# Study of Unsupervised Learning for Images and Videos with Specific Applications to CCTV Data

*A Thesis*

submitted to the
Indian Institute of Science Education and Research Pune
In partial fulfilment of the requirements for the
BS-MS Dual Degree Programme
by

**Rishabh Wanjari**

Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

April, 2022

Supervisor: Aniruddha Pant

# Certificate

This is to certify that this dissertation entitled Study of Unsupervised Learning for Images and Videos with Specific Applications to CCTV Data towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Rishabh Wanjari at the Indian Institute of Science Education and Research under the supervision of Aniruddha Pant, AlgoAnalytics, during the academic year 2021-2022.

Aniruddha Pant

Committee:

Aniruddha Pant

Sourabh Dube

# Declaration

I hereby declare that the matter embodied in the report entitled Study of Unsupervised Learning for Images and Videos with Specific Applications to CCTV Data are the results of the work carried out by me at the Indian Institute of Science Education and Research, Pune, under the supervision of Aniruddha Pant and the same has not been submitted elsewhere for any other degree.

Rishabh Wanjari

*This thesis is dedicated to everyone who ever had to go through the pain of writing one themselves*

# Acknowledgements

Born too late to explore the Earth,

born too early to explore the galaxy,

but born just in time to be surrounded

by extremely wonderful people.

– *Rishabh Wanjari, circa 2022*

Firstly, I would like to thank LaTeX for being an abhorrent typesetting system. Markdown is so much better. A genuine thank you to the rest of the FOSS community and the developers of other applications I use for making the process much more bearable and enabling the rest of my workflows.

Shoutout to all my compatriots for helping me get through these trying times. All the drama and entertainment that we shared together will leave a lasting mark on me. To try and put into words the emotions and experiences I have gone though would be a futile effort (at least on my part) because mere (250) words cannot capture the insane adventure I have had at IISER. As such, it is left as an exercise to the reader.

Finally, Lakshmi, this one's for you 🥺

# Contents

# List of Figures

# List of Tables

# Abstract

Security is of paramount importance in today's world. Public places such as shopping malls, banks, ATMs, city squares, and parks are increasingly equipped with CCTV cameras. These cameras aid in monitoring these spaces and keeping them safe for citizens to use. However, such a large amount of video data cannot be constantly monitored in real-time by humans. Such monitoring would require trained, vigilant workers whose sense of judgement can be trusted. Anomalous behaviour is rare, making the job harder to perform for humans. Additionally, the definition of such behaviour varies by time, place and context. As a result, there is a large demand for this monitoring to be automated. Such automation would need to be accurate, fast and reliable. It would lead to better security and enable monitoring in a larger area. This work aims to use unsupervised deep learning networks to automatically identify anomalies in such data and report them in real-time.

# Chapter 1

# Introduction

Closed-circuit television (CCTV) was first developed in the late 1920s. It involves sending signals from specific video cameras to another location, usually a monitoring room. This system was initially developed during World War II for weapon testing purposes [1], where scientists and engineers could not safely observe the impact of these weapons. This wartime technology was then commercialised, which eventually led to the widespread use of CCTV we see today for surveillance purposes. The early implementations of this technology involved live monitoring, as there was no way to write information to a storage device. As technology improved, recording this information for later use became possible. The convenience this provided led to the technology becoming very popular. With time, it also became easier to incorporate a multitude of cameras into a single monitoring system. In recent years, miniature cameras that can be attached to a person's body, known as body cams, have gained popularity, particularly within law enforcement. Today, this technology is widespread worldwide, used on all scales, from individuals to large organisations, from homeowners to the government. [2]

## 1.1 Motivation

According to a recent market report by IHS Markit ([3] [4]), approximately 770 million CCTV cameras are present in 150 of the world's major cities. These cameras are primarily used for monitoring and surveillance purposes, mainly in the public sector, such as retail stores, malls and small businesses. Recently, there has been a rise in CCTV monitoring amongst homeowners. In both settings, cameras are primarily used as a deterrent to crime, as well as for crime-solving in the event of said crime. Major cities also use these cameras on their roads, not as speed cameras, but as a way to monitor traffic flow and inform decisions related to traffic management. Organisations may also use these cameras to monitor their employees. This monitoring is seen more commonly in warehouses and factories than in corporate offices.

### 1.1.1 Purpose

The monitoring of multiple streams of CCTV footage usually takes place in a central monitoring room. It may be staffed by numerous security personnel. However, commonly, it is only staffed by a single person or, in some cases, none and serves only to record information, particularly for small organisations or businesses. In such cases, it becomes increasingly difficult for the staffed personnel to monitor all the streams with a high enough level of attention. The problem only gets worse with an increase in cameras. This issue makes it much easier for mistakes to occur, leading to damages, physical or otherwise, to the concerned organisation.

This work aims to alleviate the problem mentioned above by developing an end-to-end pipeline that can monitor these camera feeds directly and inform the system if any abnormal activity occurs. This pipeline consists of data ingestion from camera feeds, pre-processing this video data into separate frames that can be used by a machine learning model, the model itself, and finally the reporting of the outcome of the model.

We aim **not** to build an all-encompassing model that can work on any camera feed. Instead, we aim to construct a robust model architecture that can be trained on a per camera basis, which can then be deployed to any CCTV system. The intended end user for this pipeline is any person who requires a robust security system, whether that be a simple shopkeeper or a trained security guard. As such, the pipeline must be easy to deploy and provide results that are easily readable and approachable.

With such a pipeline in place at any location, the security personnel's attention can be directed to events that truly matter instead of being wasted on monitoring otherwise uneventful streams.

### 1.1.2 Scope

We have set a few expectations and limitations for the project due to the time constraints and implementation details:

- A single model architecture must work across different datasets, with different training, testing, and validation data.
- The model must run be performant on low-end dedicated GPUs without sacrificing performance in meaningful metrics.
- The datasets used must use stationary cameras. This includes data from traditional surveillance cameras but excludes body cams or vehicle cameras.
- We would like to stress that the goal of this model is only to detect anomalies reliably. We do not concern ourselves with the exact details of each anomaly, as other object detection methods can handle this.

## 1.2 Outline

The upcoming Chapter 3 addresses the theoretical and practical background necessary to grasp the scope of this work entirely. Chapter 4 discusses the model we have arrived at through this research and the data used to evaluate it. Chapter 5 will scrutinise the results and the methods taken to produce them. Finally, we will summarise what we have achieved and give an outlook on what avenues future research in this area could take.

# Chapter 2

# Theoretical and Practical Background

We have used an unsupervised machine learning algorithm to achieve our results in this project. In this section, we break down the constituent parts of this model.

## 2.1 Previous Work Done

There has been considerable work done in the field of anomaly detection in surveillance footage already [5]. These methods have used machine learning techniques in both supervised and unsupervised contexts. They have used PCA ([6] [7]), SVMs ([8] [9] [10]), isolation forests ([11]), K-means clustering ([12]), Gaussian mixture models ([13] [14]). In particular, one class SVMs, or OC-SVMs, are a popular technique. However, their performance starts to degrade as the size and complexity of the data increases. Approaches have been suggested to alleviate this scalability problem ([10] [15] [16] [17]). A deep clustering embedding (DEC) algorithm that integrates an unsupervised autoencoder with a clustering network has also been shown to be successful in these types of tasks. [18]

Broadly, there are three prevalent types of approaches to this problem [19]:

1. Deep learning for feature extraction: The deep learning part of the model is separated from the anomaly detection aspect. The deep learning network is used for independent feature extraction from the data, which is then interpreted separately.
2. Learning feature representations of normality: The two parts of the model are more closely linked here, with the normality or some other measure of anomalies used to influence the learning of the network.
3. End-to-end anomaly score learning: There are no two separate parts to the model. It is a complete end to end machine learning model that is used to learn the anomaly scores and evaluate the final results.

Another approach that has seen some research interest is to use transfer learning to train a deep learning model with pre-trained layers from popular networks such as AlexNet ([20]), VGG ([21]) and ResNet ([22]).

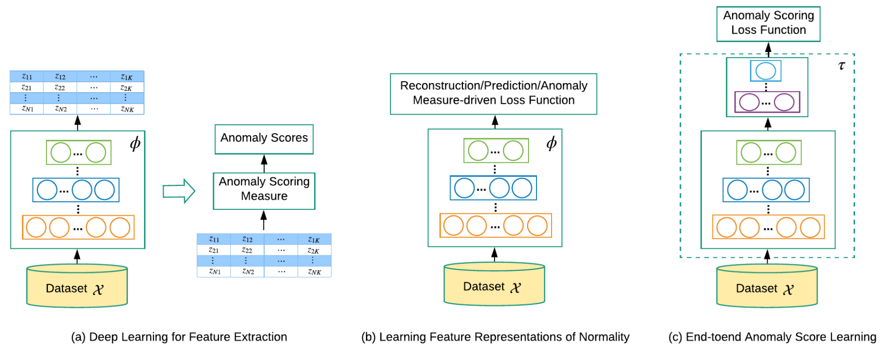

Anomaly Scores
Anomaly Scoring Measure
Dataset $\mathcal{X}$
$\phi$
Reconstruction/Prediction/Anomaly Measure-driven Loss Function
Anomaly Scoring Loss Function
$\tau$
(a) Deep Learning for Feature Extraction
(b) Learning Feature Representations of Normality
(c) End-toend Anomaly Score Learning

Figure 2.1: A visual representation of the three approaches. [19]

These pre-trained layers are used to extract low level features that can then be used for further processing to determine an anomaly score.

The predominant amount of approaches use spatial data to evaluate anomalies. However, competing top models in the field also incorporate temporal data through various means - the frame order being chief amongst them. However, researchers have also utilised frame-level optical flow images ([23]) to achieve excellent metrics.

The current state of the art model [24] utilises two GAN networks working in tandem to determine the anomaly score of a single frame. The first GAN takes an RGB frame as input and attempts to construct an optical flow image. The second GAN performs the opposite process: it attempts to construct an RGB image from an optical flow image. The final anomaly score is calculated using a combination of the reconstruction errors of both images. This approach makes it possible to make pixel-level detections possible.

A closely competing model, which preceded the previous method, uses optical flow as well [15]. However, this model maps the RGB image to a binary map using their novel Binary Quantisation layer to track temporal changes. The intuition behind the idea is that each set of frames should have similar binary maps unless a significant amount of motion is present. This method also achieves good pixel level detection accuracy. Such pixel-level accuracy is not necessary for our use case, and as such, we have decided to focus on achieving good performance at frame level detection.

We have also tested a small portion of open-source models that are readily available online ([10] [25]). While we were able to replicate the results achieved in their respective papers, we found that these models could not be used in high dimensional complex datasets such as ours.

The approach described here falls squarely in the second category in the classification used previously. A crucial way in which we differentiate our approach is by putting an emphasis on model performance on hardware, something that is scarcely mentioned in other work. We aim to produce a model architecture that is both performant and accurate, with a focus on performance, as this model can be deployed in locations with a multitude of cameras, where it will be crucial to deliver fast results, or in local shops, where it is difficult to access a large amount of computing power.

## 2.2 Machine Learning

Machine learning is a field of study that gives computers the ability to learn from data without explicitly being programmed to do so. A slightly more technical definition is

> A computer program is said to learn from experience **E** with respect to some task **T** and some performance measure **P**, if its performance on **T**, as measured by **P**, improves with experience **E**.
>
> *Tom Mitchell, 1997* [26]

Common examples of machine learning include spam filters, chat-bots, computer adversaries for games such as chess, weather forecasting, voice assistants, etc. Machine learning systems can broadly be classified into the following non-exclusive categories:

- Whether they are trained with human supervision
  1. Supervised learning: the training data contains the desired outcome for each data point along with it, known as labels.
  2. Unsupervised learning: the training data does not contain any labels
  3. Semi-supervised learning: only a tiny portion of the training data is labelled
- Whether they can learn incrementally
  1. Batch learning: the system is not capable of learning in increments and must be trained on the entire training dataset at once
  2. Online learning: the system is training by feeding it data either in small groups called mini-batches or individually.
- How the system learns and performs predictions
  1. Instance-based learning: the system works by comparing new data points to old known data points
  2. Model-based learning: the system constructs a model by detecting patterns in the data and building a predictive model

### 2.2.1 Parameters and Hyperparameters

A machine learning model is defined by the parameters it has learned during training. With neural networks, this generally refers to the weights and biases of each individual neuron. Model training involves iterating on the values of these parameters until a satisfactory score in the relevant metric(s) is reached. During the training process, we can affect the parameters it has learned. This tuning is done using hyperparameters. Hyperparameters are not necessarily part of the resultant model. Most hyperparameters are used only during the training process and can not be inferred from the final trained model. Examples of hyperparameters include but are not limited to train-test-validation split ratio, choice of optimisation algorithm, choice of the loss function, number of layers, batch size, learning rate, etc.

#### 2.2.1.1 Randomised Search

The choice of hyperparameters can drastically impact the final performance of the model. The process of choosing the hyperparameters is called hyperparameter tuning and is vital to get it right. It can be performed systematically using a method known as randomised search. A traditional method of searching for the best set of hyperparameters would be to present a list of possible values for each hyperparameter, try every possible combination, and return the set that performed the best. This method is known as grid search and can be automated. The major drawback of this approach is that it suffers heavily from the curse of dimensionality.

> **Curse of Dimensionality**: In higher dimensions, as is the case with images, where the value of each pixel is treated as a separate dimension, each data point is, on average, further to other points. It becomes much harder to make predictions in these higher dimensions, as they will be based on larger extrapolations. This makes increases the risk of over-fitting the data.

Even optimising three or four hyperparameters can take a very long time to evaluate. In contrast to this, as the name suggests, random search is a technique where each hyperparameter is selected randomly. To avoid extreme values, each hyperparameter is given a distribution from which it can be sampled. Research has shown both empirically and theoretically that randomised search performs better than grid search [27].

## 2.3 Data

In order to make a successful model, one must provide it with ample, good quality training data. The model must then be tested against equally high-quality testing and validation data.

### 2.3.1 High-Quality Data

A traditional acronym in the field of computing is *GIGO* - garbage in, garbage out. This expression is particularly true for the data-intensive field of machine learning. As the goal of any ML algorithm is to learn from the data provided to it, said data must be of the highest quality possible. Even minor errors can lead to quantifiable errors in the final output, especially when the data set is small. In particular, with labelled data, it is essential to verify that the labels provided are correct. These elements make performing an intermediate data pre-processing step essential before feeding it into any machine learning algorithm.

### 2.3.2 Training Dataset

This dataset is the sample of data that is used to fit the model. It is the data that the model *learns* from.

### 2.3.3 Validation Dataset

This sample is used to evaluate the model's performance frequently. It is used to tune the model's higher-level parameters: the hyper-parameters. The model itself does not directly learn from this data. However, the tuning of hyper-parameters does get affected by the bias towards achieving better performance on the validation set.

### 2.3.4 Test Dataset

Once the model has been thoroughly trained using both training and validation datasets, it is finally tested against the test dataset. This dataset provides insight into how the model will perform on real-world data, as it has never encountered the test dataset before. It is the gold standard for providing accurate evaluation metrics on model performance.

## 2.4 Neural Networks

A subset of machine learning, called deep learning, is based loosely on the biological models of animal brains. This domain uses artificial neural networks to make an algorithm that models various kinds of behaviour.

### 2.4.1 Neurons

A neuron (or a unit) forms the most basic block of a neural network. It can take in any number of inputs to perform a weighted average sum, which then outputs a single number. There may also be a bias term that is present for each neuron. This number is then passed through an **activation function**. This function is necessary as it introduces non-linearity in the system. If this function were not present, upon chaining the outputs of multiple neurons, which perform a linear transform on their inputs, we would only end up with another linear transform. It would mean that a network with multiple layers would be reduced to a single layer, which does not help solve complex problems. The activation function also allows gradient descent to be used to optimise the model.

The output $z$ for a single neuron looks as follows:

$$z = \sum x_i w_i + b$$

There are a few candidates for activation functions that can be used. Here, we list the ones that we have utilised:

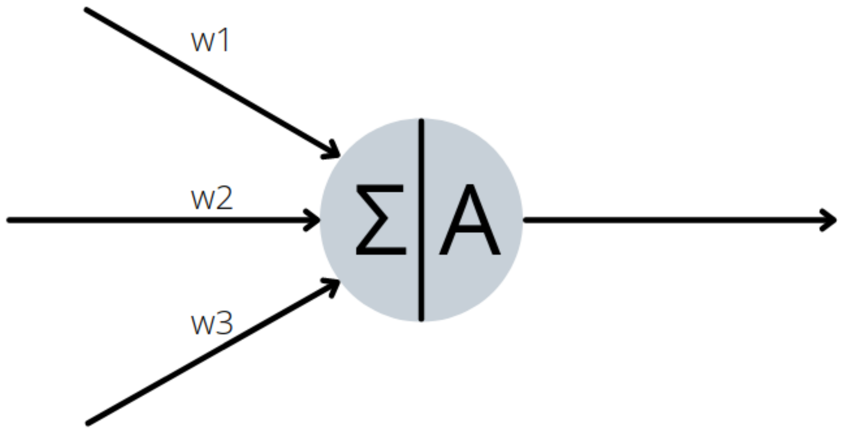


Figure 2.2: A visual representation of a neuron in a neural network.

