UPPSALA
UNIVERSITET

IT 19 077
Examensarbete 30 hp
November 2019

Network Anomaly Detection and Root Cause Analysis with Deep Generative Models

Alexandros Patsanis

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Network Anomaly Detection and Root Cause Analysis with Deep Generative Models

Alexandros Patsanis

The project's objective is to detect network anomalies happening in a telecommunication network due to hardware malfunction or software defects after a vast upgrade on the network's system over a specific area, such as a city. The network's system generates statistical data at a 15-minute interval for different locations in the area of interest. For every interval, all statistical data generated over an area are aggregated and converted to images. In this way, an image represents a snapshot of the network for a specific interval, where statistical data are represented as points having different density values.

To that problem, this project makes use of Generative Adversarial Networks (GANs), which learn a manifold of the normal network pattern. Additionally, mapping from new unseen images to the learned manifold results in an anomaly score used to detect anomalies. The anomaly score is a combination of the reconstruction error and the learned feature representation. Two models for detecting anomalies are used in this project, AnoGAN and f-AnoGAN. Furthermore, f-AnoGAN uses a state-of-the-art approach called Wasstestein GAN with gradient penalty, which improves the initial implementation of GANs.

Both quantitative and qualitative evaluation measurements are used to assess GANs models, where F1 Score and Wasserstein loss are used for the quantitative evaluation and linear interpolation in the hidden space for qualitative evaluation. Moreover, to set a threshold, a prediction model used to predict the expected behaviour of the network for a specific interval. Then, the predicted behaviour is used over the anomaly detection model to define a threshold automatically.

Our experiments were implemented successfully for both prediction and anomaly detection models. We additionally tested known abnormal behaviours which were detected and visualised. However, more research has to be done over the evaluation of GANs, as there is no universal approach to evaluate them.

Handledare: Kim Seonghyun
Ämnesgranskare: Niklas Wahlström
Examinator: Mats Daniels
IT 19 077
Tryckt av: Reprocentralen ITC

I would like to dedicate this master thesis to my beloved family that always supports me.

Declaration of Authorship

I, Alexandros PATSANIS, declares that this thesis titled, "Network Anomaly Detection and Root Cause Analysis With Deep Generative Models" and the work he presents it is all on his own. I confirm that:

- This work has been done wholly while in candidature for a masters degree at Uppsala University in collaboration with Ericsson.
- The source is given for any quoted in this work is given.
- Apart of any quotation, this work he presents is entirely my work, where I have referred to the published work of others, and no part of this master thesis has previously been submitted for a master's degree or any other qualification at Uppsala University or any other university.

Signed: ***Alexandros Patsanis***

Date: ***November, 2019***

Acknowledgements

This master thesis was a great experience and an excellent conclusion of my studies, as a master student, at Uppsala University, which opened new career opportunities. Moreover, by working in a real work environment in collaboration with Ericsson with real work challenges gave me the chance to meet new and exciting people, and make some new friends.

Firstly, I would like to thank my reviewer **Niklas Wahlström** for being an excellent reviewer, as he always provided me with an excellent and quick feedback without any delays. Moreover, I would like to thank my supervisor **Kim Seonghyun** for his advice, weekly meetings, feedbacks and most importantly, for his willingness to help, which was extremely helpful during my thesis in Ericsson.

Furthermore, I would like to thank my manager **Magnus Holmgren** for the excellent collaboration and for the opportunity to collaborate with Ericcson. Last, I would like to thank the whole group or the warm welcome, and for their support and advises during my time in Ericsson. Last, I would like to thank my family because without their support and guidance. I would not be able to accomplish and achieve my life goals.

