcal{C}_{t-1} + i_t \odot \tanh\left(W_{xc} * X_t + W_{hc} * \mathcal{H}_{t-1} + b_c\right) \\
o_t &= \sigma\left(W_{xo} * X_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \odot \mathcal{C}_t + b_o\right) \\
\mathcal{H}_t &= o_t \odot \tanh\left(C_t\right)
\end{aligned} \tag{2.9}$$

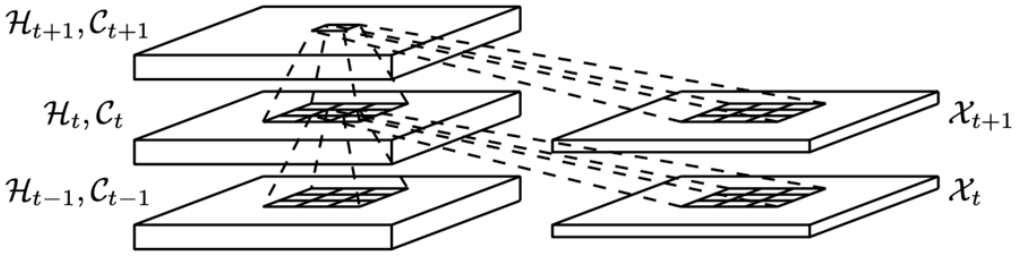


Figure 2.7: Inner structure of ConvLSTM [9]

2.5 Auto-Encoder

2.5.1 Definition

Auto-Encoder is an **unsupervised** learning technique. It is a feed-forward neural network that has been trained to duplicate its input to its output. In reality, it compresses the input into a lower-dimensional *code* representation and then reconstructs the output from this representation.

2.5.2 Architecture

An auto-encoder consists of three components:

- Encoder: An encoder is a fully connected, feed-forward neural network that compresses the input picture into a latent space representation and encodes it as a compressed representation in a reduced dimension. The compressed image is the distorted version of the original image.
- Code: This layer of the network holds the decoder's reduced representation of the input.
- Decoder: The decoder, like the encoder, is a feed-forward network with a similar topology. This network is in charge of restoring the input from the code to its original dimensions.

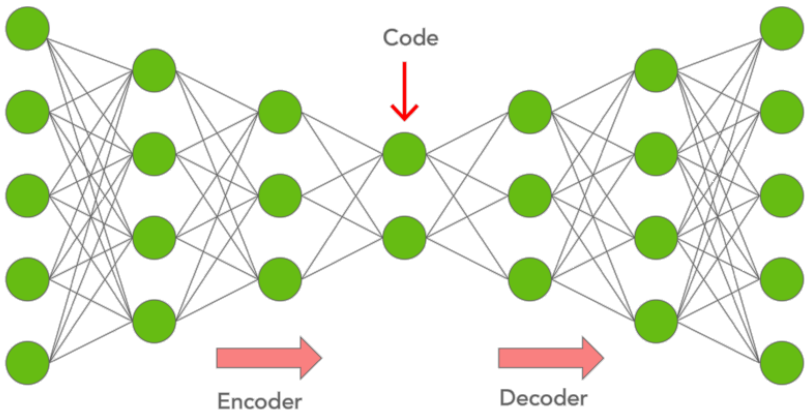


Figure 2.8: Common structure of Auto-Encoders [10]

The input is compressed and stored in the layer called Code after passing through the encoder, and then the decoder decompresses the original input from the code. The main goal of the auto-encoder is to provide an output that is similar to the input.

It should be noted that the decoder structure is the mirror image of the encoder. This is not a must, although it is often the case. The only criterion is that the input and output dimensions be the same.

2.5.3 Use Cases

Dimensionality reduction

For Hinton's 2006 study [11], he pretrained a multi-layer auto-encoder using a stack of RBMs and then used their weights to build a deep auto-encoder with successively decreasing hidden layers until he reached a bottleneck of 30 neurons. When compared to the first 30 components of a principal component analysis (PCA), the resultant 30 dimensions of the code generated a reduced reconstruction error and learnt a representation that was qualitatively simpler to interpret, clearly dividing data clusters. data clusters.