#### 2.4.1.1 Sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

#### 2.4.1.2 Hyperbolic Tangent

$$tanh(z) = 2\sigma(2z) - 1$$

#### 2.4.1.3 Rectified Linear Unit (ReLU)

$$ReLU(z) = \max(0, z)$$

#### 2.4.1.4 Leaky ReLU

$$LeakyReLU(z) = \begin{cases} \alpha z \text{ for } z < 0 \\ z \text{ for } z \geq 0 \end{cases}$$

### 2.4.2 Gradient Descent

The model will understandably not perform very well on an initial pass of data through the model. An algorithm is needed to optimise the model's parameters to fit the data better. Gradient descent fills this role. It is a method that can find the optimal solutions or get as close as possible to the optimal solution to many problems. The model's fit to the data can be expressed through a cost function that needs to be minimised. Finding the best fit is equivalent to finding the global minima of the cost function. Gradient descent achieves this by tweaking parameters iteratively until it reaches this minima or gets as close to it as possible. The first step is to randomly initialise a vector $V$ with components $v_i$ on the loss function $L(V)$. This vector is the collection of the model parameters that need to be optimised. After this, we calculate the

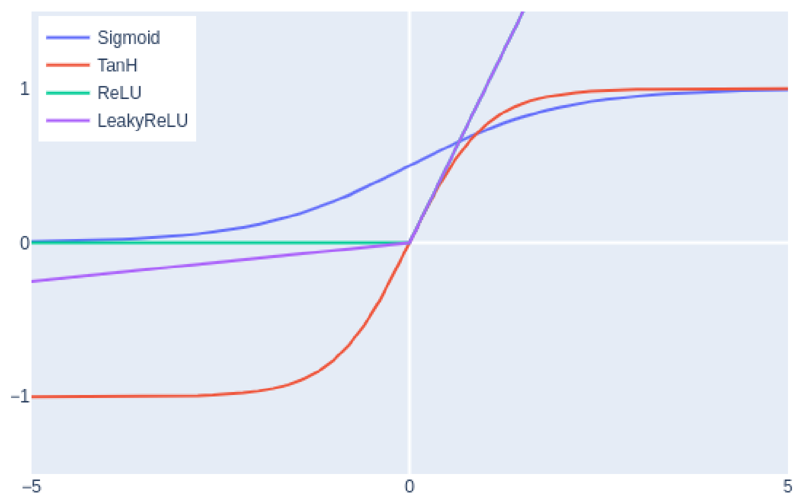


Figure 2.3: A graph of the various activation functions used

local gradient of the given point on the loss function.

$$\nabla_V L(V) = \begin{pmatrix} \frac{\delta}{\delta v_0} L(V) \\ \frac{\delta}{\delta v_1} L(V) \\ \vdots \\ \frac{\delta}{\delta v_n} L(V) \end{pmatrix}$$

Then, we can iteratively change the vector in the direction opposite the gradient.

$$V_{next} = V - \eta \nabla_V L(V)$$

Here, $\eta$ is the learning rate, which determines the magnitude of the change in the vector. $\eta$ is an important hyper-parameter of any model. Too low, and the model will take too long to converge, while setting it too high will make it so that the model can never reach the minima, and the model will continue to bounce around the region of local minima.

#### 2.4.2.1 Early Stopping

A common practice that we have employed involves monitoring the validation loss of the model on a separate holdout validation set for each epoch and halting model training when the validation loss reaches a minimum and performance starts to degrade on further iterations. This simple method is known as early stopping. This performance loss occurs because there comes a point when the model will stop generalising the data and begin to learn the noise and other undesired features in the data. In other words, it will begin over-fitting the training data. We provide some leeway on the validation performance degradation by giving the model a small number of epochs within which the validation loss must continue to decrease. This is done because

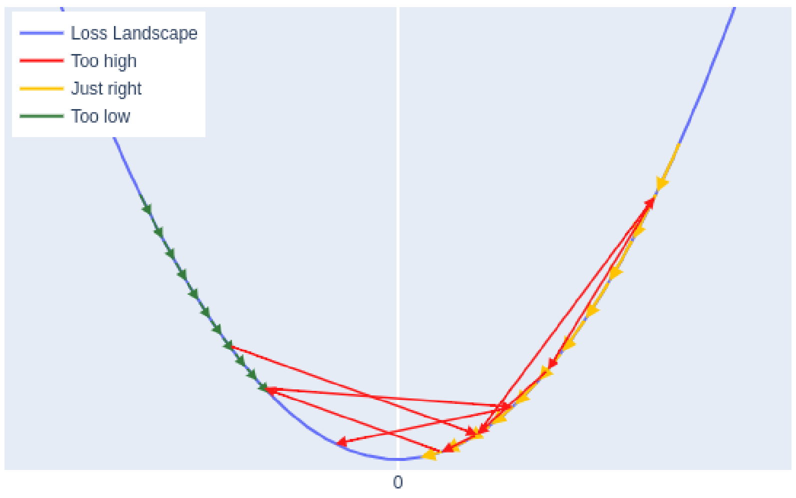


Figure 2.4: A visual representation that shows the importance of using the correct learning rate

some fortuitousness is involved with mini-batch gradient descent discussed below. This buffer makes sure that we do not stop the model before it has learned how to generalise at all.

#### 2.4.2.2 Mini Batch Gradient Descent

Computing the gradient with the complete training set for each iteration of the algorithm is computationally intensive. Instead, we can calculate the gradient using a small sampled subset of the data in *mini-batches*. This method, while more stochastic, drastically increases the model's performance, especially while using dedicated GPUs.

### 2.4.3 Backpropagation

In neural networks, we utilise the backpropagation algorithm [28]. It is an implementation of gradient descent that can efficiently compute the gradients of every model parameter (each neuron's weights and biases). For each training batch: - It computes the final prediction of the model while storing the values of the intermediate steps as well. - It then calculates the prediction error using a loss function that is defined along with the model. - Using the chain rule, it can determine the contributions of the previous layer to the resulting error. This step is done recursively, backwards, until the input layer. - Now that it has determined the gradients for each parameter, it performs a gradient descent step to update the model parameters. This step can be repeated until desired. Ideally, gradient descent is repeated until the error goes to 0. Practically, gradient descent is performed for a set number of **epochs**, which may range from the hundreds to the thousands, depending on the complexity of the model.

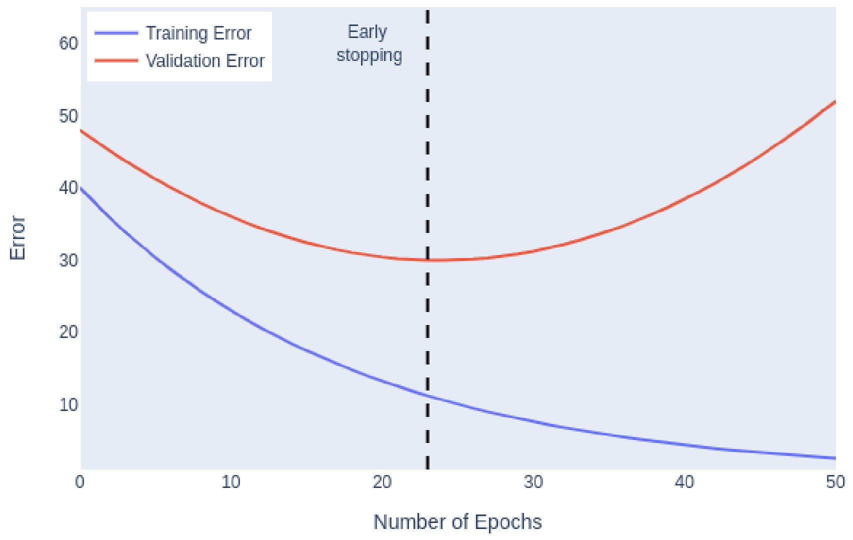


Figure 2.5: A graphical representation of early stopping. We terminate training at epoch 23.

### 2.4.4 Kernel Initialisation

Another parameter that requires some amount of consideration is the initialisation of the weights and biases of the network.

#### 2.4.4.1 Weight Initialisation

One might be inclined to think that using a zero initialisation for the weights might be feasible. However, in practicality, the derivatives with respect to the loss function end up being the same. So, all neurons are updated to identical values. The bias term plays no role in the derivative. This essentially makes the model a linear transform, and it fails to break the initial symmetry. [29]

**2.4.4.1.1 Random Initialisation** So if zero initialisation fails, then the other approach is to use random initialisation since we have no idea what the final weights of each neuron will look like. However, this initialisation can not be completely random. - If the weights are too high, the output of the activation function such as $\sigma(z)$ or $\tanh(z)$ will be near 1, where there is almost no gradient slope. With the $ReLU(z)$ type of functions, it can cause the opposite problem, where the gradients become too large. - Similarly, if the weights are too low, the gradient vanishes. This can occur for almost all commonly used activation functions. There are a few initialisation techniques that can help with the above problems:

- Xavier/Glorot Initialisation ([30]): This technique aims to make the variance of the outputs of a layer equal to the variance of its inputs. While initially developed for the sigmoid activation function, since it was popular at the time, it still sees use today when the tanh activation function is used.
- He initialisation ([31]) This follows the same idea as Xavier initialisation (making the variances similar)

but is more suited for layers with ReLU activation functions.

#### 2.4.4.2 Bias Initialisation

It is common practice to initialise biases to zero since the initial break in the symmetry of the network is provided by the weights. There are some suggestions to initialise all biases to a small positive number, but it is unclear if there is any improvement with this method. Hence, we have used zero initialisation for the biases.

### 2.4.5 Batch Normalisation

A common problem with ML models is that they can easily over-fit the training data. One of the ways that this can be alleviated is to use regularisation. We can do this at the output of each neuron with batch normalisation. This technique involves adding an extra computation step just before the activation function in each neuron. We can calculate the mean and standard deviation of the batch: $\mu, \sigma$. This step first standardises the output of each neuron per mini-batch to a standard Gaussian.

$$Z_{norm}^{(i)} = \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 - \epsilon}}$$

$\epsilon$ is a small smoothing term that prevents division by 0 errors. In the second step, the distribution is adjusted through two *learnable* hyper-parameters $\gamma, \beta$, which apply a linear transform to the normalised distribution. This means that these two parameters can be optimised using the gradient descent discussed above throughout the training process.

$$Z^{(i)} = \gamma Z_{norm}^{(i)} + \beta$$

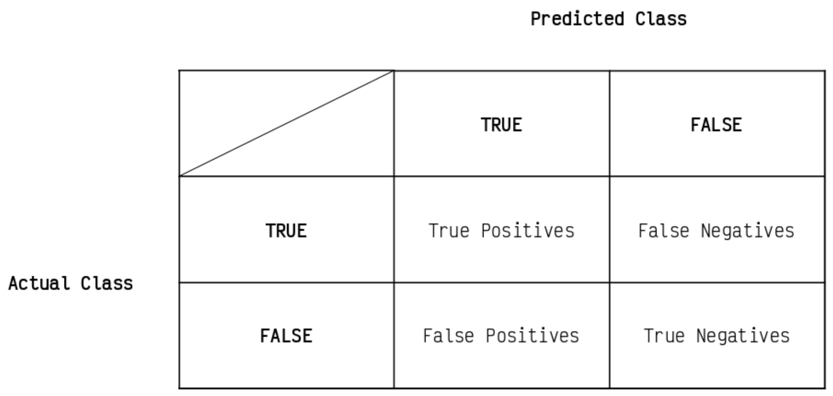
This process reduces the inter-dependency between the neurons, making it easier to pinpoint errors with gradient descent. It also has the added benefits of making model training converge faster and more stable.

## 2.5 Evaluation Metrics

As the goal of the current work is to classify events simply as anomalous or not, it can be treated as a binary classification problem. This allows us to have all the metrics associated with such a classification at our disposal. We have decided to focus our attention on a select few of them.

### 2.5.1 Confusion Matrix

The confusion matrix is a way to present the true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) classified by a model. These numbers are calculated once the model's outputs, whose outputs may not be binary, have been passed through a threshold value to determine which class the predictions belong to. Figure 2.6 provides us with a sample confusion matrix.



| | | Predicted Class | |
|---|---|---|---|
| | | TRUE | FALSE |
| Actual Class | TRUE | True Positives | False Negatives |
| | FALSE | False Positives | True Negatives |

Figure 2.6: An example confusion matrix.

It gives us a clear picture of how the model is performing and possibilities for where the model can be improved. From this matrix, a plethora of other metrics can be calculated.

### 2.5.2 Precision

It is a measure of the fraction of the predicted positives that are genuinely positive.

$$Precision = \frac{tp}{tp + fp}$$

The higher, the better.

### 2.5.3 True Positive Rate (TPR)

Also known as sensitivity or recall, it measures how many genuinely positive data points the model has actually classified as positive.

$$TPR \text{ or } Recall = \frac{tp}{tp + fn}$$

The higher, the better.

### 2.5.4 False Positive Rate (FPR)

The FPR measures how many negative points the model has erroneously labelled as positive.

$$FPR = \frac{fp}{fp + tn}$$

The lower, the better.

### 2.5.5 F-score

This metric simply combines both precision and recall into a single metric. The general F-score is given by

$$\beta = (1 + \beta^2) \times \frac{precision \times recall}{(\beta^2 * precision) + recall}$$

The choice of $\beta$ determines the weight given to each metric. A higher $\beta$ means that recall is a more valuable metric. We use the $F_1$-score, which means we equally value precision and recall. In this case, the $F_1$-score is given by

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

The higher, the better.

### 2.5.6 AUROC

The ROC (Receiver Operating Characteristic) is a curve that visualises the trade-off between the true positive rate (TPR) and the false positive rate (FPR) as the threshold is varied. The area under this curve (AUROC) gives us a measure of the model to differentiate between the two classes. The higher the AUROC, the better the model can distinguish between two classes. A perfect model would have an AUROC of 1. A completely random classifier that predicts the correct class half the time has an AUROC of 0.5. Most models lie between this range of AUROCs.

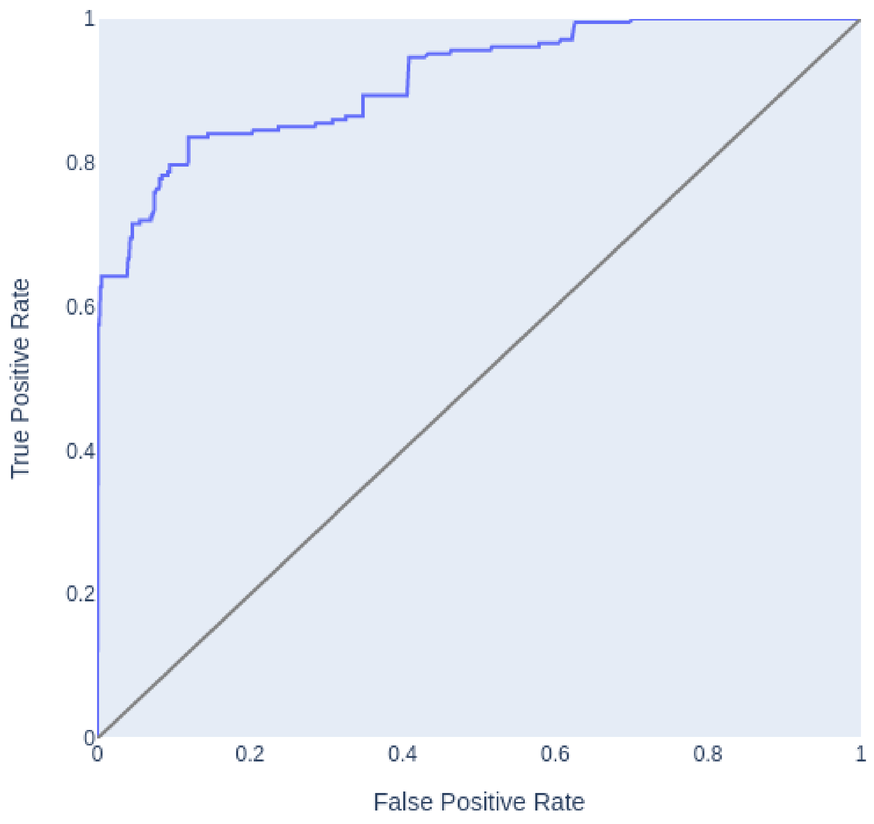


Figure 2.7: A typical ROC plot. This plot has an AUROC of 0.85

AUROC curves are not smooth and are plotted for discrete values of thresholds. For every possible threshold value, we calculate both TPR and FPR, which we then plot on a single chart. These decision thresholds are implicit and not shown on an axis. Similarly, the actual area is never highlighted, and the actual shape of the curve is given precedence.

#### 2.5.6.1 Limitations

We are aware that AUROC has its limitations. It can be an “overly optimistic” metric on the model’s performance on datasets with very few negative samples, as is the case with anomalous data. To combat this, we have made sure to use testing sets with a large number of negative data points.

### 2.5.7 Youden’s J Statistic

To select the best threshold that provides a balance of sensitivity and specificity, we utilise Youden’s J statistic, which is calculated as follows:

$$J = sensitivity + specificity - 1$$

The values for this statistic range from 0 to 1. It gives equal weight to both types of misclassifications, i.e., false positives and false negatives. Youden’s $J$ can be visualised easily on the ROC curve. Every point on the curve has an associated $J$, which can be interpreted as the height of the point above the random classifier line $y = x$. To pick the best threshold, then, is to pick the point with the highest $J$, which is the point farthest from the random classifier line.

### 2.5.8 AUPRC

The AUPRC is the area under the precision-recall curve. The PR curve shows off the trade-off between precision and recall as a function of the threshold. It is more beneficial for imbalanced datasets. We have tried to reduce this imbalance in the test datasets that we have, but they are still not balanced. Hence, it is a valuable metric in this context as well. It is useful when one prioritises finding all the positive samples (in this case, the positive samples are the anomalies, while the negative samples are the normal frames) in a dataset. ([32] [33]) While the baseline for the AUROC curve is 0.5, which is given by a completely random classifier, the baseline for the AUPRC is given by the fraction of positive samples in the dataset. So, a good value for the AUPRC is dataset dependent. Thus, it is inappropriate to compare the absolute values of AUROC and AUPRC, even though a higher value is generally better for both.