Contents

Declaration of Authorship ii

Acknowledgements iii

Contents iv

1 Introduction 1

2 Background 5

2.1 Artificial Intelligence . . . 5

2.1.1 Artificial Neuron . . . 6

2.2 Machine Learning . . . 7

2.2.1 Supervised Learning . . . 7

2.2.2 Unsupervised Learning . . . 7

2.2.3 Gradient Descent . . . 8

2.2.4 Stochastic Gradient Descent (SGD) . . . 9

2.2.5 Momentum . . . 9

2.2.6 Adaptive Learning Rates . . . 10

2.3 Deep Learning . . . 10

2.3.1 Feedforward Neural Networks (FFNNs) . . . 10

2.3.2 Convolutional Neural Networks (CNNs) . . . 11

2.3.3 Deep Residual Learning . . . 12

2.4 Maximum Likelihood Estimation . . . 14

2.5 Density Estimation . . . 15

2.5.1 Explicit Density Models . . . 15

2.5.2 Implicit Density Models . . . 16

3 Deep Spatio-Temporal Prediction 17

3.1 Formulation of the Spatial-Temporal Prediction Problem . . . 17

3.2 ST-ResNet . . . 17

3.2.1 Prediction of Flows . . . 18

3.2.2 Simultaneously Forecasting . . . 18

3.2.3 ST-ResNet Model . . . 18

3.2.3.1 Capture Close and Far Away Region Dependencies . . . 20

3.2.3.2 External Components . . . 20

3.2.3.3 Fusion Region Dependencies & External Factors . . . 20

3.2.4 DeepST . . . 20
3.3 DMVST-Net . . . 20
3.3.1 DMVST-Net Framework . . . 21
3.4 Spatial-Temporal Dynamic Network - STDN . . . 22
3.4.1 STDN Framework . . . 22

4 Deep Generative Models **24**
4.1 Introduction . . . 24
4.2 Autoregressive (AR) models . . . 25
4.3 Autoencoders . . . 25
4.4 Kullback-Leibler (KL) Divergence . . . 26
4.5 Variational Autoencoders (VAEs) . . . 27
4.6 Generative Adversarial Networks . . . 28
4.6.1 Adversarial Nets Framework . . . 29
4.6.2 Training Process . . . 30
4.7 DCGAN . . . 31
4.8 Wasserstein GAN . . . 32
4.9 Wasserstein GAN with Gradient Penalty (WGAN-GP) . . . 34
4.10 GANs Evaluation . . . 35

5 Datasets **40**
5.1 MNIST . . . 40
5.2 Converter Framework . . . 40
5.2.1 Performance Measurement Counters (PM Counters) . . . 40
5.2.2 Data Preprocessing . . . 40
5.2.3 Fetch Data . . . 40
5.2.4 Convert Data . . . 41
5.2.5 Converter - Check the Fetched Data . . . 42
5.2.6 Converter - Convert Cell Folders to Individuals Days . . . 43
5.2.7 Converter - Image Generation . . . 44
5.2.8 Generated Input Dataset . . . 45

6 Anomaly Detection Approach **47**
6.1 Anomaly Detection . . . 47
6.2 Network Anomaly Detection . . . 47
6.3 Anomaly Detection with Deep Learning . . . 48
6.4 Deep Hybrid Models (DHMs) . . . 48
6.4.1 Results of Anomaly Detection . . . 49
6.4.2 Unsupervised Deep Anomaly Detection . . . 49

6.5 AnoGAN . . . 49
6.5.1 Mapping Images to the Latent Space . . . 50
6.5.2 Combined Loss Function to Mapping Unseen Images to the Latent Space 50
6.5.3 AnoGAN Detection of Anomalies . . . 51
6.6 Fast AnoGAN (f-AnoGAN) . . . 51
6.6.1 Training f-AnoGAN for Anomaly Detection . . . 51
6.6.2 Encoder Architecture - ziz . . . 52
6.6.3 Image to Image Encoder Architecture - izi . . . 52
6.6.4 Image to Image Encoder Architecture with D residual in the feature space - izi_f . . . 53
6.6.5 Fast AnoGAN Detection of Anomalies . . . 53

7 Experiment Setup and Results **54**
7.1 Original Prediction Model Implementation . . . 55
7.2 ST-ResNet - Spatial-Temporal Prediction Implementation . . . 57
7.2.1 Results of ST-ResNet . . . 57
7.3 AnoGan - Network Anomaly Detection Implementation . . . 61
7.3.1 Results of Mapping New Images to the Latent Space & Anomaly Score . . . 63
7.4 f-AnoGAN - Implementation & Results . . . 65
7.4.1 Results of WGAN-GP . . . 66
7.4.2 Results of Implementation Training Mapping - izi_f Architecture . . . 71
7.5 Results of Anomaly Detection $A(x)$ . . . 73
7.6 Discussion . . . 74

8 Conclusion and Future work **77**
8.1 Conclusion . . . 77
8.2 Future Work . . . 78

Bibliography **80**

A Appendix: Importing Data Class **83**

B Appendix: AnoGAN - Threshold for Anomaly Detection **84**

C Appendix: f-AnoGAN -Linear Z-space Interpolation **85**

D Appendix: According the Real Time Anomaly Detection **86**

List of Figures

1.1 An example of a Converted Image . . . 1
1.2 Information Parts that the Converter Framework Combines . . . 2
1.3 Normal (left) - Anomalous (right) Generated grey-scale (64x64) Images . . . 2
1.4 Pipeline of the Anomaly Detection & Root Cause Anomaly . . . 4