Information retrieval

Information retrieval benefits in particular from the dimensional reduction, since the search in certain types of low-dimensional spaces can become more powerful. Auto-encoders were actually used for semantic hashing, suggested by Hinton and Salakhutdinov in 2007 [12]. When training the algorithm to generate low-dimensional binary code, all the entries

in the database can be saved in a hash table that maps binary code vectors to entries. This table then makes it easier to retrieve the information by returning all entries with the same binary code as the query, or slightly less similar entries by flipping some of the encoding bits of request.

Machine translation

Auto-encoders have been applied to machine translation, commonly known as Neural Machine Translation (NMT). Unlike traditional auto-encoders, the output doesn't match the input - it's in different language. In Natural Machine Translation, texts are handled as sequences that have to be encoded during the learning process, while sequences in the target language(s) are generated on the decoder side. Language-specific auto-encoders integrate further linguistic features into the learning process, such as the decomposition features of Chinese.

Anomaly detection

Another application for auto-encoders is anomaly detection. By learning to reproduce the most salient features in the training data according to some of the constraints described previously, the model is encouraged to learn to accurately reproduce the most commonly observed features. In the case of anomaly, the model should degrade its reconstruction performance. In most cases, only normal instance data are used to train the auto-encoder; in others, the frequency of anomalies is low relative to the set of observations, so their contribution to the learned representation can be neglected. After training, the auto-encoder will accurately reconstruct "normal" data, while failing to do so with unfamiliar anomalous data. The reconstruction error (the error between the original data and its low dimensional reconstruction) is used as an anomaly score to detect anomalies.

Conclusion

During this chapter, we have seen how the models that we will be using work and the mathematics behind them.

In the next chapter, We will tackle the realisation and evaluation of my project and a comparison of our algorithm with the state-of-the-art algorithms.

CHAPTER 3

REALISATION AND EVALUATION

Introduction

3.1 Project's Pipeline

The following diagram summarizes the pipeline of my project :

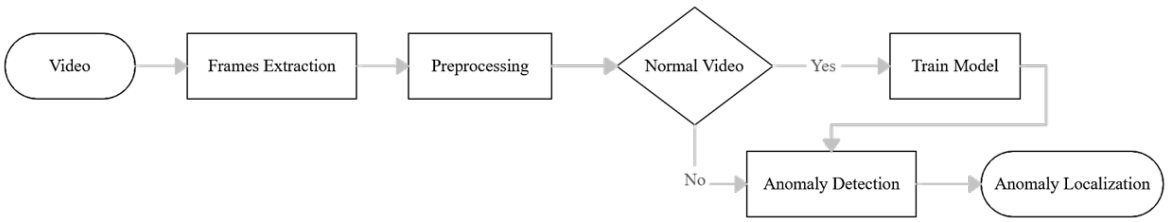


Figure 3.1: Project's Pipeline

3.2 Datasets

We have chosen 4 benchmark dataset in the problem of anomalies detection in videos. The problem with the ShanghaiTech datasets is that this dataset is not labeled so we can't compare the performance of our model with other algorithms.

Dataset	**#Frames**			**#Abnormal Events**	**#Scenes**
	Total	**Training**	**Testing**		
CUHK Avenue	30,652	15,328	15,324	47	1
UCSD Ped1	14,000	6,800	7,200	40	1
UCSD Ped2	4,560	2,550	2,010	12	1
ShanghaiTech	317,398	274,515	42,883	130	13

Tableau 3.1: Comparison of different datasets used

The training videos contain only normal videos and the testing ones contains anomalies in some frames.

3.2.1 The CUHK Avenue dataset

The CUHK Avenue dataset was created by filming street activities at the Chinese University of Hong Kong with a stationary camera with a resolution of 640 X 360 pixels. This dataset contains 16 train video examples of typical human behavior and 21 test video samples of unusual occurrences and human activities. Normal behaviors include pedestrians on the sidewalk and groups of pedestrians congregating on the sidewalk, but abnormal events include persons littering/discarding stuff, loitering, going toward the camera, walking on the grass, and abandoning objects.