### 2.5.9 Reconstruction Error

The approach we employ with this method is to analyse how well the candidate model can reconstruct the input images. We can think of these two images, one original and the other reconstructed by the model, as input signals to a measure that can describe how similar these two images are. For more than 50 years, Mean Squared Error (MSE) has been the prevalent method to compare two signals. [34]

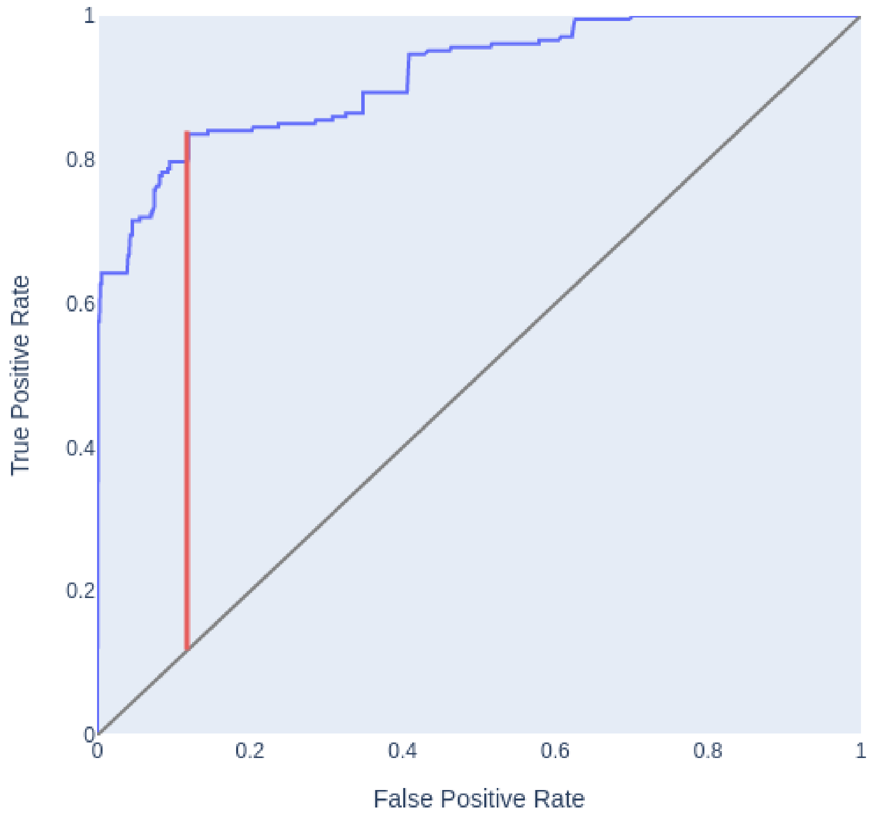


Figure 2.8: The red line represents the value of Youden's J statistic.

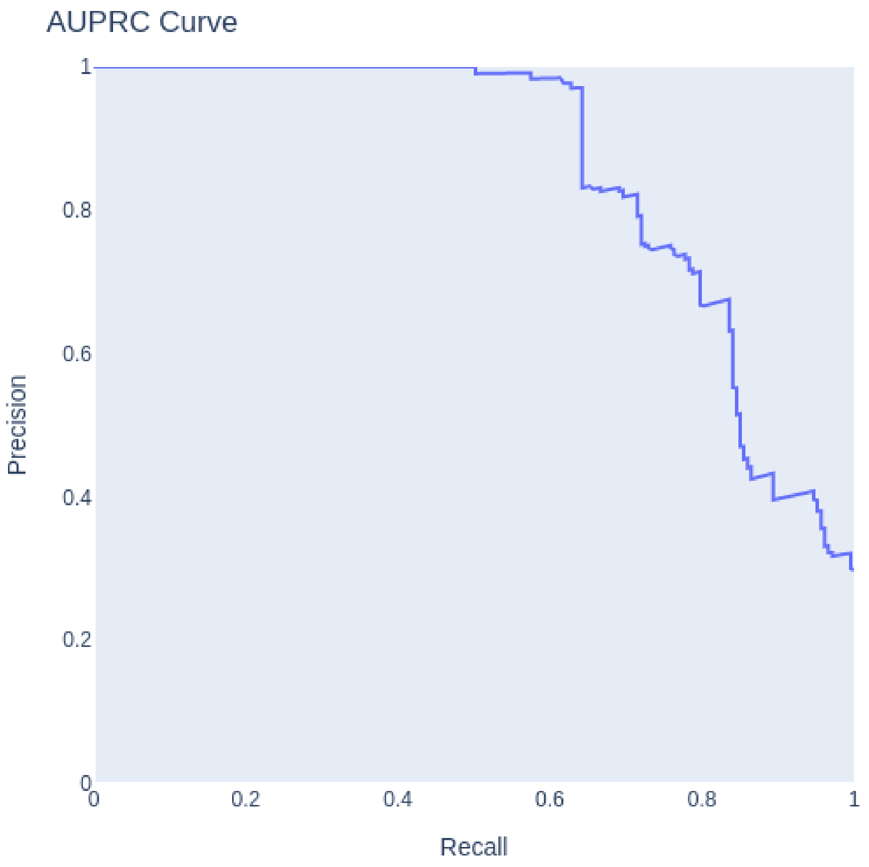


Figure 2.9: An example of an AUPRC curve

#### 2.5.9.1 Mean Squared Error (MSE)

MSE is a signal fidelity measure that compares two signals by providing a score that describes how similar they are. In most cases, one of the signals is assumed to be the original, best possible signal, without any errors, while the other is a distorted version whose quality is being evaluated. Under this assumption, MSE can be used to measure signal quality. This applies to our case as well. If the two signals are $\mathbf{x}, \mathbf{y}$ with finite discrete components $x_i, y_i$, with $i = (1, N)$ where $N$ is the total number of components (for images, these components are pixels), then the MSE between the two signals is given by:

$$E(x,y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2$$

We can see a few important points from this definition:

- MSE(x,y) seemingly has no strict upper bound, and the maximum value depends on how the signal is measured.
- MSE(x,y)=0 if and only if x=y
- MSE(x,y)=MSE(y,x)

**2.5.9.1.1 Is MSE suitable?** There is evidence that MSE is not suitable for image processing tasks. As seen in Figure 2.10, MSE can be significantly large for images that, visually, look very similar.

There are some implicit assumptions one makes when choosing MSE as an evaluation metric:

- Signal fidelity is independent of spatial or temporal relationships between the pixels
- All pixels contribute equally to signal fidelity
- Signal fidelity is independent of the sign of the error between pixels These assumptions are very powerful and impose certain limitations on where MSE can be used accurately. In the realm of visual perception of images, these assumptions do not hold. As seen in Figure 2.11, images that look eclectically different to the human eye can have the same MSE value.

There is another alternative that we can use, called the Structural Similarity Index (SSIM).

#### 2.5.9.2 Structural Similarity Index (SSIM)

The main philosophy that underlies the SSIM method is based on human perception and the human visual system. It seeks to measure the structural information present in images, as humans have also been conditioned to do. SSIM measures the similarity of the three aspects: luminance $l(x,y)$, contrast $c(x,y)$, and similarity $s(x,y)$. It does so in patches across the entire image. The final SSIM is obtained by computing the average SSIM of all the patches in the image. As before, we have $\mathbf{x}, \mathbf{y}$ with components $x_i, y_i$. Let $\mu_j, \sigma_j$ be the mean and variance of the $j^{th}$ signal and $\sigma_{xy}$ be the covariance of $\mathbf{x}, \mathbf{y}$. We can think of the mean as an estimate of the luminance, the variance as an estimate of contrast, and the covariance measures the

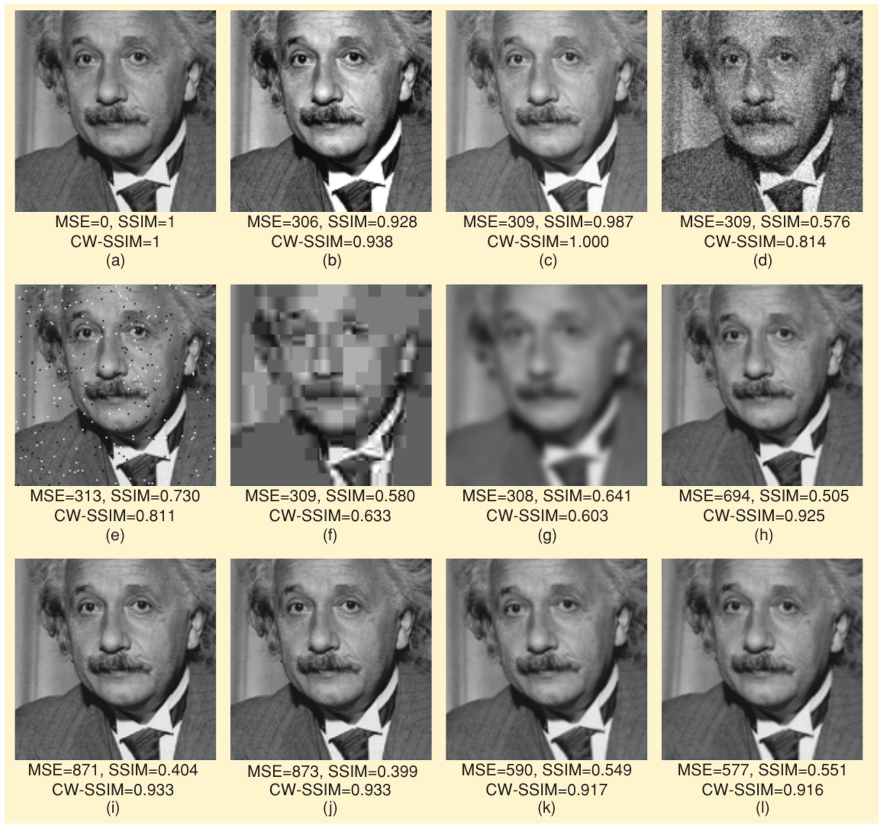


Figure 2.10: Demonstration of lack of usefulness of MSE for images [34]

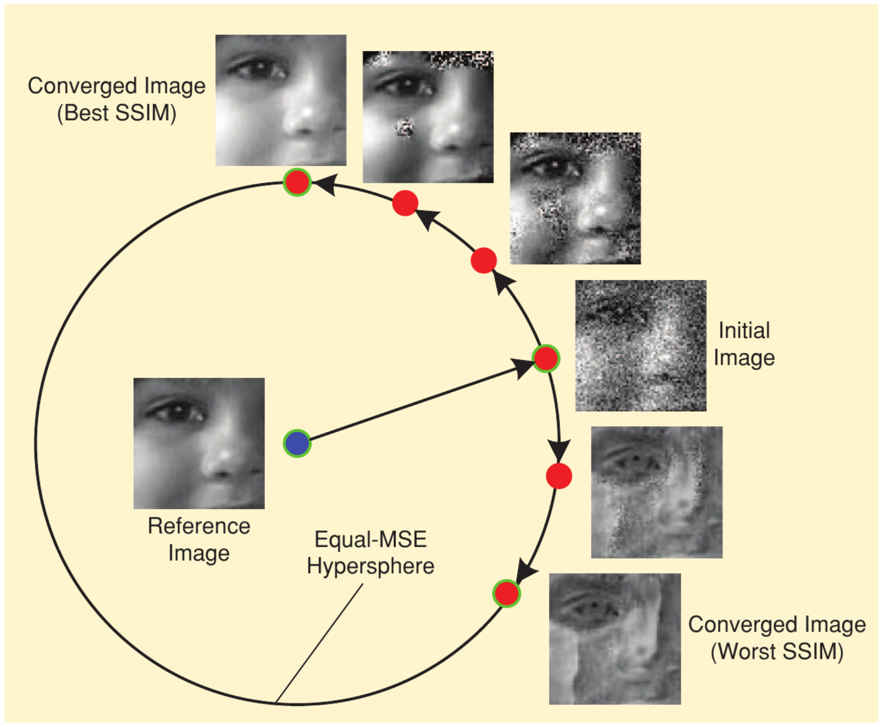


Figure 2.11: An equal MSE hyper-sphere for an image

tendency of the signals to vary together, which can be used to estimate structural similarity. These aspects are given by:

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}$$

where

$$C_1 = (K_1L)^2; C_2 = (K_2L)^2; C_3 = \frac{C_2}{2}$$

Here, $L$ is the dynamic range of the pixel values and $K_1, K_2$ are scalar constants. We have used $K_1 = 0.01$ and $K_2 = 0.03$. Finally, the SSIM index between two signals is given by

$$SSIM(x,y) = [l(x,y)]^{\alpha}[c(x,y)]^{\beta}[s(x,y)]^{\gamma}$$

$\alpha, \beta, \gamma$ are used to set the relative importance of each component. If we set $\alpha = \beta = \gamma = 1$, the SSIM is given by [35]

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

From this, we can see:

- SSIM is bounded.
- $-1 \leq SSIM(x,y) \leq 1$ - $SSIM(x,y) = 1$ if and only if $x = y$ - $SSIM(x,y) = SSIM(y,x)$

The effectiveness of using SSIM over MSE can be seen in the previous two Figures 2.11 and 2.10. One can easily discern that images with higher SSIM do indeed look more similar to the original image.

### 2.5.10 Frame Buffer

Any real-world data set is not perfect. There is an abundant amount of noise present in the data. This noise can also manifest itself in the reconstruction error of each frame. It can cause a short sequence of frames that would otherwise be considered normal to be flagged as anomalous. Likewise, the inverse is also true. There can be short spikes of activity that may be flagged as normal during an anomalous event. To prevent these accidental false spikes, we have implemented a frame buffer system that requires the length of the event (whether anomalous or normal) to exceed a certain number of frames before it can be classified. This value is yet another hyperparameter that we can tune.

# Chapter 3

# Current Model/Approach

## 3.1 Overarching Framework

Anomalies, especially for video data, can be hard to define concretely. In the real world, the definition of an anomaly can vary considerably depending on the situation and context. For example, for a camera monitoring a store, video footage of burglary, arson and violence have entirely different characteristics. Nevertheless, they are all classified as anomalous behaviour (for a reasonable shopkeeper). However, these need not be the only kinds of anomalies. There could be other events that do not fall under any such classification. This is why, traditionally, security personnel must monitor CCTV feeds to ensure maximum security. It is possible to define anomalies in a way that makes it easier for us to analyse them.

> **Anomalies** Also known as outliers or novelties, anomalies are defined as data points that deviate significantly from the normal data.

This definition allows us to focus on the task of identifying anomalies rather than concern ourselves with the exact nature of each anomaly. A data point can be labelled as anomalous if it deviates significantly from the well-established normal pattern for the given data with a chosen metric. It is the model's role to learn what the normal distribution looks like and decide what the threshold for deviating significantly for a point to be classified as an anomaly. This work aims to find an unsupervised machine learning framework that can reliably detect such anomalies in CCTV surveillance footage from empirical data.

> **Why unsupervised learning?** Anomalies are typically a rare phenomenon. In most real-world data, the normal instances make up an overwhelmingly large proportion of the data. As a result, it is not easy to find and make labelled data sets that could be used in a supervised setting.

We aim to achieve this using a weakly labelled dataset. That is, the training data that is fed into the model does not contain any anomalous frames. We, of course, have a fully frame-level labelled dataset available to us so that we can perform an accurate evaluation of the model. However, we do not provide any of these labels to the model. They are only used in the evaluation phase of model testing. Importantly, our aim is not to construct a single all-encompassing model that will work on all eight datasets. Instead, our goal is to construct a reliable model architecture that can be trained on individual datasets - which correspond to individual cameras on-premise - and can perform reliably. We aim to create a performant model so that it can be trained and deployed on a large fleet of cameras at once.

The model learns what an ordinary scene looks like for each dataset and the pattern of images it comprises of. As a result, when normal frames are passed to the model for testing, their reconstruction error should be low. When anomalous frames are introduced, their reconstruction error should be relatively higher than that of the normal images since the model has not encountered such images during training. By deciding on an appropriate threshold for the reconstruction error, we can construct a binary classifier than can classify the frame as normal or anomalous.

Autoencoders and GANs are similar in some aspects: both learn dense representations, are unsupervised, and can be used to generate new samples. They also have similar applications. We decided to test out both approaches and see how they would both compare on performance and evaluation metrics with these similarities in mind.

## 3.2 Datasets

In this work, we have used eight different datasets. Each dataset describes a different scenario that such a model can be expected to be deployed in. They contain clearly defined normal and anomalous events that can be used for this analysis. These datasets are described in Table 3.1 We also provide the number of frames avaiable in each dataset in the last column. These datasets have been extracted from *real world* CCTV footage from various events at different locations and times of the day. This large variety in scenarios ensures that any model that is trained using these datasets will be quite robust. We provide a sample of the images from the datasets, with normal and anomalous frames, in Figure 3.1.

For video CCTV data such as ours, we have verified that the video contains no unnecessary frames such as loading screens, black screens, or other unhelpful frames. We have also taken steps to ensure that the labels provided to the test set are not heavily biased by the people who have labelled them. All datasets are available in the RGB colour space. The frame sizes of each dataset vary but have been standardised to fit the model during the pre-processing stage. In particular, with CCTV footage, many camera feeds record at low frame rates and resolutions.