2.1 Relationship between Artificial Intelligence, Machine Learning and Deep Learning 5
2.2 Artificial Neuron Architecture . . . 6
2.3 Gradient Descent . . . 8
2.4 An example of Stochastic Gradient Descent with and without Momentum . . . 9
2.5 An example of a Convolution Operation . . . 12
2.6 An example of two Residual Blocks . . . 13
2.7 An example of Gaussian Distribution (Normal Distribution) . . . 14

3.1 The Spatial-Temporal (ST-ResNet) - Architecture . . . 19

4.1 Autoencoder (AE) - Architecture . . . 26
4.2 Variational Autoencoders (VAE) - Architecture . . . 27
4.3 Basic Architecture of Generative Adversarial Networks. . . . 28
4.4 An example of Wasserstein-GAN Transport Plan. . . . 32
4.5 Wasserstein Generative Adversarial Network (WGAN) - Architecture . . . 34

5.1 Fetching and Converting Process . . . 41
5.2 Converter - JSON Setting file . . . 42
5.3 Converter Framework - Generated grey-scale (64x64) Images . . . 45
5.4 Converter Framework - Anomalous Generated grey-scale (64x64) Images . . . 46

7.1 Original ST-ResNet RMSE - Closeness: 3 Period: 1 Trend: 1 . . . 55
7.2 Original ST-ResNet RMSE - Closeness: 3 Period: 3 Trend: 3 . . . 56
7.3 Original ST-ResNet RMSE - Closeness: 1 Period: 1 Trend: 1 . . . 56
7.4 ST-ResNet RMSE - Predicted 32x32 - Closeness: 1 Period: 1 Trend: 1 . . . 58
7.5 Prediction Results 32x32 - Closeness: 1 Period: 1 Trend: 1 . . . 58
7.6 ST-ResNet RMSE - Predicted 64x64 - Closeness: 1 Period: 1 Trend: 1 . . . 59
7.7 Prediction Results 64x64 - Closeness: 1 Period: 1 Trend: 1 . . . 59
7.8 ST-ResNet RMSE - Predicted 64x64 - Closeness: 3 Period: 1 Trend: 1 . . . 60
7.9 Prediction Results 64x64 - Closeness: 3 Period: 1 Trend: 1 . . . 60
7.10 MNIST - Labeled Training with AnoGAN . . . 61

7.11 Experiment - Labeled Training with AnoGAN . . . 62
7.12 Experiment - Unlabeled Training with AnoGAN . . . 63
7.13 Experiment - Mapping to the latent space - AnoGAN . . . 64
7.14 f-AnoGAN: WGAN-GP - Generator Loss . . . 66
7.15 f-AnoGAN: WGAN-GP - Critic (Discriminator) Loss . . . 67
7.16 f-AnoGAN: WGAN-GP - Critic (Discriminator) - Mean Costs between Validation and Anomaly . . . 68
7.17 f-AnoGAN: WGAN-GP: Histogram of Critic Scores for Validation and Anomalous Images . . . 68
7.18 f-AnoGAN: WGAN-GP - Critic (Discriminator) - F1 score . . . 69
7.19 f-AnoGAN: WGAN-GP - Ground Truth - Training Input . . . 70
7.20 f-AnoGAN: WGAN-GP - Generated Samples - 64x64 grey-scale - Iteration 224000 70
7.21 f-AnoGAN: Training Mapping with izi$_f$ $-$ *Loss* . . . 71
7.22 f-AnoGAN: Validation Loss - Training Mapping with izi$_f$ . . . 72
7.23 f-AnoGAN: Real and Reconstructed samples - Iteration 100 . . . 72
7.24 f-AnoGAN: Real and Reconstructed samples - Iteration 50000 . . . 72
7.25 f-AnoGAN: Anomaly Score - Healthy Sample . . . 73
7.26 f-AnoGAN: Anomaly Score - Anomaly Sample . . . 73
7.27 Test AnoGAN - Query Image - Timestamp: 00:00:00 . . . 74
7.28 Test f-AnoGAN - Query Image - Timestamp: 00:00:00 . . . 75
7.29 Test AnoGAN - Query Image - Timestamp: 06:00:00 . . . 75
7.30 Test f-AnoGAN - Query Image - Timestamp: 06:00:00 . . . 75
7.31 Test AnoGAN - Query Image - Timestamp: 12:00:00 . . . 76
7.32 Test f-AnoGAN - Query Image - Timestamp: 12:00:00 . . . 76
7.33 Test AnoGAN - Query Image - Timestamp: 18:00:00 . . . 76
7.34 Test f-AnoGAN - Query Image - Timestamp: 18:00:00 . . . 76