Figure 3.2: Various example images of the CUHK Avenue dataset

3.2.2 The ShanghaiTech dataset

The ShanghaiTech Campus dataset is widely regarded as one of the most comprehensive and realistic datasets available for video anomaly identification. It includes footage from 13 various cameras located throughout the ShanghaiTech University campus, as well as a wide range of anomaly types. It contains about 270, 000 training frames and 130 anomalous events. The majority of the anomaly events in the ShanghaiTech dataset are human-related.

Figure 3.3: Various example images of the ShanghaiTech dataset

3.2.3 The UCSD pedestrian Dataset

A stationary video camera with a resolution of 238X158 pixels was used to acquire the UCSD pedestrian dataset, which focused on two pedestrian pathways. This set includes two datasets, Ped 1 and Ped 2, which capture various crowd scenarios ranging from sparse to crowded. The train video samples only contain scenarios of pedestrians walking on the pathway, whereas the test video samples contain anomalous pedestrian movement patterns such as walking across the sidewalk or on the grass, as well as unexpected behavior such as skateboarding, cycling, and vehicular movement. The Ped 1 dataset includes 34 train and 36 test video samples, while the Ped 2 dataset contains 16 train and 12 test video samples. Both datasets were recorded at a frame rate of 26 frames per second (FPS).

Figure 3.4: Various example images of the UCSD pedestrian dataset (ped1)

Figure 3.5: Various example images of the UCSD pedestrian dataset (ped2)

3.3 Data Preprocessing

Frame extraction : First of all, we have extracted frames from videos according to the fps (frames per second) of the video. Per example, for both the CUHK Avenue dataset and the UCSD pedestrian dataset, videos were captured at a frame rate of 26 FPS (frames per second) and for the ShanghaiTech dataset, it was caotured at 25 fps.

Resize and normalization: As the video samples have different dimensionality, we pre-process the inputs by resizing the extracted frames to 224×224 pixels, and normalizing pixel values by scaling between 0 and 1.

Clips by stride : Finally, we have selected the depth of temporal cuboid, T = 8 representing an approximate duration of one-third of a second (for 26 fps videos). In this step, we have used the sliding window technique

3.4 Our Proposed Algorithm

3.4.1 Introduction

The algorithm attempts to solve the problem following a five-step process:

- **Step 1:** We pass the video through a spatiotemporal auto-encoder which aims to reconstruct the input.
- **Step 2:** We calculate the reconstruction cost which describes how much far the reconstructed video is different from the input video.
- **Step 3:** We analyze the reconstruction error using two thresholds which are temporal and spatial.
- **Step 4:** Using the temporal threshold, we can detect the period in which there is an anomaly.
- **Step 5:** In this period and using the spatial threshold (anomaly threshold), we can detect the localization of the anomaly in the spatial reference.

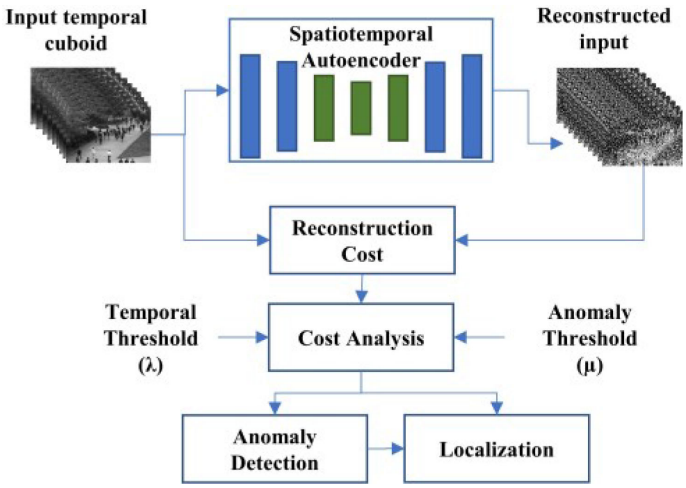


Figure 3.6: Our algorithm

3.4.2 Employed Architecture

The following table present the employed architecture :