Table 3.1: A detailed description of the eight datasets used.

| Dataset | Description | Anomalous Event(s) | Dataset Size |
|---|---|---|---|
| abbey | A crossroad with both people and cars | Road accidents | 10,726 |
| beach | An empty beach at night | People walking around; torches being lit | 8,404 |
| castro | Another busy street surrounded by buildings | Traffic jams or road accidents | 4,054 |
| meteor-night | A still sky with no motion | Appearance of meteors | 4,604 |
| motorway | A busy highway | Traffic jams | 6,679 |
| tractor | Tractors passing on a dirt road | Tractors catching fire or other accidents | 13,504 |
| volcano | The peak of a volcano | The volcano erupting | 5,132 |

Due to the nature of anomalous events - the fact that a majority of them are rare and occur for a small period of time - the labelling and gathering of these datasets is relatively simple and not time-consuming. In fact, since the training data is weakly labelled as normal, it is straightforward to gather training data from any surveillance camera. The ability to easily create training data makes deploying such a model in real-world scenarios uncomplicated.

### 3.2.1 Image Pre-processing

These raw images can not be fed directly into the model. They must first undergo preliminary processing to ensure the data is of sufficiently high quality and standardised. We have taken the following steps in pre-processing the images from these eight data sets:

- Standard image shape: All images have been re-scaled to have a fixed size of $144 \times 144$ pixels. This involves either scaling up or scaling down the raw images. We have used the techniques described in Section 3.4.2 for up-sampling. These techniques can also be applied to downscale an image, which we have done.
- Colour channels: Images have been converted to greyscale, which decreases the number of colour channels from three for RGB to just one. This step significantly increases performance as we have cut the number of parameters the model needs to learn by one third. We also know that the exact colours of objects do not matter in such anomaly classification. For example, one does not need to know the colour of the cars to detect a traffic jam in the *motorway* data set.
- Normalisation: This step ensures that each input parameter (each individual pixel in this case) has

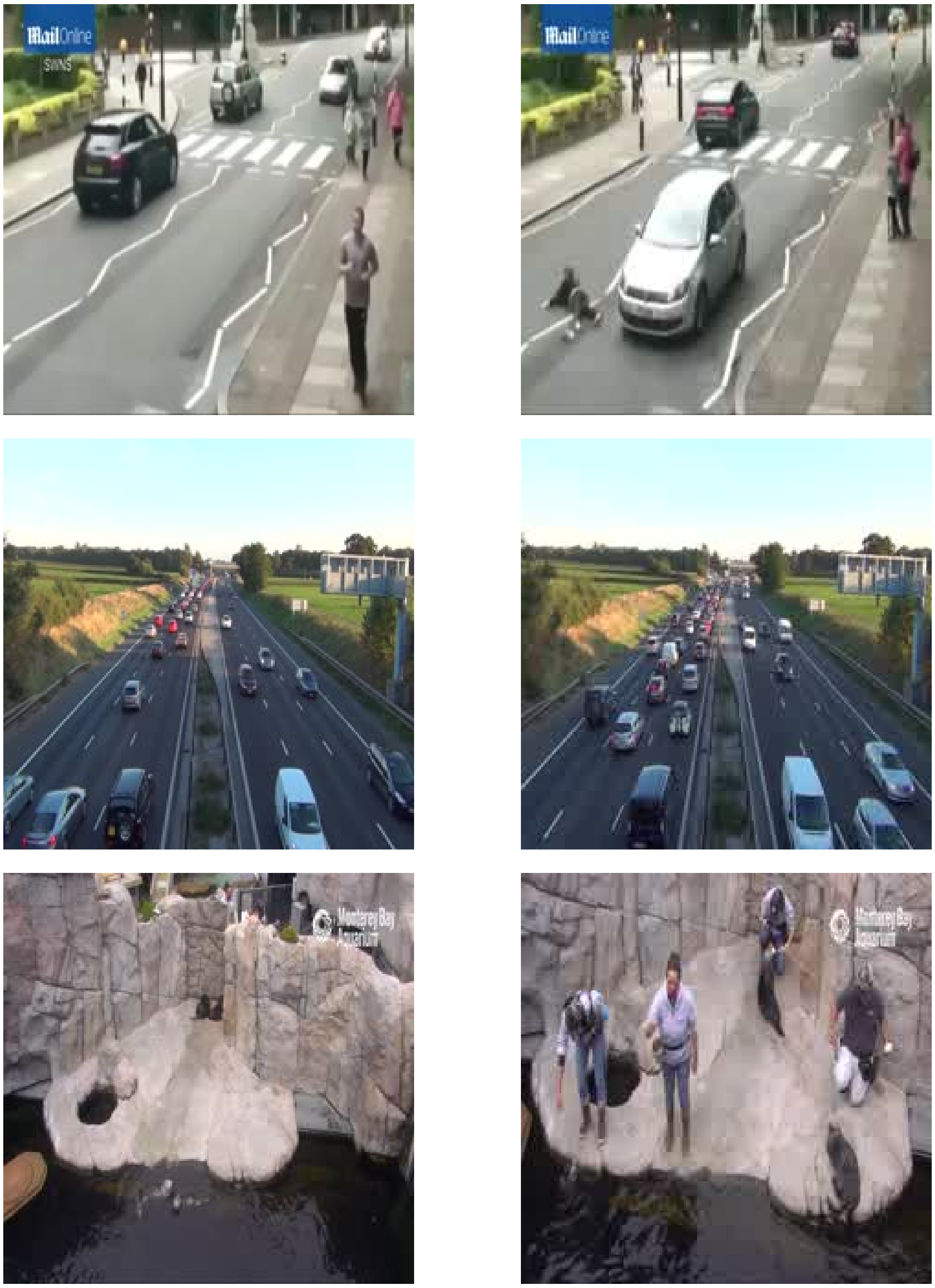


Figure 3.1: A comparison between normal frames and anomalous frames three datasets: abbey, motorway and otter respectively. The normal frames are on the left with the corresponding anomalous frames on the right.

a similar data distribution. This makes it so that a few numerically large parameters do not take precedence over other parameters. The pixels of each image are normalised to the $[-1, 1]$ range.

The input image that is fed to the model is finally a 144px $\times$ 144px greyscale image, where the values of each pixel range from [-1,1]. A comparison between a raw image and its corresponding input image can be found in Figure 3.2.

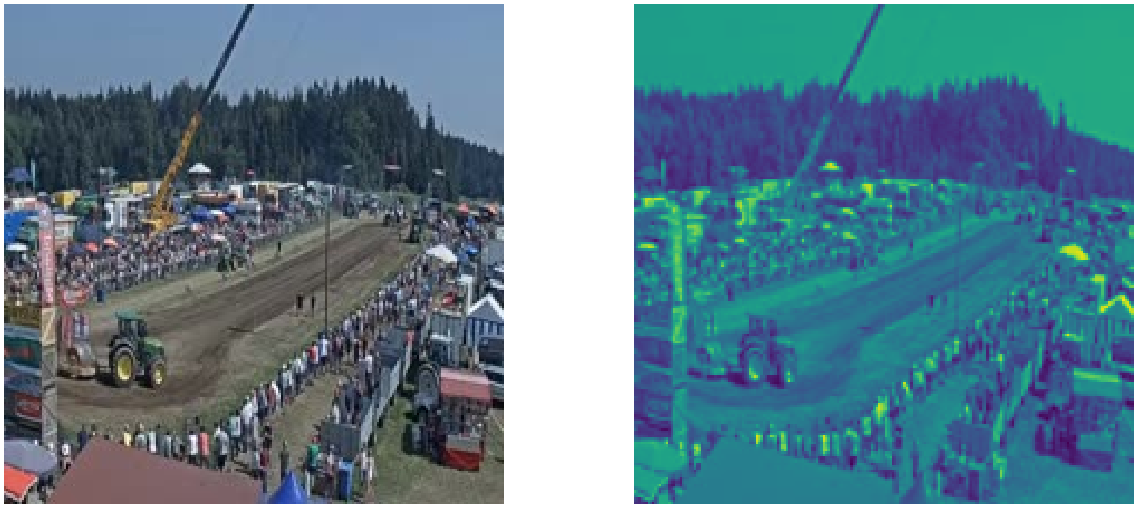


Figure 3.2: A comparison between a raw (left) and pre-processed (right) image. The pre-processed is displayed using the standard *viridis* colour map.

## 3.3 Autoencoders

Autoencoders are an unsupervised learning class of models that are capable of learning dense representations of the input data. These representations have much lower dimensions than the input data, which makes autoencoders great for dimensionality reduction. An autoencoder is trained to output its input. While this may seem like a trivial task, it is complicated by adding additional layers in-between the input and output that prevent the model from learning the identity mappings. This constraint forces the autoencoder to learn efficient representations of the input data in these deeper layers. In other words, it forces the autoencoder to focus on the essential features of the image. It consists of two parts: the encoder, which tries to map the inputs to the latent representation, and the decoder, which tries to map the latent representations back to their original form. In this way, an autoencoder is symmetrical. Since there are no labels provided, the “label” for each pixel is the pixel itself. The difference between the two images is calculated using a reconstruction error. There are a few possible choices for the reconstruction error, as is discussed in Section 2.5.9. Once the complete architecture has been trained, it is possible to use the output of the encoder to obtain a compressed representation of the input, which can be further used for other tasks. Since it is a neural network, it is capable of learning nonlinear relationships within the data. This makes it a more powerful version of PCA

([36]). While PCA aims to learn an optimal hyper-plane that describes the data, autoencoders are able to learn an optimal manifold, which models the data much better.

### 3.3.1 Under-complete Autoencoders

The way we constrain the model in this approach is to limit the size of the hidden layers, which limits the amount of information that can flow through the network [37].

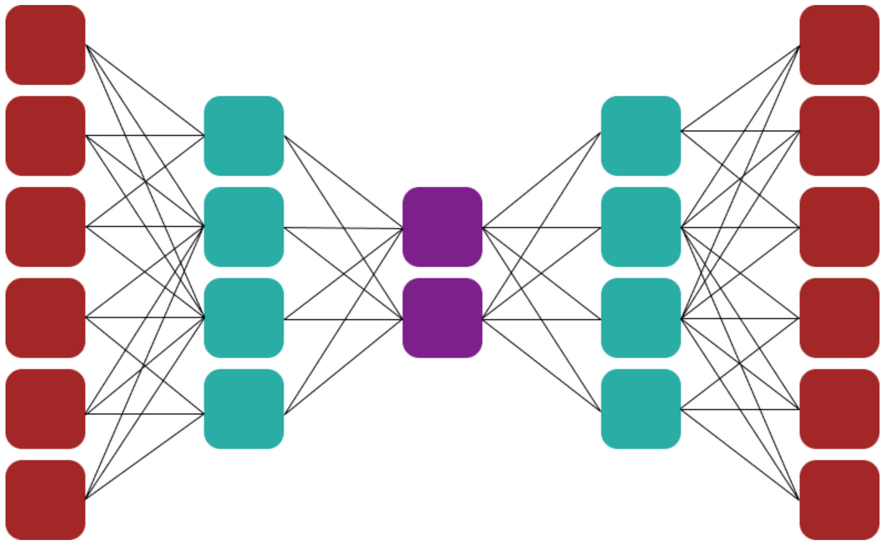


Figure 3.3: Visual representation of the architecture of an under-complete autoencoder

The important hyperparameters for this kind of model are the number of layers and the size of the bottleneck layer. Adding too many layers to the encoder and decoder is not helpful, even if the bottleneck layer is small, as this does not prevent the model from overfitting the data.

### 3.3.2 Variational Autoencoders

There are two major types of generative models: variational autoencoders (VAEs) and generative adversarial GANs. VAEs are autoencoders whose encodings in the latent space have been regularised during the training process that allows one to randomly sample from this latent space and generate new data points. This regularisation step is necessary as there is no guarantee that it will take place automatically since a normal autoencoder is trained with a singular goal in mind: to minimise the reconstruction error. So during training, the model will strive to achieve this, regardless of the structure of the latent space. The regularisation is implemented as follows:

- The input is encoded as a Gaussian distribution over the latent space with a different mean and variance for each input
- A point is sampled from this distribution, which is then passed through the decoder

- The reconstruction error is computed and back-propagated through the network

The loss function used here is the sum of two parts: the reconstruction error that compares the two images, and the latent loss, which compares the encoded distribution to a standard Gaussian distribution. The latent loss $\mathcal{L}$, which is the Kulback-Leibler divergence, is given by [38]

$$\mathcal{L} = -\frac{1}{2}\sum_{i=1}^{n}[1 + \log(\sigma_i^2) - \sigma_i^2 - \mu_i^2]$$

where $n$ is the number of dimensions and $\sigma_i, \mu_i$ are the $i^{th}$ component of the variance and mean. VAEs also make it possible to interpolate between images at an encoding level instead of at a pixel level.

## 3.4 Convolutions

In machine learning, a convolution is a process of taking a matrix (of size smaller than the input image) called the kernel and passing it over an input image to transform the image based on the kernel. In mathematical form, it looks as follows:

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k]$$

where $f, h$ are the image and kernel respectively and $m, n$ is the size of the image in pixels.

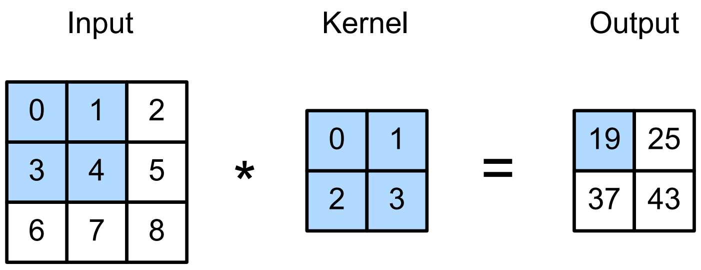


Figure 3.4: A visual representation of a convolution [39]

This function works for a single 2D matrix and outputs another 2D matrix. Each 2D matrix here represents a channel. With images, it is common for them to be single-channel (grey) or multi-channel (RGB). It is possible to perform a convolution with an arbitrary number of channels from layer to layer. Each such mapping has a different kernel associated with it. The above formula shows that the output size will be smaller than the input. To mitigate this problem, one may choose to add *padding* to the image, which is a border around the image, usually consisting of empty pixels (with a value of 0). The above formula is valid for a convolution operation with a *stride* of 1. This means that we move the kernel over the image one pixel at a time. However, it is also possible to shift it by multiple pixels. The number of pixels shifted per step is known as stride. If the padding is given by $p$, the stride is given by $s$, and the size of the kernel is given by $f$, then the dimensions of the output image can be calculated as (assuming both the input image and the kernel are square)

$$n_{out} = [\frac{n_{in} + 2p - f}{s} + 1]$$

A convolution operation performed on an image as part of a machine learning model is referred to as a convolutional layer. Consequently, convolutions and convolutional layers in model architectures are better suited to handling and processing image data than traditional dense, fully connected layers. This can be attributed to a few points below. We assume a similar dense layer would have the same number of neurons.
- A convolutional layer uses fewer parameters by forcing the input values to share parameters. There will be $m * n$ connections with a dense layer where $m, n$ are the input and output sizes. The weights are much smaller with a convolutional layer as each output is mapped to (usually) a small number of input pixels, depending on the kernel size. The output is formed only by the spatially nearest neighbours in the input.
- The relationship between the output and neighbouring input pixels is already taken into account with convolutional layers. Here, we assume that the neighbouring pixels in the input are related, and the pixels at the opposite ends of the image do not significantly influence each other. This assumption holds true in many visual analysis tasks. When this assumption can not reasonably be held to be true, it would make sense to use dense layers over convolutional ones. Due to the above reasons, we have decided to use convolutional layers in our models.

### 3.4.1 Pooling Layers

In order to maintain a performant model, it is computationally less expensive to decrease the size of an image by using a pooling layer, as opposed to using a strided convolution. It has empirically been found that using this strategy with a minimal loss in accuracy in several benchmarks. In this scenario, we decided that the performance uplift is worth the minor loss in evaluation metrics that may occur. [40]. A pooling layer, much like a convolution layer, uses a kernel with a size smaller than the input to perform a fixed operation on the image. There are no parameters to learn in this case. No padding is applied, so the resultant image is always smaller than the input. Two common types of pooling operations are:

- Average pooling: Returns the *average* of all the values in each patch of the image
- Max pooling: Instead of calculating the average, it simply returns the maximum value in the patch

With either method, pooling helps make the representation *translation invariant* to local changes. This means that if we shift the input to a layer by a small amount, most of the output values will not change. This makes the model more robust.

### 3.4.2 Upsampling Layers

In a similar fashion to pooling layers that perform downscaling of the input, there are upsampling layers whose output dimensions are larger than the input dimensions. Upsampling can not reconstruct any information that is lost during the downsampling process. In most cases, there are no parameters to be learned. However, more complicated techniques do exist ([41] [42]). The methods we tried are:

- Nearest neighbour: the unknown pixels are assigned the same value as the nearest pixel
- Bi-linear: the value of the pixel is interpolated using the nearest rows and columns of pixels. A visualisation of this algorithm is provided in Figure 3.5. The bi-linear interpolation is achieved by performing three linear interpolations. We first interpolate between $f(x_i, y_j), f(x_{i+1}, y_j)$ and $f(x_i, y_j), f(x_i, y_{j+1})$ to get $f(a, y_j)$ and $f(x_i, b)$ respectively. The order of this initial pairing is irrelevant. We then interpolate between these two points to finally arrive at $f(a, b)$.