C.1 f-AnoGAN: WGAN-GP - Linear z-space Interpolation for Random Endpoints - Iteration 220000 . . . 85

List of Tables

3.1 ST-ResNet - Training Parameters 19

7.1 Experiment - Python Packages 54
7.2 Spatial-Temporal Prediction Models 54
7.3 Experiment - Modified ST-ResNet - Training Parameters 57
7.4 Experiment - Final Results - Spatial-Temporal Prediction odel 61
7.5 Generator - AnoGAN Implementation 61
7.6 Experiment - AnoGAN - Training Parameters 62
7.7 Experiment - Mapping to New Images and Predicted & Anomaly Score Based on $\mathcal{A}(x)$ 65
7.8 Experiment - Training f-AnoGAN - Parameters for WGAN with Gradient Penalty 67
7.9 Experiment - Training f-AnoGAN - Parameters for izi_f Architecture 71
7.10 Experiment - Anomaly Score A(x) of New Unseen & Predicted Images 73
7.11 Experiment - Overview between AnoGAN vs f-AnoGAN Implementation 74

8.1 Experiment - Generative Adversarial Network (GANs) - Papers 77

List of Abbreviations

AI	*A*rtificial *I*ntelligence
ML	*M*achine *L*earning
DL	*D*eep *L*earning
ANN	*A*rtificial *N*eural *N*etwork
NN	*N*eural *N*etwork
MSE	*M*ean *S*quared *E*rror
SGD	*S*tochastic *G*radient *D*escent
FNN	*F*eedforward *N*eural *N*etwork
CNN	*C*onvolutional *N*eural *N*etwork
RNN	*R*ecurrent *N*eural *N*etwork
LSTM	*L*ong *S*hort-Term *M*emory
ReLU	*R*ectified *L*inear *U*nit
GANs	*G*enerative *A*dversarial *N*etworks / Vanilla GANs
FVBNs	*F*ully *V*isible *B*elief *N*etworks
NICA	*N*onlinear *I*ndependent *C*omponents *A*nalysis
AD	*A*nomaly *D*etection
AE	*A*utoencoder
VAEs	*V*ariational *A*utoencoders
ROP	*R*esult *O*utput *P*eriod

List of Symbols

a	A scalar (integer or real)
a	A vector
A	A matrix
A	A tensor
I_n	Identity matrix with n rows and n columns
I	Identity matrix with dimensionality implied by context
$e^{(i)}$	Standard basis vector $[0, \dots, 0, 1, 0, \dots, 0]$ with a 1 at position i
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable
$\mathcal{N}(x, \mu, \sigma^2)$	Gaussian Distribution or Normal Distribution

Chapter 1

Introduction

This thesis was a collaboration with Ericsson AB based in Kista, Stockholm. Ericsson is a Swedish multinational networking and telecommunication company and is one of the leading providers of ICT to service providers, where around 40% of the world's mobile traffic passes through the Ericcson' networks. The company offers many services such as software telecommunications operators, equipment, mobile and fixed broadband operations.

The companies' telecommunication network generates massive amounts of data with high complexity. Therefore, it is crucial for the network to remain stable. Real-time anomaly detection plays an essential role in ensuring that the network operation is working efficiently. Ensuring that the network is stable can be done by taking actions in detecting anomalies and preventing any anomalous activities by identifying the abnormalities and the cause of the potential problems.

Ericsson deploys a large number of new features in its customer network. The operations that these features are accomplished in the nodes of the radio base stations generate a large amount of data. Additionally, during the day the amount of traffic load increases as the demand is higher, whereas during the night the traffic load decreases dramatically. This variation between night and day creates a **behaviour pattern** that can be used to identify possible divergences in the network.