Type of layer	Output Shape	Weights (N)	Weights (%)
Input	(8, 224, 224, 1)		
TimeDistributed	(8, 56, 56, 128)	93440	1.1%
LayerNormalization	(8, 56, 56, 128)	256	0.0%
TimeDistributed	(8, 28, 28, 64)	1384512	15.6%
LayerNormalization	(8, 28, 28, 64)	128	0.0%
ConvLSTM2D	(8, 28, 28, 64)	295168	3.3%
LayerNormalization	(8, 28, 28, 64)	128	0.0%
ConvLSTM2D	(8, 28, 28, 32)	110720	1.2%
LayerNormalization	(8, 28, 28, 32)	64	0.0%
ConvLSTM2D	(8, 28, 28, 64)	221440	2.5%
LayerNormalization	(8, 28, 28, 64)	64	0.0%
TimeDistributed	(8, 56, 56, 64)	692288	7.8%
LayerNormalization	(8, 56, 56, 64)	128	0.0%
TimeDistributed	(8, 224, 224, 128)	5972096	67.4%
LayerNormalization	(8, 224, 224, 128)	256	0.0%
TimeDistributed	(8, 224, 224, 1)	93313	1.1%

Tableau 3.2: Architecture of the model

The model is based on the spatiotemporal auto-encoder architecture. It is composed of five parts :

- Spatial encoder : two convolution layers
- Temporal encoder : one convolutional LSTM layer
- Bottleneck : one convolutional LSTM layer
- Temporal decoder : one convolutional LSTM layer
- Spatial decoder : three transposed convolution layers

After each layer, we use a normalization layer to normalize the activities of the neurons with the aim to reduce the training time. In fact, with unnormalized data, numerical ranges of features may vary strongly. The figure above explains this attitude :

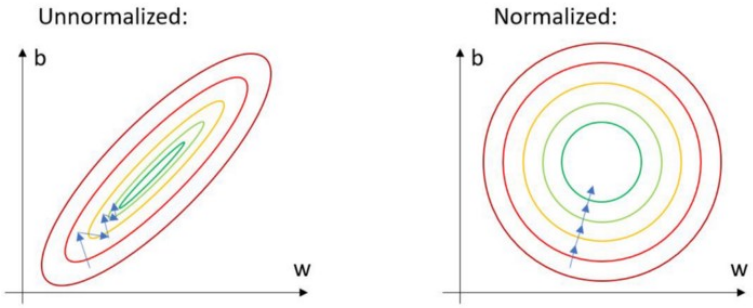


Figure 3.7: Normalized data allows faster convergence [13]

We trained the spatiotemporal model using Stochastic gradient descent (SGD) algorithm with a learning rate of 0.01.

3.4.3 Used Metrics

Training Metric

To train the spatiotemporal auto-encoder the euclidean loss between the input cuboid and the reconstructed one. The formula that calculates that loss is given as follows :

$$\begin{aligned} \varphi(i,j,k) &= \left|X_{(i,j,k)} - \bar{X}_{(i,j,k)}\right|^2 \\ E &= \left(\sum_{i=0}^{T}\sum_{j=0}^{w}\sum_{k=0}^{h}\varphi(i,j,k)\right)^{\frac{1}{2}} \end{aligned} \tag{3.1}$$

Regularity Score

For detecting anomalies, we compute a score that tells us how much this frame is regular (normal). This score is named the regularity score. This score is calculated as follows:

- We compute the reconstruction error of a pixel's intensity value I at the location (x,y) in frame t of the video using L2 norm:

$$e(x,y,t) = \|I(x,y,t) - f_W(I(x,y,t))\|_2 \tag{3.2}$$

- Where Fw is the learned model by the LSTM convolutional autoencoder. Then we compute the reconstruction error of a frame t by summing up all the pixel-wise errors:

$$e(t) = \sum_{(x,y)} e(x,y,t) \tag{3.3}$$

- The reconstruction cost of a T-frames sequence that starts at t can be calculated as follows:

$$SecRecCost(t) = \sum_{t'=t}^{t+T} e(t') \tag{3.4}$$

- Then we compute the abnormality score Sa(t) by scaling between 0 and 1.

$$s_a(t) = \frac{SecRecCost(t) - SecRecCost(t)_{\min}}{SecRecCost(t)_{\max}} \tag{3.5}$$

- We can derive regularity score Sr(t) by subtracting abnormality scores from 1.

$$s_r(t) = 1 - s_a(t) \tag{3.6}$$

Evaluation Metric

To evaluate our proposed algorithm and in order to compare it with other state-of-the-art algorithms, we have used the AUC-ROC metric which has been explained in the second chapter.