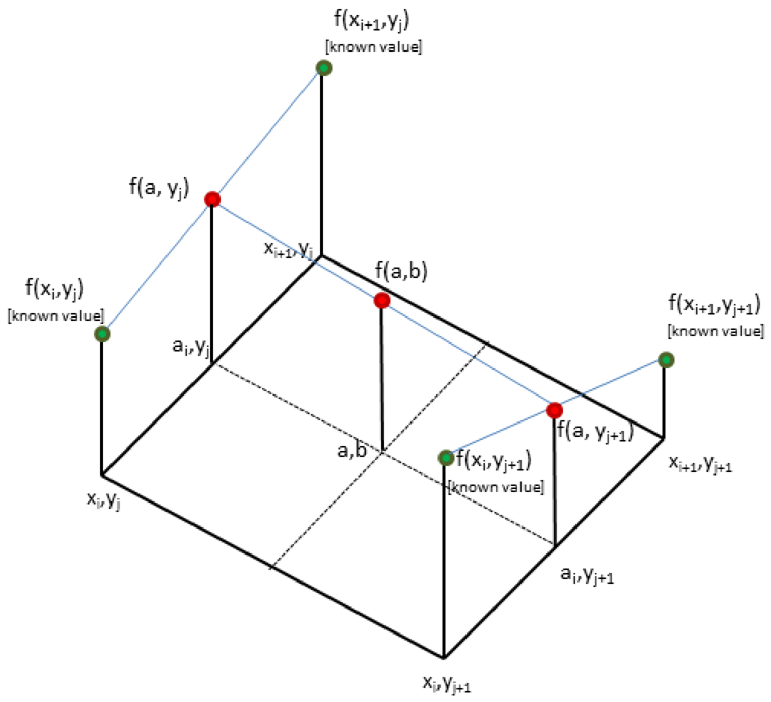


Figure 3.5: A pictorial representation of bi-linear interpolation for four points.

## 3.5 GANs

The idea behind GANs is simple in thought but tediously challenging to implement. We make two neural networks compete against each other and, like other industries and sectors in the real world, hope that the competition between them makes both of them excel at their respective roles. The two competing models are:

- Generator: It takes a random distribution (mostly a Gaussian) and outputs some target data, which is typically an image.

- Discriminator: It takes the output of the generator and compares it to a real data point from the training data. It outputs a value that tells us if the image is real or not.

Both networks have opposing goals: the generator tries to produce realistic images that can fool the discriminator, while the sole goal of the discriminator is to learn to differentiate between real images and the "fake" images provided to it by the generator. The GAN architecture formally is based on a zero-sum non-cooperative game. The game ends when it reaches a *Nash Equilibrium.* At this point, no player benefits from changing their current strategy. It has been shown that the only Nash equilibrium that GANs can reach is the one where the generator wins, and the discriminator is forced to take a random guess (50% each) if the image is real or fake. [43] Formally, the GAN objective function, which is a minimax function, can be written as

$$\min_{\theta} \max_{\phi} V(G_{\theta}, D_{\phi}) = \mathbb{E}_{\mathbf{x} \sim \mathbf{p}_{data}}[\log D_{\phi}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D_{\phi}(G_{\theta}(\mathbf{z})))]$$

Due to this, we cannot train the GAN in the normal way. Training takes place in two steps, first by training the discriminator and back-propagating its weights, and then by training the generator and back-propagating its weights.

### 3.5.1 Difficulties in Training GANs

In practice, many complications arise while trying to train GANs:

- Oscillations of parameters: the model parameters never converge Since both the generator and discriminator compete against each other, their parameters may continue to oscillate and never stabilise. Many factors can contribute to this behaviour, making GANs very sensitive to initial parameters.
- Mode collapse: the generator's outputs become less diverse This can happen if the generator gets good at producing a particular kind of image that can fool the discriminator. Since we reward this, it will continue to produce only those particular images and will not produce other kinds of images. Mode collapse is rarely to a single point, but partial collapses are common.
- Unbalanced generator and discriminator: Both networks can be unbalanced in the amount of information each can process. As an extreme example, consider a dense, fully connected network as the discriminator and a convolutional network as the generator. The discriminator is not powerful enough to provide good feedback data for the generator, and as a result, the generator will not learn to generate realistic images. However, currently, there is no metric to compare the power of two networks.

These obstacles make training GANs difficult, but the increase in performance is well worth the extra investment. We have tried a GAN based approach with the expectation of getting a much more performant model.

### 3.5.2 BiGANs

While GANs are able to learn the mappings from a simple distribution to a complex dataset and use this mapping to generate new samples of data, normal GANs have not been able to learn the inverse mapping from dataset to simple distribution. This means that to generate a data point from a GAN, one must provide the input in the form of a point on the simple distribution. This approach does not fit in with the overarching framework and approach of the current work. To circumvent this, we utilise Bidirectional Generative Adversarial Networks (BiGANs) as a way to learn the inverse mapping and, more importantly, the ability to retrieve a reconstruction error from a GAN based model [44].

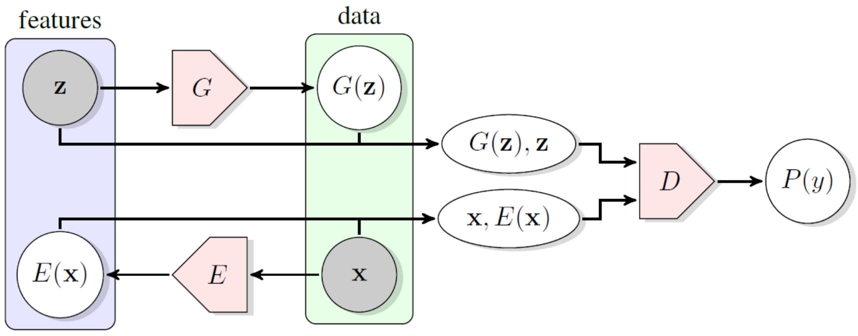


Figure 3.6: The model architecture for a BiGAN model [44]

The model contains an additional encoder $E$, which learns the inverse mapping. The BiGAN discriminator discriminates both in the data space and the latent vector space instead of doing it only in the data space. In this case, the objective function is given by:

$$\min_{\theta} \max_{\phi} V(G_\theta, D_\phi) = E_{x \sim p_X}[\log D_\phi(x)] + E_{x \sim p_G}[\log(1 - D_\phi(x))]$$

In order to successfully fool the discriminator, the encoder must successfully invert the generator output such that $E(G(z)) = z$. Using a fully trained version of this model, it is possible to provide an input image to the encoder, which is then passed to the generator to provide a new image that will be as close as possible to the input image. The two images can then be used to compute the reconstruction error.

# Chapter 4

# Results and Discussion

## 4.1 Results

### 4.1.1 sUAE Model

First, we provide the results of a previous autoencoder model that a different team at the company has previously worked on. This previous pipeline (henceforth referred to as the sUAE pipeline) has an undercomplete autoencoder architecture. It takes as input a $224 \times 224 \times 3$ image in the RGB colour space. The latent space comprises of vectors with dimensions $32 \times 32 \times 128$. It uses a relatively simple architecture, shown in Table 4.1. A subset of the results given by the model is provided in Table 4.2.

Table 4.1: Architecture for the prior sUAE model.

| Layer | Parameters |
|---|---|
| Conv2D | Filters = 32, BatchNorm, ReLU |
| MaxPooling2D | 2x2 |
| Conv2D | Filters = 64, BatchNorm, ReLU |
| MaxPooling2D | 2x2 |
| Conv2D | Filters = 128, BatchNorm, ReLU |
| MaxPooling2D | 2x2 |
| Conv2DTranspose | Filters = 128, BatchNorm, ReLU, Stride = 2 |
| Conv2DTranspose | Filters = 64, BatchNorm, ReLU, Stride = 2 |
| Conv2DTranspose | Filters = 32, BatchNorm, ReLU, Stride = 2 |

Table 4.2: A selection of results that showcase the sUAE model's performance.

| Dataset | Precision | Recall | F1 | AUROC | AUPRC |
|---|---|---|---|---|---|
| Abbey | 0.84 | 0.72 | 0.79 | 0.49 | 0.80 |
| Motorway | 0.70 | 0.89 | 0.79 | 0.73 | 0.74 |
| Tractor | 0.78 | 0.7 | 0.74 | 0.83 | 0.74 |

There are a few disadvantages with this implementation:

- The pipeline does not automatically select the best threshold for anomaly detection. This threshold has to be set manually for each dataset and for each iteration of the model. This makes it cumbersome to train and deploy the model on any dataset. This problem is further exacerbated if one wishes to do so for all eight datasets included here or wants to extend the analysis to other, new datasets. The manual entry of the threshold also introduces a new source of error that can significantly impact model performance, as the threshold selection is very arbitrary.
- A fundamental flaw in this pipeline is that a consequence of changing the threshold results in a change in the AUROC value as well. This should not happen, as once the model is trained, the reconstruction errors it provides for each image are independent of the threshold value that is decided much later on in the process. We can infer from this that there is a flaw in the analysis section of the pipeline.
- There is also much performance that can be gained by using Upsampling layers along with non-strided Conv2D layers instead of strided convolutions.
- The metrics are not within an acceptable range, as such low metrics have led to the pipeline missing anomaly events of intermediate length. For a practical model, we require that all anomalous events be detected, at least for a few frames of images at a bare minimum. Achieving perfect frame-level accuracy is clearly beneficial, but missing entire anomalous events is not acceptable, as there are multiple data points available that belong to each anomaly.

We have done the work described herein as an effort to improve the sUAE pipeline mentioned above, making a new pipeline that is more robust, performant, and scores higher in the relevant evaluation metrics.

### 4.1.2 BiGAN model

The BiGAN approach is appealing because the research work shown in Section 2.1 leverages various kinds of GANs for incredible accuracy in test datasets. We have tried a similar approach with the expectation of a high level of accuracy at the cost of some performance. We present the architecture of this network in Tables 4.3 and 4.4.

Table 4.3: Generator architecture for the BiGAN model.

| Generator Layers | Parameters |
|---|---|
| Conv2DTranspose | Filters = 256, Kernel = 4 BatchNorm, ReLU, Stride = 3 |
| Conv2DTranspose | Filters = 128, Kernel = 4 BatchNorm, ReLU, Stride = 3 |
| Conv2DTranspose | Filters = 64, Kernel = 4 BatchNorm, ReLU, Stride = 4 |
| Conv2DTranspose | Filters = 1, Kernel = 4 BatchNorm, ReLU, Stride = 4 |

Table 4.4: Encoder architecture for the BiGAN model.

| Encoder Layers | Parameters |
|---|---|
| Conv2D | Filters = 64, Kernel = 4, BatchNorm, LeakyReLU, Stride = 4 |
| Conv2D | Filters = 128, Kernel = 4, BatchNorm, LeakyReLU, Stride = 4 |
| Conv2D | Filters = 256, Kernel = 4, BatchNorm, LeakyReLU, Stride = 3 |
| Conv2D | Filters = 50, Kernel = 4, LeakyReLU, Stride = 3 |

The discriminator follows the same architecture as the encoder, except that it is in the reverse order. The underlying space that we are sampling from has 50 dimensions, as shown by the number of filters in the last layer of the encoder. This model was trained for a maximum of 800 epochs.

The results for this method are shown in Table 4.5

Table 4.5: Results from the BiGAN model.

| Dataset | Precision | Recall | F1 | AUROC | AUPRC |
|---|---|---|---|---|---|
| abbey | 0.85 | 0.3 | 0.44 | 0.57 | 0.48 |
| beach | 0.9 | 0.53 | 0.67 | 0.74 | 0.85 |
| castro | 0.72 | 0.42 | 0.53 | 0.64 | 0.65 |
| meteor night | 0.31 | 0.61 | 0.41 | 0.56 | 0.37 |
| motorway | 0.46 | 0.79 | 0.58 | 0.68 | 0.63 |
| otter | 0.89 | 0.94 | 0.91 | 0.99 | 0.99 |
| tractor | 0.56 | 0.56 | 0.56 | 0.6 | 0.98 |
| volcano | 0.93 | 0.95 | 0.94 | 0.89 | 0.78 |
| Average | 0.7 | 0.64 | 0.63 | 0.71 | 0.72 |

### 4.1.3 VAE Model

The VAE model takes the input as $144 \times 144$ images and produces the same size output. We have utilised:

- Convolutional layers with batch normalisation and ReLU as the activation function, with max pooling layers and bi-linear up-sampling layers to change the dimensions of the image in between layers.
- Dense layers for moving between the latent space and data space to comply with the VAE architecture
- He initialisation for all layers
- A sigmoid activation for the final layer to achieve pixel outputs between (0,1)
- A maximum number of epochs of 100. However, by implementing early stopping, this maximum was rarely reached. The model architecture is given in Table 4.6. The results from this model are given in Table 4.7.

Table 4.6: Architecture for the VAE model.

| Layer | Parameters |
|---|---|
| Conv2D | Filters = 16, BatchNorm, ReLU |
| MaxPooling2D | 2x2 |
| Conv2D | Filters = 32, BatchNorm, ReLU |
| MaxPooling2D | 2x2 |
| Conv2D | Filters = 64, BatchNorm, ReLU |
| MaxPooling2D | 2x2 |
| Conv2D | Filters = 128, BatchNorm, ReLU |
| MaxPooling2D | 2x2 |
| Flatten | - |
| Collection of Dense Layers | Size = 60, ReLU |
| Conv2D | Filters = 128, BatchNorm, ReLU |
| UpSampling2D | 2x2 |
| Conv2D | Filters = 64, BatchNorm, ReLU |
| UpSampling2D | 2x2 |
| Conv2D | Filters = 32, BatchNorm, ReLU |
| UpSampling2D | 2x2 |
| Conv2D | Filters = 16, BatchNorm, ReLU |
| UpSampling2D | 2x2 |
| Conv2D | Filters = 1, BatchNorm, Sigmoid |

Table 4.7: Results from the VAE model.

| Dataset | Precision | Recall | F1 | AUROC | AUPRC |
|---|---|---|---|---|---|
| abbey | 0.98 | 0.7 | 0.82 | 0.82 | 0.96 |
| beach | 0.75 | 0.8 | 0.77 | 0.85 | 0.79 |
| castro | 0.67 | 0.56 | 0.61 | 0.72 | 0.78 |
| meteor night | 0.78 | 0.88 | 0.83 | 0.79 | 0.82 |
| motorway | 0.8 | 0.75 | 0.77 | 0.8 | 0.73 |
| otter | 0.99 | 0.97 | 0.98 | 0.99 | 0.99 |
| tractor | 0.9 | 0.88 | 0.89 | 0.89 | 0.95 |
| volcano | 0.97 | 0.98 | 0.97 | 0.95 | 0.94 |
| Average | 0.86 | 0.82 | 0.83 | 0.85 | 0.87 |

### 4.1.4 AE Model

Finally, we present the AE model architecture. This model follows precisely the same architecture as that described in Table 4.6, with the exception of the Flatten and Dense layers, which have been removed. This means that the latent space representation is no longer regularised, and the model is not guaranteed to be able to generate new images from a random point in the latent space. The other properties of this model are also identical, as described in the previous Section 4.1.3. The results from this model are given in Table 4.8.

Table 4.8: Results from the AE model.

| Dataset | Precision | Recall | F1 | AUROC | AUPRC |
|---|---:|---:|---:|---:|---:|
| abbey | 0.88 | 0.88 | 0.88 | 0.62 | 0.87 |
| beach | 0.81 | 0.82 | 0.81 | 0.92 | 0.85 |
| castro | 0.79 | 0.99 | 0.88 | 0.92 | 0.66 |
| meteor night | 0.89 | 1 | 0.94 | 0.99 | 0.97 |
| motorway | 0.74 | 0.92 | 0.82 | 0.83 | 0.84 |
| otter | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 |
| tractor | 0.93 | 0.95 | 0.94 | 0.99 | 0.98 |
| volcano | 1 | 1 | 1 | 0.99 | 1 |
| Average | 0.88 | 0.94 | 0.91 | 0.92 | 0.91 |

### 4.1.5 Performance

We also provide a comparison of the model run times on the various datasets in Figures 4.1, 4.2 and 4.3. Models were run both on a normal laptop CPU (i7-8750H, released in 2017) and on a Google Colab GPU (Tesla K80, released in 2014), which can be run on any computer that has access to a browser. Both models can be run on either processing unit, but a GPU enables us to accelerate model development greatly.

We also provide the AUROC, AUPRC and frame-level graphs for the AE based model, as it is the model with the best performance. For the sake of brevity, we discuss two datasets. The other graphs corresponding to this model are available in the Appendix.

#### 4.1.5.1 Violin Plots

We have provided the performance results using violin plots. As violin plots are relatively uncommon, we have included this section to give a brief overview of these plots.

A violin plot is a hybrid of a traditional box plot and a kernel density plot. A box plot can be misleading, as various different distributions of data can produce similar, if not the same, box plots. The addition of the density plot helps visualise the distribution of the data points. A wider density plot indicates a higher frequency of data points, and vice-versa.

As is standard, the box within each density plot represents the interquartile range (IQR). The dashed line and the solid line represent the mean and mode respectively. Next to the violin plot, we have also represented the individual data points. It is important to note that the horizontal spread of the data points is done only for visualisation purposes and has no other meaning.

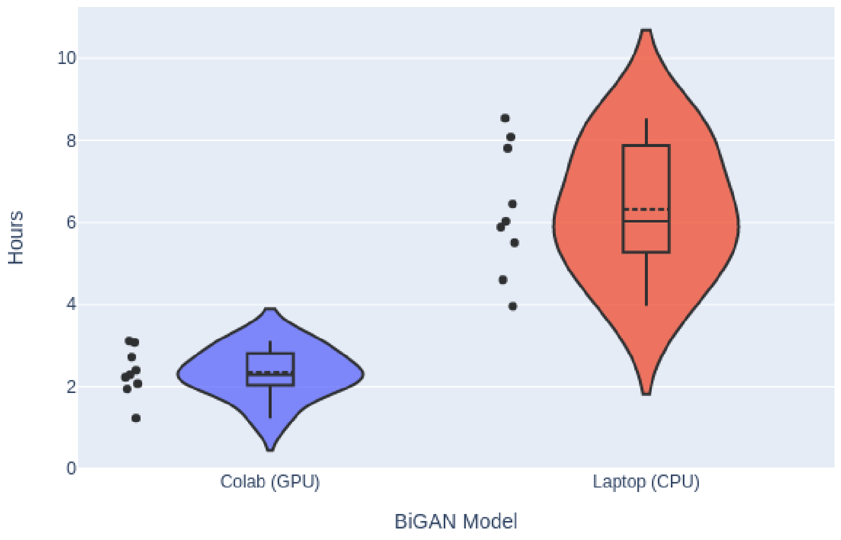


Figure 4.1: Graph comparing the run times on the GPU and CPU respectively for the BiGAN model.