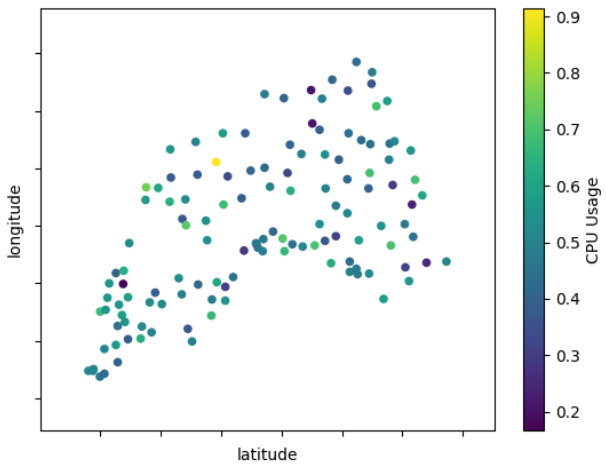


FIGURE 1.1: An example of a Converted Image

The figure is an example of a generated image, which represents the CPU load in a specific timestamp in the network. Each point represents an aggregation of cells over the same area, and the colours represent a mean value of all the aggregated cells.

In this way, our dataset consists of **Performance Measurement** (PM) counters, taken for multiple locations over an area of interest [1]. Furthermore, to work with multiple location, where each of this locations represents multiple PM counters, a **python converter framework** created (section 5.2), where for every 15 minutes (timestamp) interval an image is generated, as illustrated in 1.1, which represents aggregated PM counters for every cell location on a map. Figure 1.2 shows the information that the converter framework combines.

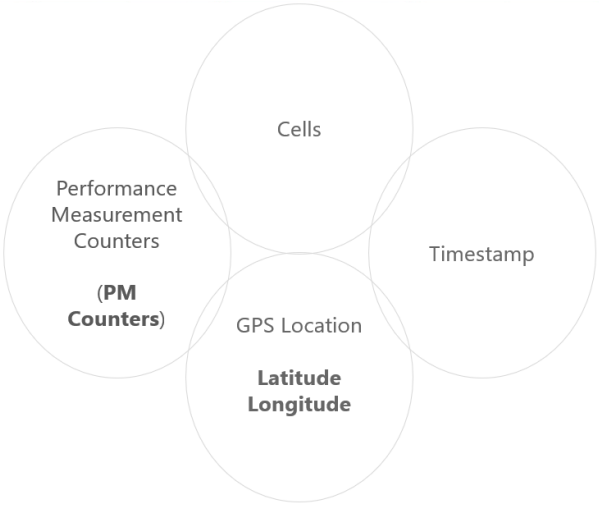


FIGURE 1.2: Information Parts that the Converter Framework Combines

This figure illustrates the components that the converter combines, where the python-based converter framework combines multiple cells with different locations over one performance management counter from some period.

In this work, we consider the case that the data pattern follows a **periodical pattern**. As a result, when a divergence occurs in the behaviour pattern of the network, this might means that a network anomaly appears. Moreover, the objective of this thesis is to identify anomalies in the next timestamp among multiple radio base stations in different locations in an area. Specifically, the expected anomalies it is possible to occur after a **massive update** over all the cells in an area, where most of the locations will have higher density. For instance, in the case of an anomaly occurs, figure 1.1 would have multiple locations represented with high-density values (yellow colour). However, in order to work with the deep learning models[2], we have converted the images to grayscale, where darker colours mean higher density values. Figure 1.3 illustrates two generated images, where the image on the left is considered as normal and the image on the right as anomalous.

FIGURE 1.3: Normal (left) - Anomalous (right) Generated grey-scale (64x64) Images

[1] In our case, the area of interest is a specific city, as the locations considered sensitive data, we cannot share the exact city. However, this project can be applied in a different location as is not bounded with just one location.

[2] We have to define the meaning of the colour in order to work with coloured images

Note, that these aggregated PM counters are several for each location, and the average of them are represented in every location, as the objective of this thesis is to find anomalies that occur in a vast area and not just in one particular PM counter. Therefore, in the case of an anomaly, multiple aggregated PM counters will be affected.

On the one hand, anomaly detection is a challenging area, where studies in statistical approaches have been done at the early of the 19th century [1]. Moreover, annotating images as normal and abnormal is a time-consuming and expensive task. Also, defining an anomaly can be a challenging task as **unknown anomalies** might exist. On the other hand, because of the network complexity, a robust big data analytics solution has been required. Deep learning, as subsection 3.2.4 describes, has shown robust applications in complex real-world problems such as fraud, cyber-intrusion, medical anomaly sensor networks anomaly, and so on. Moreover, Deep Learning methods can learn the hidden semantic parts of representation, where two primary approaches for learning exists, **Supervise Learning** where labelled data are needed during the training and **Unsupervised Learning** where no label data are needed.