3.4.4 Experiments and Evaluation

Choice of thresholds

After many tests, we have chosen these values of thresholds :

Dataset	Temporal Threshold	Spatial Threshold
CUHK Avenue	0.95	24
UCSD Ped1	0.92	24
UCSD Ped2	0.95	24
ShanghaiTech	0.75	24

Tableau 3.3: Choice of temporal and spatial thresholds

Evolution of loss function

For example, we have trained our model using the UCSD ped2 dataset for 35 epochs and we got the following graph :

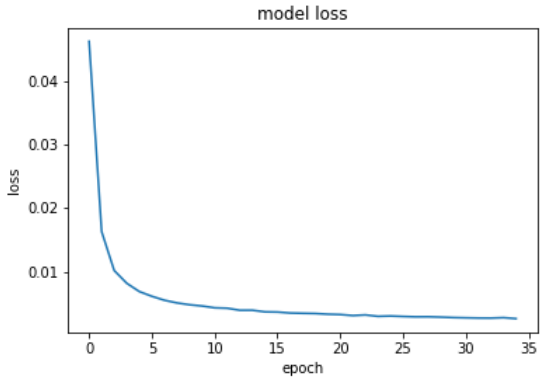


Figure 3.8: Loss graph for UCSD pedestrian dataset (ped2)

- During the first five epochs, the loss decreases rapidly but after the fifth epoch, it decreases slowly.
- In the training phase, our goal is to maximum fit our model to the training set. So, we don't need to worry about the over-fitting and under-fitting problems.
- Then, by increasing the number of epochs, we will get always a better model so better results.

Test our model on a testing video

For testing, we have tested our model on various videos. For example, we have chosen the fourth video of the UCSD ped2 dataset. The following graph shows the evolution of the regularity score for different frames of this video.

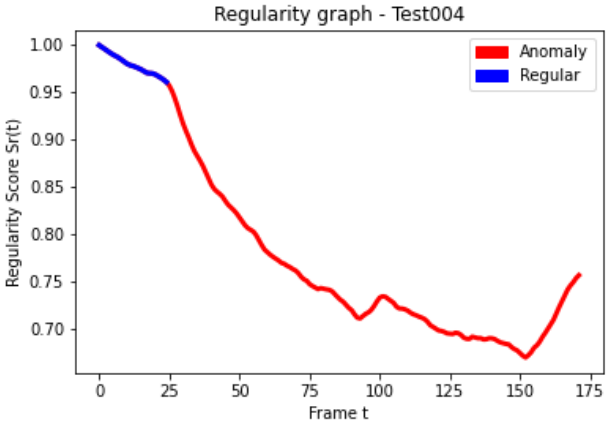


Figure 3.9: Regularity graph for test video N°4

The values of regularity score, that are superior to the temporal threshold (are colored blue), are detected as normal and abnormal in the other case (are colored red).

We have chosen some frames to show the localization results of our model.

(a) Frame 23 (b) Frame 143 (c) Frame 155

Figure 3.10: Various example results given by the model for this video

Comments :

- Frame 23 : According to the regularity graph, this frame is classified as normal (regular). So we can see that there's no anomaly localized in that frame.
- Frame 143 : According to the regularity graph, this frame is classified as abnormal. So we can see that there 4 anomalies localized in that frame. Three of them are real anomalies (True Positive) which are the car, the bicycle, and a man who walks in the wrong direction. The fourth object is classified as abnormal (the man in the center of the frame) but is normal action (False Positive).
- Frame 155 : According to the regularity graph, this frame is classified as abnormal. So we can see that there 3 anomalies localized in that frame. All of them are correctly classified.