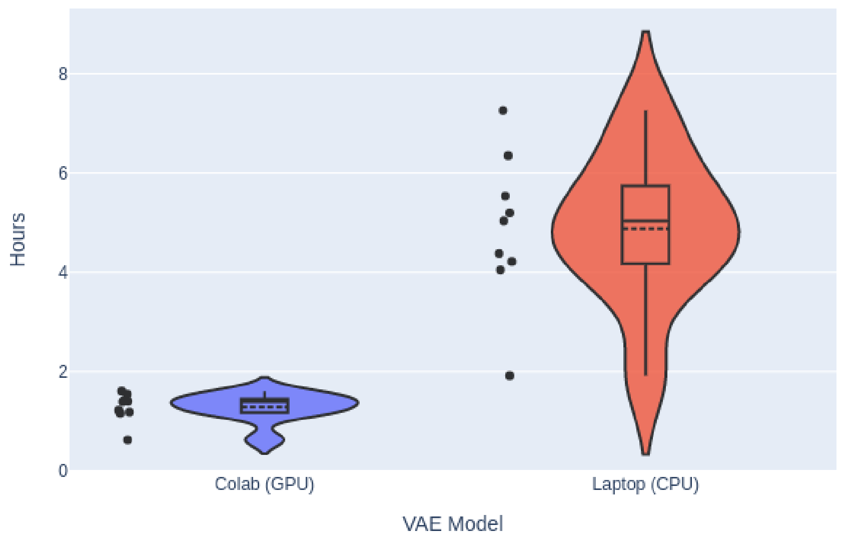


Figure 4.2: Graph comparing the run times on the GPU and CPU respectively for the VAE model.

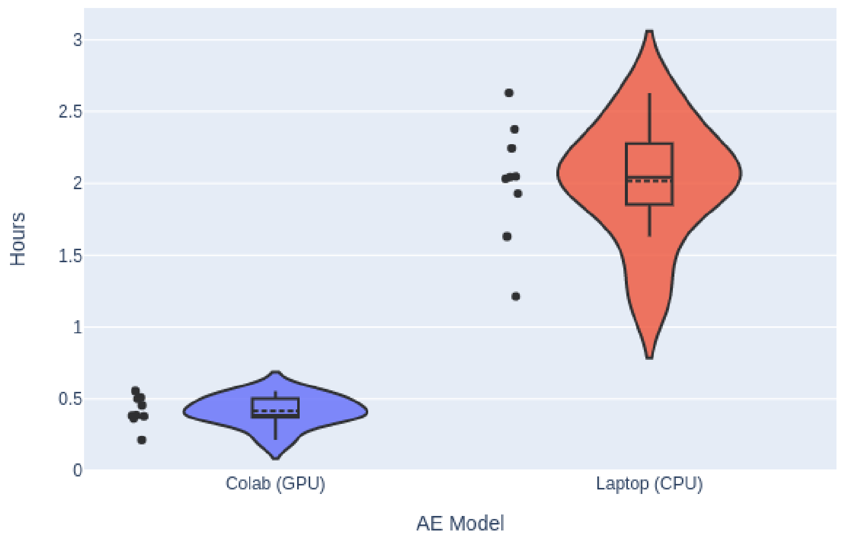


Figure 4.3: Graph comparing the run times on the GPU and CPU respectively for the AE model.

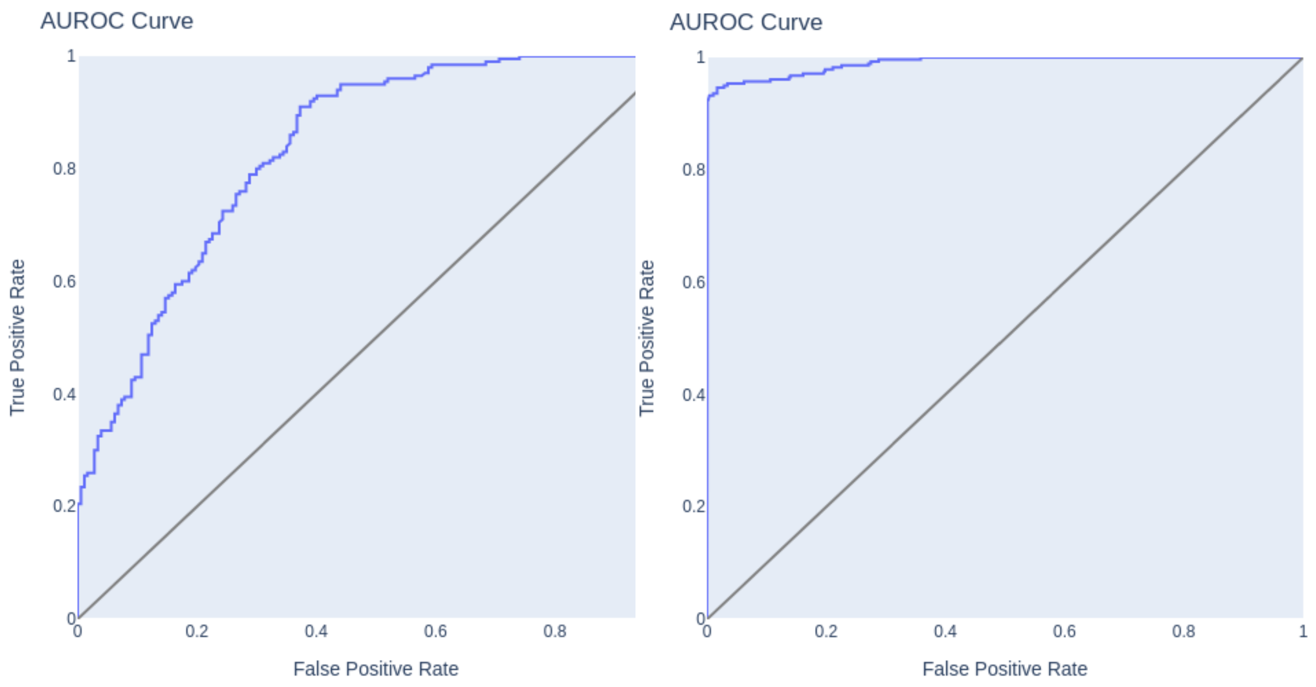


Figure 4.4: AUROC curves for motorway and tractor respectively

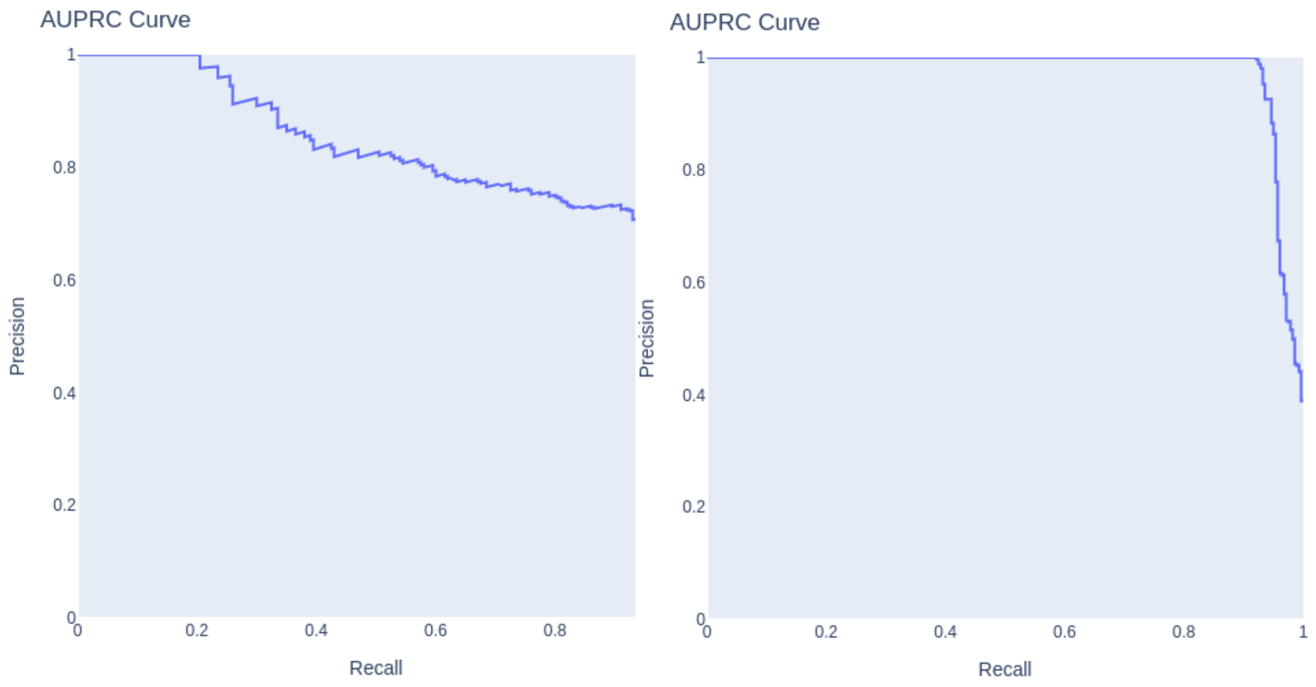
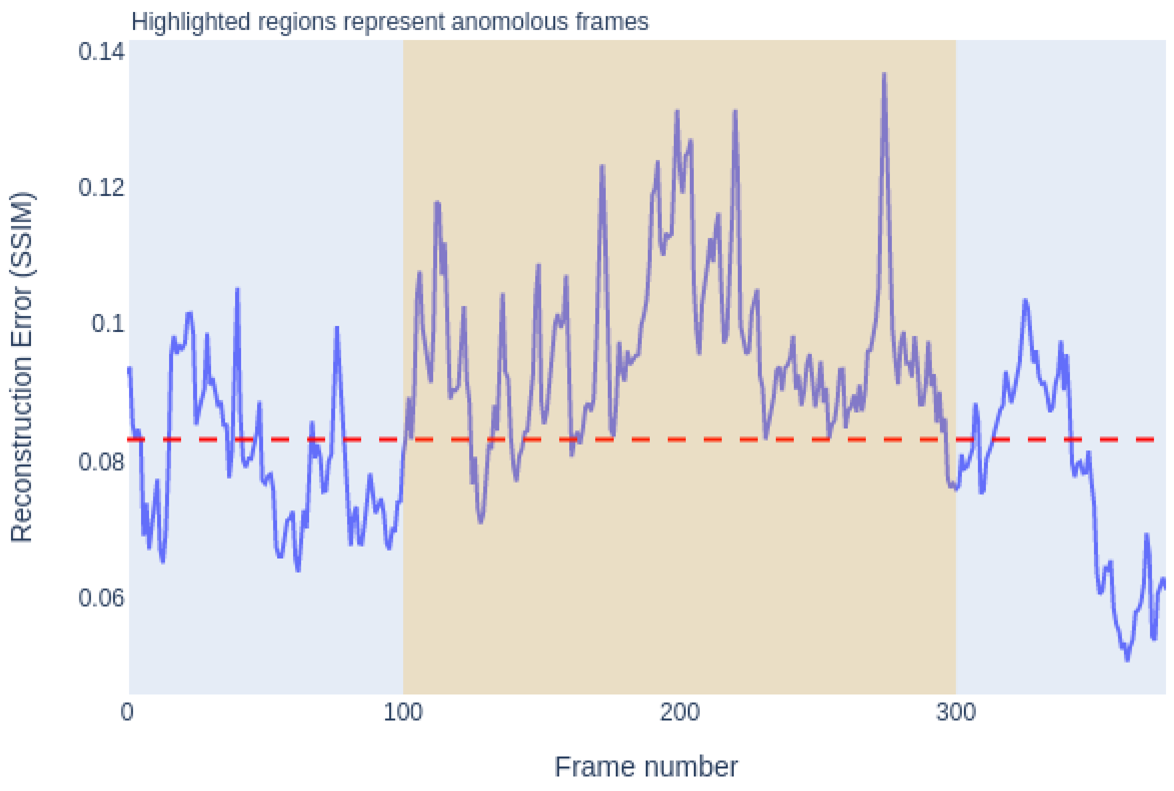


Figure 4.5: AUPRC curves for motorway and tractor respectively



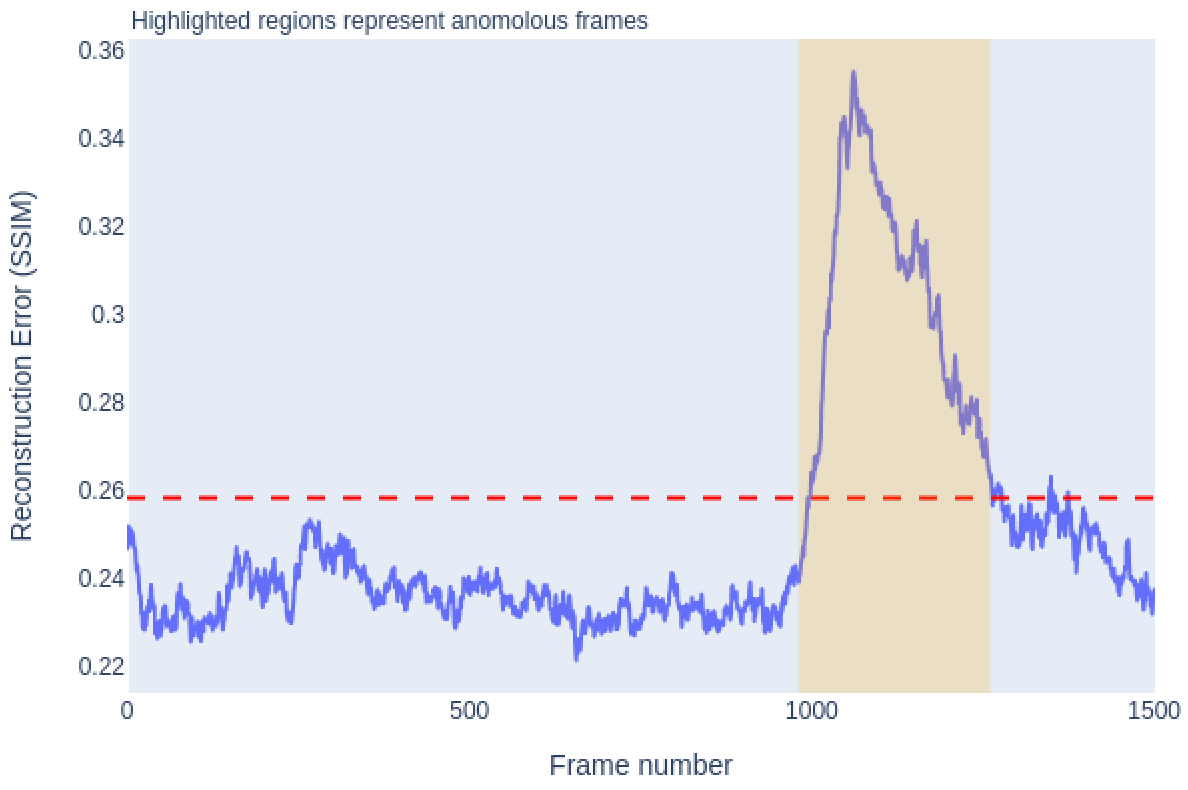


Figure 4.6: Frame level predictions for motorway and tractor respectively. The red line represents the threshold chosen using Youden's J(Section 2.5.7). The lightly yellow coloured background portion represents the frames that are anomalous.

## 4.2 Discussion

## 4.3 BiGAN review

Firstly, we would like to discuss the results of the BiGAN model. We tried this model with the expectation that the trained discriminator would be able to differentiate between the normal images and anomalous images as the encoder would poorly reconstruct the anomalous images in the distribution space, and the subsequent reconstruction by the generator would likewise be poor. Hence, the pairs $(z, G(z)), (x, E(x))$ would not not be able to fool the discriminator. However, this is not the case in practice, as evidenced by the results in Table 4.5. Additionally, we also observed that the results from the model would fluctuate from epoch to epoch. However, we have verified that mode collapse did not occur, as the model was able to output many kinds of images from the training data set.

We have tried to eliminate the common difficulties that prevail in GAN training, but more work can be done in getting results with this method.

- We have added additional augmented training images, using minor translations and rotations to generate additional images to increase the training data set size multi-fold.
- We also conducted hyperparameter searches to find optimal parameters for the model.

Our hypothesis for why this model did not work can be summarised in the points below:

- The variance of the results across epochs suggests that the model parameters did not converge and continued to oscillate.
- Both the discriminator and the generator were not well balanced enough with respect to each other for the Nash Equilibrium to be reached.
- GANs are known to be very sensitive to hyperparameters, and perhaps the hyperparameter search we conducted was not thorough enough.
- Even after data augmentation, the training data set was not large enough for the GAN to adequately learn all the features of the data set.

Future work in this direction could attempt to remedy the problems described above. Other GAN architectures could also be tested, with loss functions other than the function described in Section 3.5. Another important consideration is the fact that we needed to develop a GAN model that could work on all eight data sets simultaneously. In many works previously (discussed in Section 2.1), a GAN model has been developed for each individual data set. It may be that our approach is not compatible with the GAN architecture.