However, an alternative approach is to use a **Semi-Supervised Learning** approach, where only a small number of labelled data needed. For example, in a classification task, where the goal is to classify normal from anomalous images, we could use a model which could learn to distinguish the anomalies from the healthy images without explicit have labelled data.

Additionally, **Deep Generative Models** (4.1) have been used in the concept of anomaly detection [2, 3]. In recent literature, a subcategory of deep generative models, called **Adversarial Generative Networks** (GANs) (4.6) has shown the potential to learn in a semi-supervised way, as was mentioned in [4]. Moreover, Deep Generative Models can be divided into two categories, one is the **explicit density** models (2.5.1) and the secondly the **implicit density models** (2.5.2), where the prior defines explicit a log-likelihood function, where the last aims to generate samples and compare them with true distribution and given that information to adjust the parameters of the model.

In this way, implicit density models aim to avoid density intractability. There are two leading generative model, **Variational Autoencoders** 4.5, which makes a use of explicit density estimation and GAN, which are using an implicit density approach. However, because variational autoencoders uses of a lower bound approximation of the log-likelihood which results in blurry samples, have been less attractive compared with GANs which has shown good generative samples.

The intuition of GANs is based on game theory where GANs consists of two networks, the **Generator** and the **Discriminator**. The training consists of only healthy data, where the Generators' objective is to generate samples close to the true data distribution and fool the Discriminator, whereas the Discriminator receives samples from both the healthy training data and the generated, and aims to distinguish (classify) if these data are fake (generated) or real. GANs has been quite successful in the last years, with more than 500 models[3].

In this way, GANs has been used in a vast variety of anomaly detection problems such as lesion detection in medical imaging, credit-card fraud detection, Anomaly Detection in Wireless Networks, etc. The anomaly detection approach aims to learn the **manifold** of the healthy input data, and classifying new unseen data either healthy or anomalous. There are two main steps to achieve such an anomaly detection model:

- The first step consists of a model that is able to learn to **generate representation** close the healthy input data. Our contribution includes an extensive overview of state-of-the-art approaches using GANs for generating samples. Also, vanilla GANs [5] compared with

[3]The GAN zoo, presents a list of the most recent GANs implementations: https://github.com/hindupuravinash/the-gan-zoo

different loss objectives for training, which can improve the performance of the model such as Wasserstein GANs (4.8) and Wasserstein GANs with gradient penalty (4.9).

- The second step is to make a **mapping** of new unseen images to the latent space, where the mapping can be used to score the tested images. Our contribution consists of investigating techniques for mapping new images to the latent space, where are implementation consider mapping techniques [2, 3].

On the other hand, evaluation of GANs is an important open research field, wherein this thesis GAN evaluation methods were studied, based on [6]. More specifically, section 4.10 describes the different GAN evaluated methods for both **quantitative** and **qualitative measurement** evaluation. Moreover, the quantitative evaluation was based on **F1 score** computed from **Precision** and **Recall**, and the **Wasserstein Critic** loss used for addressing limitation such as **overfitting** and **mode collapse**. Additionally, subsection 4.10 describes the qualitative evaluation measurements, wherein this project investigation and visualisation of the internals of the network used to evaluate the model quantitatively, where **space continuity** can be used for analysing the level of detailed that a model can capture (Appendix C)

Overall, Deep Learning has been quite successful in the last years, and new algorithms have been invented. These algorithms gave higher accuracy and speed. Therefore, this project aims to research state-of-the-art approaches to consider Deep Generative Models in the area of network anomaly detection and root causes analysis. Additionally, our intuition is to use both anomaly detection models and prediction models to detect anomalies and visualise them automatically. Moreover, the prediction model can be used for scoring the predicted images and therefore provides an automatic way of setting a threshold.

More specifically figure 1.4 illustrates the pipeline for anomaly detection model, where two models, one for prediction (ST-ResNet: section 3.2) and the other one for anomaly detection (AnoGAN: section 6.5 and f-AnoGAN: section 6.6), have been trained with the healthy network data. Then for a specific timestamp of interest, the predicted model results in an image corresponding to that timestamp. The predicted image, which corresponds to the next timestamp, is passed to the anomaly detection model. Then, the anomaly detection model assigns an expected anomaly scored (threshold) by assuming that the prediction model predicts the timestamp of interest accurately.

Next, the anomaly detection model is used for assessing the tested timestamp, where an anomaly score has been assigned. Final, both results of predicted and tested input data can be evaluated. For instance, high anomaly score values indicate the presence of abnormalities.