3.5 Discussions

Our algorithm compared to some of state-of-the-art algorithms

We compare the results of respective models using frame-level ROC curves, the corresponding area under the curve (AUC). The comparison is presented in Table 3.4, where the results appear as reported by respective authors.

Model	Ped 1	Ped 2	Avenue
Conv-AE	81.0	90.0	70.2
S-RBM	70.3	86.4	78.8
Unmasking	68.4	82.2	80.6
ISTL	75.2	91.1	76.8
Our algorithm	70.4	84.2	74.6

Tableau 3.4: Comparison of AUC

Discussion of results

Our model outperforms some methods for each dataset. But it still does not give the best results.

There are some False Positives (the model identifies it as an anomaly but it is normal) in the anomaly detection which are forgiven in this context. They are false alerts. But in some cases, it gives False Negatives (the model identifies it as normal but it is an anomaly) so it will miss a lot of crucial anomalies and it causes a loss of trust in the system.

Our model has been trained for 35 epochs (the UCSD ped2 dataset). Each epoch takes about 35 minutes. This long time per epoch is due to the huge number of parameters which is about 9 million. This low number of epochs can be the cause of some bad results. In fact, for ISTL article [1], they mentioned that they have trained their model for 1500 epochs using high-performance computing specification, which is not affordable using the Google Colab platform.

Conclusion

During this chapter, we presented how we've approached the problem as well as the results we've obtained.

GENERAL CONCLUSION

In this internship, I had implemented a deep learning model that detects anomalies within a video. This model is based on the spatiotemporal auto-encoder. Our model does not only detect anomalies but also can localize them in the spatial reference.

This project can be enhanced and become real-time anomaly detection which is the final purpose of this type of problem. Several features may be added.

In this internship, I had the chance to work in a research-oriented environment and on an important type of data science problem: Anomaly Detection using deep learning techniques.

Also, I had learned many Deep Learning techniques like Auto-encoders, LSTM layers, and some new metrics like AUC-ROC.

BIBLIOGRAPHY

[1] R. Nawaratne et al. "Spatiotemporal Anomaly Detection Using Deep Learning for Real-Time Video Surveillance". In: *IEEE Transactions on Industrial Informatics* 16.1 (2020), pp. 393–402. DOI: 10.1109/TII.2019.2938527.

[2] M. Hasan et al. "Learning Temporal Regularity in Video Sequences". In: (June 2016), pp. 733–742. DOI: 10.1109/CVPR.2016.86.

[3] H. Vu. "Deep Abnormality Detection in Video Data". In: (Aug. 2017), pp. 5217–5218. DOI: 10.24963/ijcai.2017/768.

[4] R. T. Ionescu et al. "Unmasking the abnormal events in video". In: *CoRR* abs/1705.08182 (2017). arXiv: 1705.08182.

[5] W. S. McCulloch. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* (1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259.

[9] X. Shi et al. "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting". In: (2015). arXiv: 1506.04214 [cs.CV].

[11] G. E. Hinton and R. R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: 313.5786 (July 2006), pp. 504–507. DOI: 10.1126/science.1127647.

[12] R. Salakhutdinov and G. Hinton. "Semantic hashing". In: *International Journal of Approximate Reasoning* 50.7 (2009). Special Section on Graphical Models and Information Retrieval, pp. 969–978. ISSN: 0888-613X. DOI: https://doi.org/10.1016/j.ijar.2008.11.006.

WEBOGRAPHY

[6] D. Mishra. *Transposed Convolution Demystified*. URL: https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba.

[7] A. Bhandari. *AUC-ROC Curve in Machine Learning Clearly Explained*. URL: https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/.

[8] *Understanding LSTM Networks*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[10] H. Mujtaba. *Introduction to Autoencoders? What are Autoencoders Applications and Types?* URL: https://www.mygreatlearning.com/blog/autoencoder/.

[13] J. Willaert. *How To Calculate the Mean and Standard Deviation — Normalizing Datasets in Pytorch*. URL: https://towardsdatascience.com/how-to-calculate-the-mean-and-standard-deviation-normalizing-datasets-in-pytorch-704bd7d05f4c.