Due to the time constraints of the project, we decided to pursue other architectures instead of diving deeper into the GAN based architectures, which have a reputation of being difficult to train. This difficulty, coupled with the fact that we have to find a GAN architecture that needs to work across eight datasets without individual tweaking, were crucial factors in our decision.

## 4.4 VAE Review

The VAE based model performs significantly better than the BiGAN model. We have found that this model does not miss any anomalous events at all. It is also able to generate new images from the data set it is trained on. However, the AE model manages to outperform this model. As the only difference between these two models is the normalisation of the latent space, we hypothesise that finer tuning of this part of the model may yield comparable, if not better, results than the AE model.

## 4.5 AE Review

The AE based model is the model that we have seen the best results with. This model performs much better than all of the previous models. The frame level graphs show how implementing a frame buffer system has positively impacted the results. The data plotted in Figure 4.6 is the raw anomaly score for each frame. The frame buffer ensured that the short spikes above and below the red threshold line did not have a negative impact on the performance metrics of the model.

With an average F-score of 0.91, we can say with certainty that this model performs well across all eight data sets and will perform very well on new data sets on the field as well. The drop in metrics arises from the fact that, in most cases, at a frame level, there is an unclear boundary between normal events and anomalous events. We have achieved what we set out to accomplish with this in context. The evaluation metrics of the model are at a highly satisfactory level, especially for the model's intended use cases in surveillance and security.

### 4.5.1 Avenues for Improvement

There do remain some areas in which model performance might be improved. These points are valid for both the AE and VAE models, as they are based on the same underlying architecture.

- The current frame buffer system we have implemented is a flat number cut-off. A better system would have a more reactive approach that would take more factors such as the frequency of the spikes into account and would consider the reconstruction error of the frames as well.
- We have only explored the use of undercomplete autoencoders to constrain the network. Another approach, known as sparse autoencoders, is also possible [45]. This approach constrains the model by penalising the activation of hidden units. We have not explored the effects of such an approach in this work.
- As seen in Sections 3.4.1 and 3.4.2, we have used the most common approaches used in pooling and

upsampling layers. There are more methods of image scaling possible, such as polynomial scaling, vectorisation, Fourier transforms, Lanczos resampling, etc. [46]

- We have used greyscale images to obtain a meaningful increase in performance. The usefulness of full-colour images could also be explored. Additionally, there are different ways to convert a colour image to a greyscale image. It has been shown that the method used for this conversion can significantly impact the final model performance [47]. These additional methods, such as contrast boosting or other filters, can also be explored to see if they impact this specific model's performance in any way.

## 4.6 Model Performance

The BiGAN model, as expected, takes the most extended amount of time to train. This is because GANs must be trained for a large number of epochs to achieve satisfactory results. In this case, the results are not worth the large amount of time any such GAN based models take.

The performance of the autoencoder based models is more appealing than the GAN based model. The AE model, which we have chosen fit for deployment in the real world, can be trained quickly on most modern laptops offline. Time taken can be significantly reduced by connecting online to a Colab GPU. The low time taken also makes it a good fit for iterative development of other pipelines based on this model. The VAE model does take significantly longer due to the inclusion of additional dense layers, so an increase in model training time is expected. While not fit for deployment, we envision such a model being useful for development purposes where additional training images need to be generated. In such situations, where GPUs are more accessible, this model can be used with significant impact.

## 4.7 Impact

We believe that the model we have built throughout this project has considerable implications in the security and monitoring space.

TensorFlow [48], the library upon which the model is built (using Python [49]), allows us to deploy this model to either the cloud or to any individual's computer. Unlike training the model, which is a time-consuming process as the weights need to be constantly updated and re-evaluated, deploying and running the model with fixed parameters is something that almost any computer can do in close to real-time. The ubiquity of computing power and the ease of access to such devices means that once a model has been trained, frames coming from multiple cameras can be analysed as they are being received, using a relatively inexpensive set-up.

This anomaly detection system can be deployed and used by anyone, from small shopkeepers who need to ensure their store's safety to companies that need to monitor their warehouses to governments that need to stay vigilant on their streets. CCTV cameras are already present in an overwhelmingly large quantity throughout our cities ([3] [4]). This system allows us to effectively utilise the infrastructure that has already been laid down to enable intelligent and efficient detection of anomalous events that will bolster security at a minimal expense to those in charge of these premises.

# Chapter 5

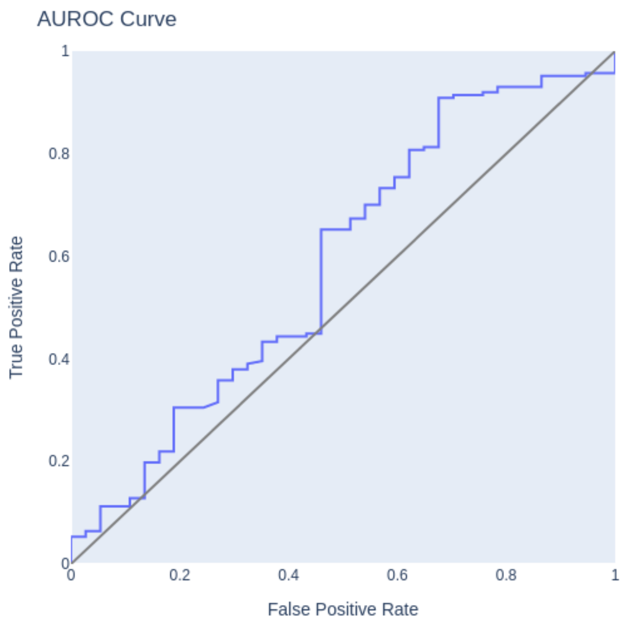
# Conclusion

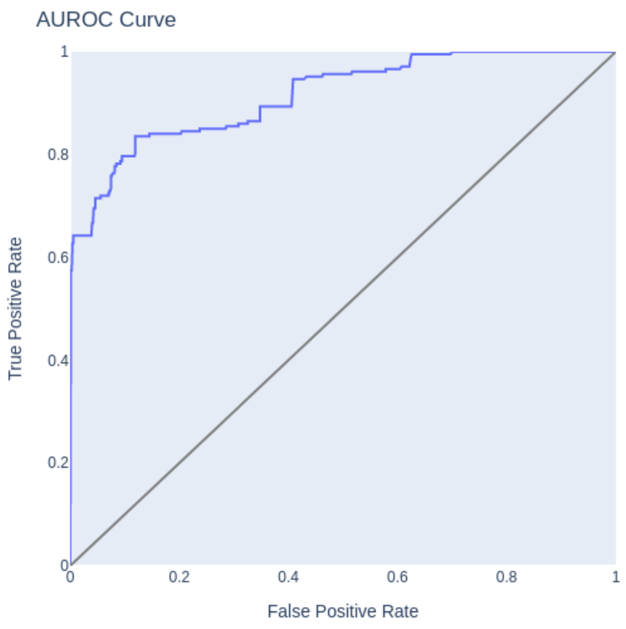
In summary, through the work done in this thesis, we have tested two prevalent machine learning architectures, both of which are unsupervised, to develop an automatic anomaly detection system that uses the reconstruction error of images as the primary metric to perform binary classification of the images into anomalous and non-anomalous (normal) images. The GAN based method was not as successful as we initially hypothesised, which we attribute to the myriad of problems that GANs face during training. Our second approach, based on autoencoders, resulted in a much more performant model that achieved excellent metric scores and easily beat the previous sUAE model without any significant drawbacks. The high performance of our AE model makes it relatively easy to train and even more easier to deploy on the field. This model has the potential to drastically increase the effectiveness of security and surveillance systems by lowering the barrier of entry and making such systems accessible to everyone.

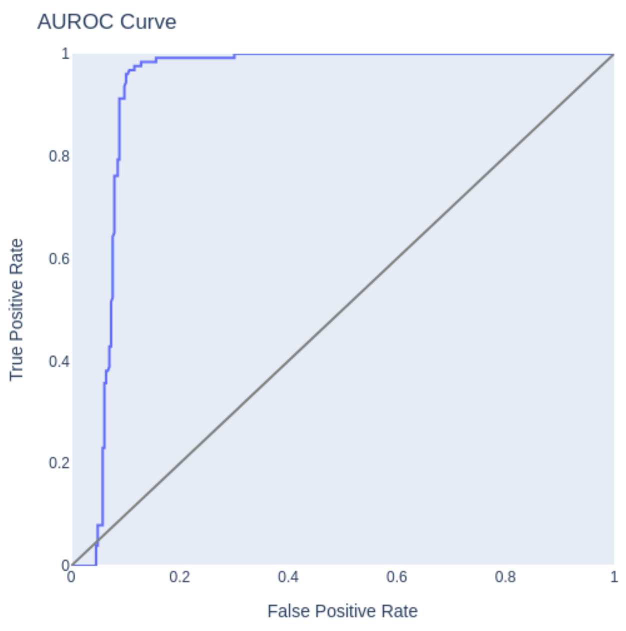
# Appendix A

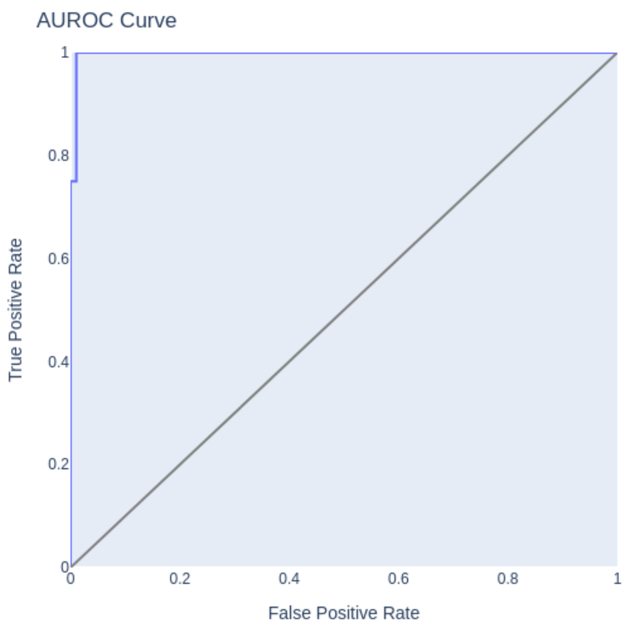
# List of Performance Figures

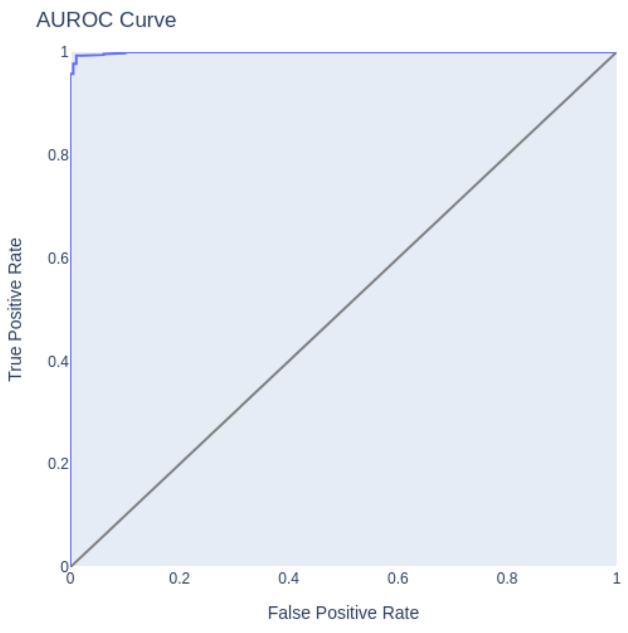
The first set of graphs correspond to the AUROC graphs for the datasets abbey, beach, castro, meteornight, otter and volcano respectively. The second set corresponds to the AUPRC graphs, while the final set corresponds to the frame level predicitions made by each model, following the previously mentioned order. The graphs for motorway and tractor have already been provided in Section 4.1.