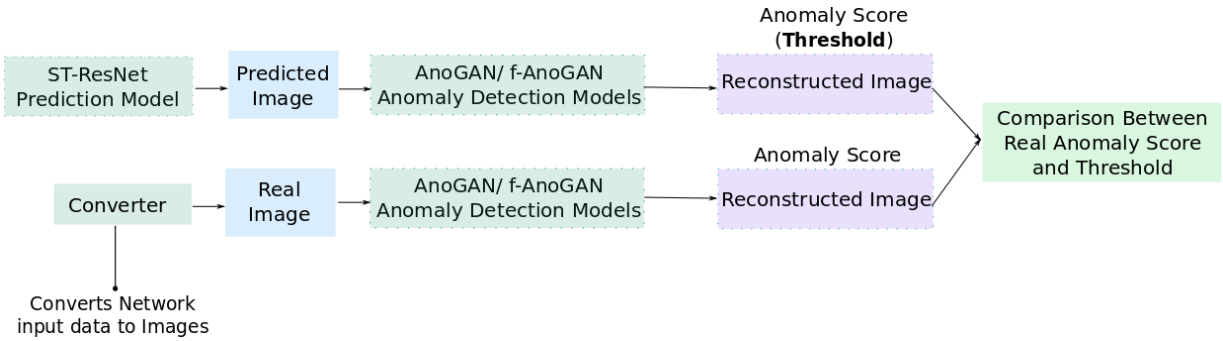


FIGURE 1.4: Pipeline of the Anomaly Detection & Root Cause Anomaly

Overall, in this project, a simple implementation of settings a threshold has been implemented. Therefore, more investigation has to be done with experts to define a threshold. However, this work shows that **Deep Generative Models** can tackle the manual procedure of adjusting a threshold.

Chapter 2

Background

This chapter provides an introduction to all the theoretical concepts that are included in this project. In this way, the reader can have a cohesive perception between the theory and the methods considered and used during this thesis. Therefore, this chapter does not attempt to describe every concept of AI, Machine Learning and Deep Learning in deep detailed, as that is not feasible due to the time limits of a master thesis. In contrast, the chapter aims to provide a concise understanding of different concepts. Furthermore, for a deeper understanding and for curious readers, related references will be provided thought the chapter.

Consequently, the following chapter starts by explaining high-level concepts such as **Artificial Intelligence**, **Machine Learning** and **Deep Learning**, to more details such as algorithms and methods, where

- Section 2.1: Artificial Intelligence is mainly based on [7, 8].
- Section 2.2 Machine Learning is based on [8, 9].
  - Subsections 2.2.4 - 2.2.6 are based on [8].
- Section 2.3: Deep Learning is mainly based on [8]. (Credits [1])
  - Subsection 2.3.1: Feedforward Neural Networks (FFNNs) is based on [8, 11].
  - Subsection 2.3.2: Convolutional Neural Networks (CNNs) is based on [8, 12].
  - Subsections 2.5.1: Explicit Density Models and 2.5.2: Implicit Density Models are based on [4, 13].

2.1 Artificial Intelligence

Artificial Intelligence (AI) can be regarded as a combination of a wide range of research fields such as computer science, biology, physiology, and so on. AI aims to mimic the behaviour of neurons of a human brain by transferring knowledge as a programme that simulates human intelligence.

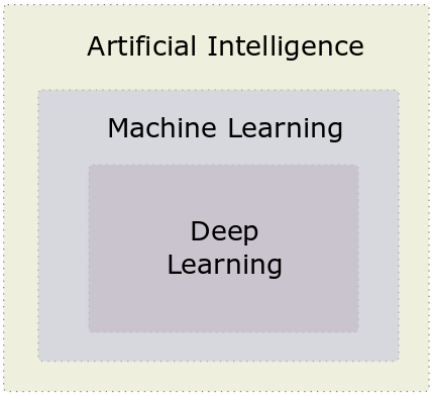


FIGURE 2.1: Relationship between Artificial Intelligence, Machine Learning and Deep Learning

[1] Gredits to [10] study working with GANs implementation

Due to the difficulty of AI to depending on hard-coded knowledge, **Machine Learning** has been introduced to tackle this problem by extracting knowledge using learning algorithms. In this way, instead of explicit hard-code the algorithm, it can be learned to adjust the necessary knowledge for different tasks, such as classification and logistic regression.

In **Machine learning** (ML), **Representation Learning** intends to learn both the mapping from representation to output and the representation itself. The intuition behind this approach is to tackle the problem of designing optimal feature sets to extract from them.