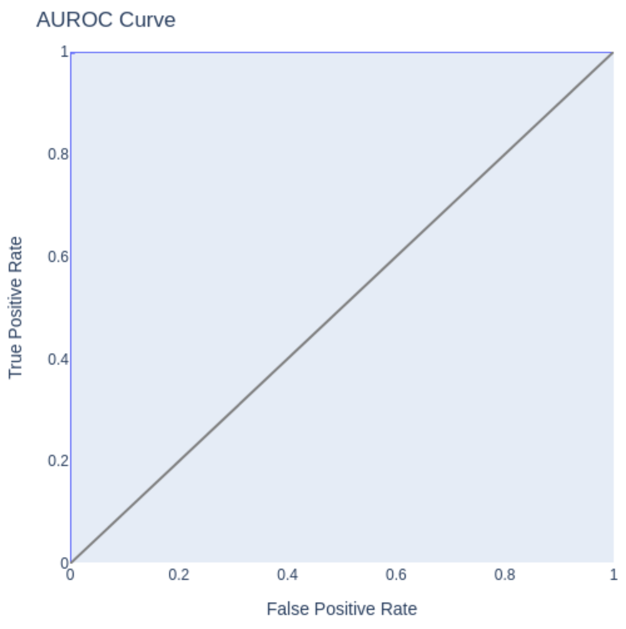


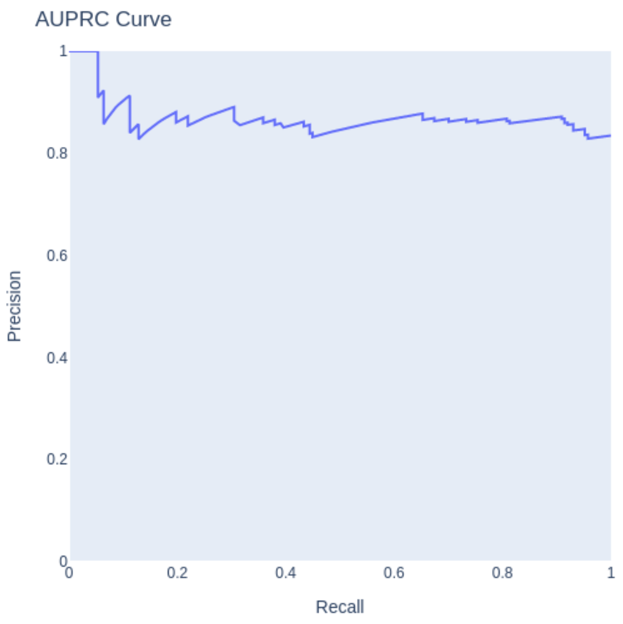


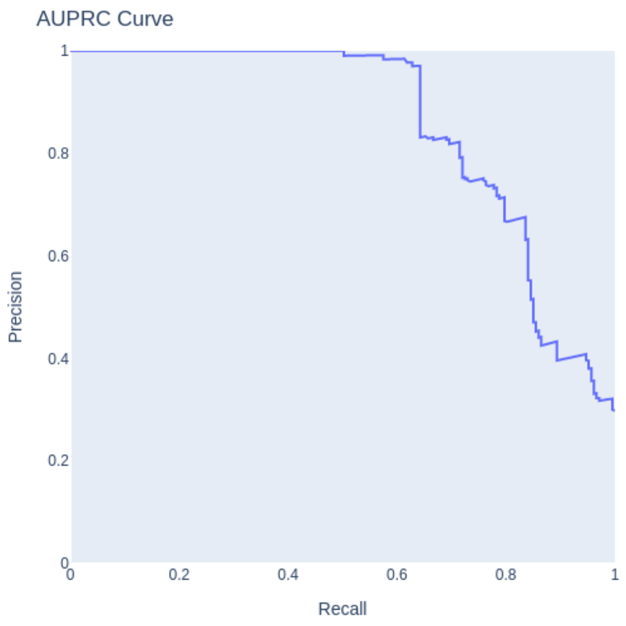

AUROC Curve
True Positive Rate
False Positive Rate
0
0.2
0.4
0.6
0.8
1

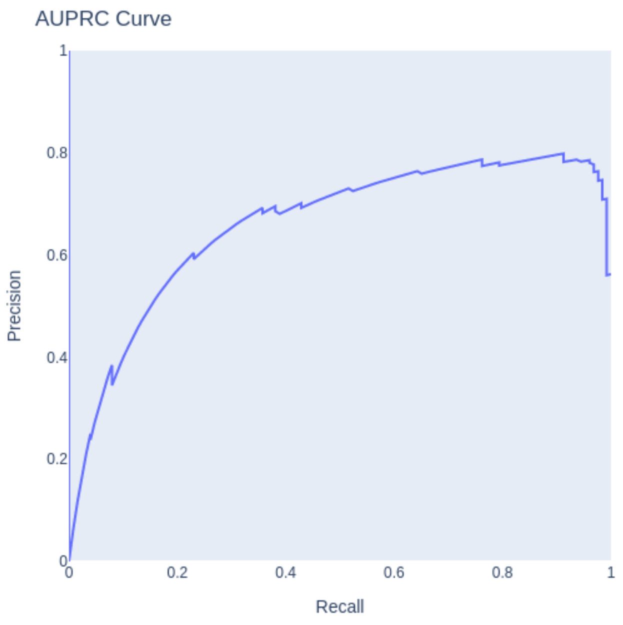

AUROC Curve
True Positive Rate
False Positive Rate
0
0.2
0.4
0.6
0.8
1

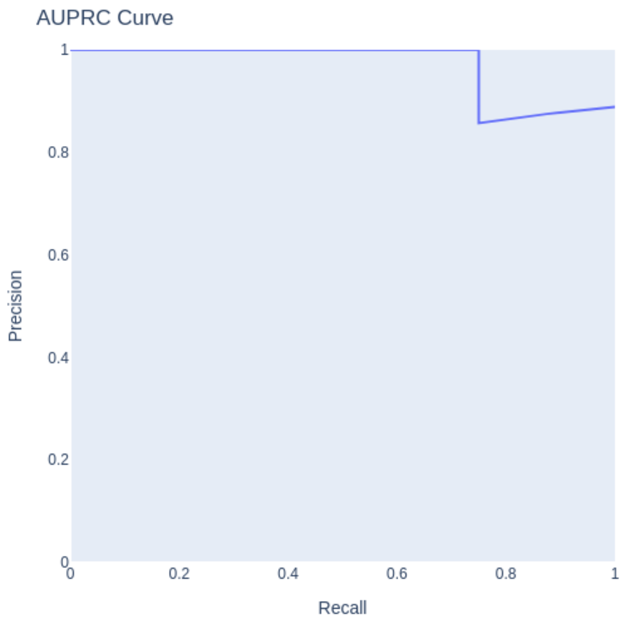

AUROC Curve
True Positive Rate
False Positive Rate
0
0.2
0.4
0.6
0.8
1

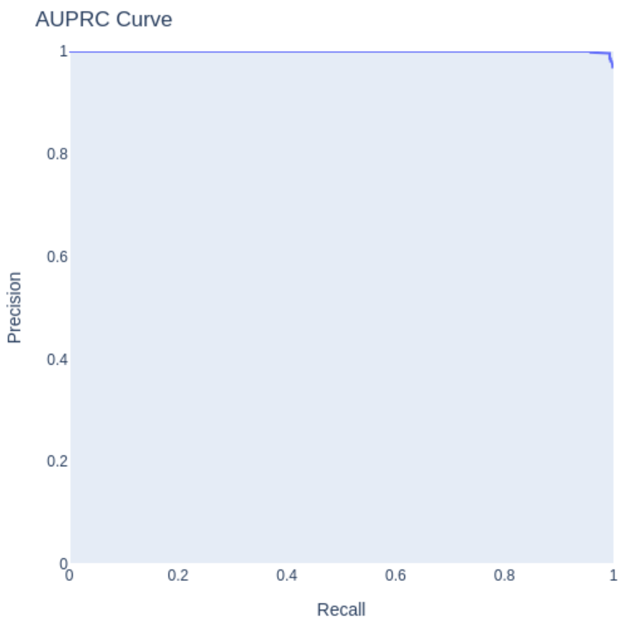

AUROC Curve
True Positive Rate
False Positive Rate
0
0.2
0.4
0.6
0.8
1

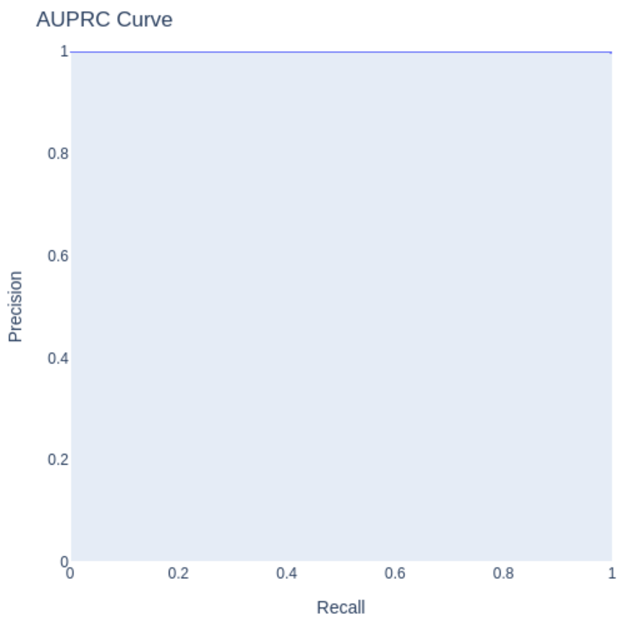

AUPRC Curve
Precision
Recall
0
0.2
0.4
0.6
0.8
1

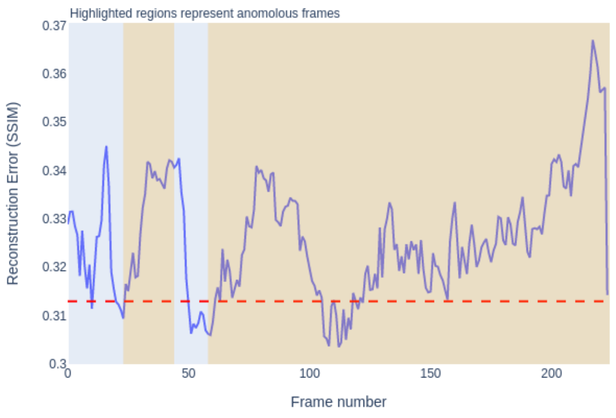

AUPRC Curve
Precision
Recall
0
0.2
0.4
0.6
0.8
1

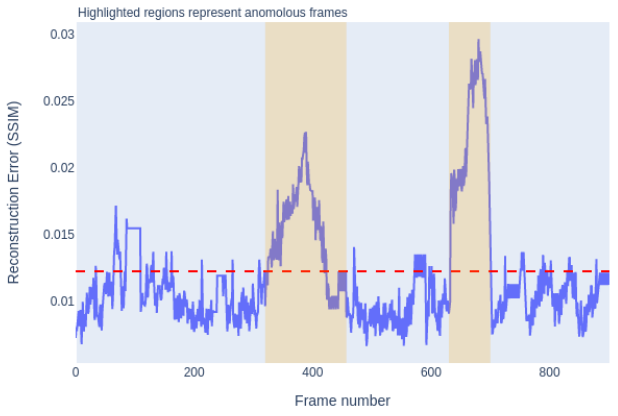

AUPRC Curve
Precision
Recall
0
0.2
0.4
0.6
0.8
1

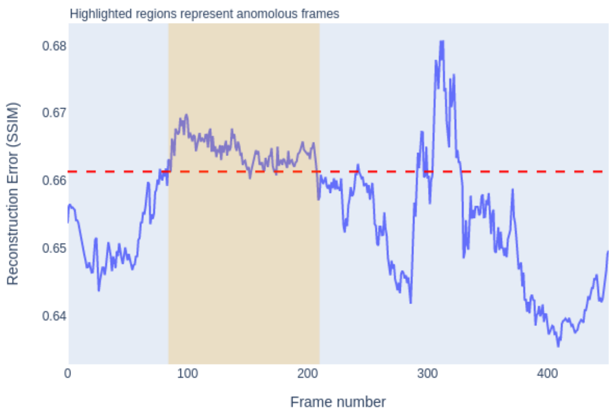

AUPRC Curve
Precision
Recall
0
0.2
0.4
0.6
0.8
1

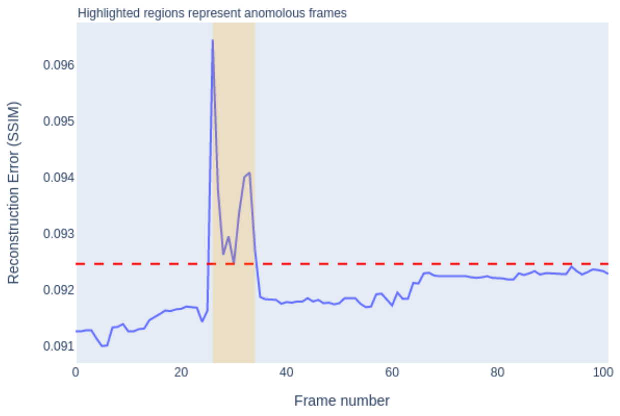

AUPRC Curve
Precision
Recall
0
0.2
0.4
0.6
0.8
1

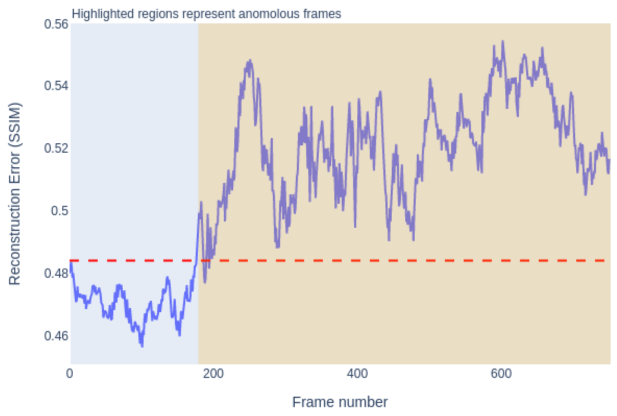

AUPRC Curve
Precision
Recall
0
0.2
0.4
0.6
0.8
1

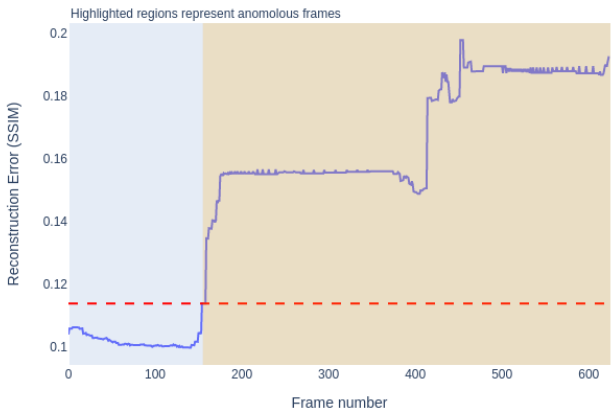

Highlighted regions represent anomolous frames
Reconstruction Error (SSIM)
Frame number
0.37
0.36
0.35
0.34
0.33
0.32
0.31
0.3
0
50
100
150
200


Highlighted regions represent anomolous frames
Reconstruction Error (SSIM)
Frame number
0.03
0.025
0.02
0.015
0.01
0
200
400
600
800


Highlighted regions represent anomolous frames
Reconstruction Error (SSIM)
Frame number
0.68
0.67
0.66
0.65
0.64
0
100
200
300
400


Highlighted regions represent anomolous frames
Reconstruction Error (SSIM)
Frame number
0.096
0.095
0.094
0.093
0.092
0.091
0
20
40
60
80
100


Highlighted regions represent anomolous frames
Reconstruction Error (SSIM)
Frame number
0.56
0.54
0.52
0.5
0.48
0.46
0
200
400
600


Highlighted regions represent anomolous frames
Reconstruction Error (SSIM)
Frame number
0.2
0.18
0.16
0.14
0.12
0.1
0
100
200
300
400
500
600

# References

1. Glinsky, A.: Theremin : Ether music and espionage. University of Illinois Press (2000)

2. Zhu, S., Chen, C., Sultani, W.: Video anomaly detection for smart surveillance. CoRR. (2020)

3. Markit, I.: Video surveillance: How technology and the cloud is disrupting the market, https://cdn.ihs.com/www/pdf/IHS-Markit-Technology-Video-surveillance.pdf, (2019)

4. Lin, L., Purnell, N.: A world with a billion cameras watching you is just around the corner, https://www.wsj.com/articles/a-billion-surveillance-cameras-forecast-to-be-watching within-two-years-11575565402, (2019)

5. Kiran, B., Thomas, D., Parakkal, R.: An overview of deep learning based methods for unsupervised and semi-supervised anomaly detection in videos. Journal of Imaging. 4, 36 (2018). https://doi.org/10.3390/jimaging4020036

6. Candes, E.J., Li, X., Ma, Y., Wright, J.: Robust principal component analysis?, https://arxiv.org/abs/0912.3599, (2022)

7. Chalapathy, R., Menon, A.K., Chawla, S.: Robust, deep and inductive anomaly detection. arXiv.org. (2017). https://doi.org/10.48550/arXiv.1704.06743

8. Support vector method for novelty detection | proceedings of the 12th international conference on neural information processing systems, https://dl.acm.org/doi/10.5555/3009657.3009740, (2022)

9. Erfani, S., Baktashmotlagh, M., Rajasegarar, S., Karunasekera, S., Leckie, C.: R1SVM: A randomised nonlinear approach to large-scale anomaly detection, https://people.eng.unimelb.edu.au/sarahme/papers/Erfani2015R1SVM.pdf

10. Nguyen, M.-N., Vien, N.A.: Scalable and interpretable one-class SVMs with deep learning and random fourier features. CoRR. (2018)

11. Liu, F.T., Ting, K.M., Zhou, Z.-H.: Isolation forest. 2008 Eighth IEEE International Conference on Data Mining. (2008). https://doi.org/10.1109/icdm.2008.17

12. Xiong, L., Póczos, B., Schneider, J.: Group anomaly detection using flexible genre models. Advances in Neural Information Processing Systems. 24, (2022)

13. Zimek, A., Schubert, E., Kriegel, H.-P.: A survey on unsupervised outlier detection in high-dimensional numerical data. Statistical Analysis and Data Mining. 5, 363–387 (2012). https://doi.org/10.1002/sam.11161

14. Barnett, V., Tolewis: Outliers in statistical data. Wiley (1994)

15. Ravanbakhsh, M., Nabi, M., Mousavi, H., Sangineto, E., Sebe, N.: Plug-and-play CNN for crowd motion analysis: An application in abnormal event detection. CoRR. (2016)

16. Ribeiro, M., Lazzaretti, A.E., Lopes, H.S.: A study of deep convolutional auto-encoders for anomaly detection in videos. Pattern Recognition Letters. 105, 13–22 (2018). https://doi.org/10.1016/j.patrec.2017.07.016

17. Géron, A.: Hands-on machine learning with scikit-learn, keras, and TensorFlow. O'Reilly

18. Xie, J., Girshick, R., Farhadi, A.: Unsupervised deep embedding for clustering analysis, https://arxiv.org/abs/1511.06335, (2015)

19. Pang, G., Shen, C., Cao, L., Anton: Deep learning for anomaly detection: A review. CoRR. (2020)

20. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems. 25, (2012)

21. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv.org. (2014). https://doi.org/10.48550/arXiv.1409.1556

22. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. arXiv.org. (2015). https://doi.org/10.48550/arXiv.1512.03385

23. Horn, B.K.P., Schunck, B.G.: Determining optical flow. Artificial Intelligence. 17, 185–203 (1981). https://doi.org/10.1016/0004-3702(81)90024-2

24. Ravanbakhsh, M., Nabi, M., Sangineto, E., Marcenaro, L., Regazzoni, C.S., Sebe, N.: Abnormal event detection in videos using generative adversarial nets. CoRR. (2017)

25. Schlegl, T., Seeböck, P., Waldstein, S.M., Schmidt-Erfurth, U., Langs, G.: Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. arXiv.org. (2017). https://doi.org/10.48550/arXiv.1703.05921

26. Mitchell, T.M.: Machine learning. MacGraw-Hill (1997)

27. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research. 13, 281–305 (2012)

28. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature. 323, 533–536 (1986). https://doi.org/10.1038/323533a0

29. Guo, J.: AI notes: Initializing neural networks - deeplearning.ai, https://www.deeplearning.ai/ai-notes/initialization/, (2019)

30. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks, http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

31. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification, https://arxiv.org/pdf/1502.01852.pdf, (2015)

32. Saito, T., Rehmsmeier, M.: The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. PLOS ONE. 10, e0118432 (2015). https://doi.org/10.1371/journal.pone.0118432

33. Draelos, R.: Measuring performance: AUPRC and average precision, https://glassboxmedicine.com/2019/03/02/measuring-performance-auprc/, (2019)

34. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: From error visibility to structural similarity. IEEE Transactions on Image Processing. 13, 600–612 (2004). https://doi.org/10.1109/TIP.2003.819861

35. Wang, Z., Bovik, A.C.: Mean squared error: Love it or leave it? A new look at signal fidelity measures. IEEE Signal Processing Magazine. 26, 98–117 (2009). https://doi.org/10.1109/MSP.2008.930649

36. Jolliffe I. T., C.J.: Principal component analysis: A review and recent developments, https://doi.org/10.1098/rsta.2015.0202, (2016)

37. Vincent, P., Ca, P., Larochelle, H., Toronto, L., Edu, Lajoie, I., Ca, Y., Ca, P.-A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion yoshua bengio pierre-antoine manzagol. Journal of Machine Learning Research. 11, 3371–3408 (2010)

38. Kingma, D.P., Welling, M.: Auto-encoding variational bayes, https://arxiv.org/abs/1312.6114, (2014)

39. Zhang, A., Lipton, Z.C., Li, M., Smola, A.J.: Dive into deep learning. arXiv.org. (2021). https://doi.org/10.48550/arXiv.2106.11342

40. Springenberg, J., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net, https://arxiv.org/pdf/1412.6806.pdf, (2015)

41. Watson, A.: Deep learning techniques for super-resolution in video games.

42. Wronski, B., Garcia-Dorado, I., Ernst, M., Kelly, D., Krainin, M., Liang, C.-K., Levoy, M., Milanfar, P.: Handheld multi-frame super-resolution. ACM Transactions on Graphics. 38, 1–18 (2019). https://doi.org/10.1145/3306346.3323024

43. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Farley, W., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. Advances in Neural Information Processing Systems. 27, (2014)

44. Donahue, J., Krähenbühl, P., Darrell, T.: Adversarial feature learning. arXiv.org. (2016). https://doi.org/10.48550/arXiv.1605.09782

45. Makhzani, A., Frey, B.: K-sparse autoencoders. arXiv.org. (2013). https://doi.org/10.48550/arXiv.1312.5663

46. Burger, W., Burge, M.J.: Principles of digital image processing. Springer London (2009)

47. Kanan, C., Cottrell, G.W.: Color-to-grayscale: Does the method matter in image recognition? PLoS ONE. 7, e29740 (2012). https://doi.org/10.1371/journal.pone.0029740

48. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems, https://www.tensorflow.org/, (2015)

49. Van Rossum, G., Drake, F.L.: Python 3 reference manual. CreateSpace, Scotts Valley, CA (2009)