An example of a representation learning algorithm is called **Autoencoder**, as section 4.3 describes, where Autoencoders make a use of two functions. The first one converts the input data into a different representation, called **encoder**, whereas the second, called **decoder**, aims to decipher the converted input to the original representation. Moreover, creating feature sets can be quite complicated when it is aimed to accomplish a more complex task.

Due to the demand for real-world tasks, where it is challenging to learn features representation with a high-level of abstraction from the raw data. **Deep Learning** (DL), as section 2.3 describes, aims to leverage this problem by splitting the representation expressed by more straightforward representations and learn an abstraction of the input, such images, speech, and so on. The relation between Artificial Intelligence, Machine Learning and Deep Learning, as was described above, is illustrated in figure 2.1.

2.1.1 Artificial Neuron

An artificial neuron [7] consists of an input vector $\boldsymbol{x} = (x_1, x_2, ..., x_I)$, each of the values of this vector is associated with a weight. These weights are used to increase or decrease the strengthen of the input. In order to compute the output of a neuron, an activation function is used which receives weights and a bias (threshold). The purpose of weights is to increase or decrease the strengthen of the *input*; whereas, **bias** (threshold) is used to influence the strength of the output.

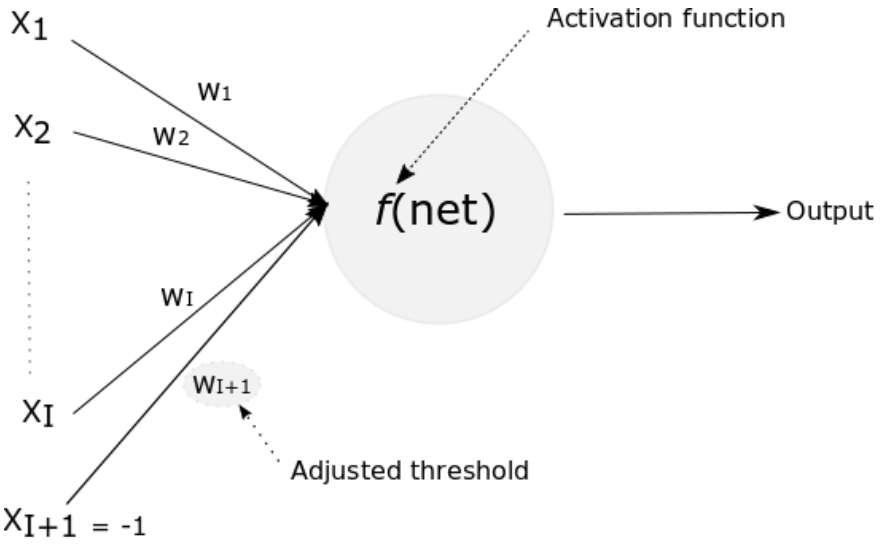


FIGURE 2.2: Artificial Neuron Architecture

The figure illustrates an artificial neuron, where $\boldsymbol{x}$ refers to the input values, $\boldsymbol{w}$ refers to the weights, and $\boldsymbol{x_{I+1}}$ to the bias term. Note that the bias term can take different values, and $\boldsymbol{x_{I+1}} = -1$ is just an example.

Figure 2.2 illustrates an artificial neuron; the **net** refers to the sum of the inputs $\boldsymbol{x}$ multiplied with its corresponding weights. Moreover, during training both weights and bias are adapted, where the input vector (net) holds the bias as an additional input unit $\boldsymbol{x_{I+1}}$. The sign of $\boldsymbol{x_{I+1}}$

can be changed for different **activation functions** [2]. In that way, the equation $f(net - bias)$ can be written as $f(net)$, where f is the **activation function**, and the net is the weighted sum of the input.

$$net = \sum_{i=1}^{I} x_i w_i \tag{2.1}$$

2.2 Machine Learning

Commonly, Machine Learning is divided into three types, **Supervised Learning**, **Unsupervised Learning** and **Reinforcement Learning**. The following two subsections talk about Supervised and Unsupervised Learning. Reinforcement Learning is out in the scope of this thesis. However, Deep Generative Models can be incorporated with Reinforcement Learning, as is mentioned in [4].

2.2.1 Supervised Learning

Supervised learning aims to learn a mapping or an accusation of elements from an input space x to an output space y. In Supervised Learning, it is given a labeled set of inputs-outputs, which is formulated by equation 2.2:

$$\mathcal{D}\{(x_i, y_i)\}^N{}_{i=1}, \tag{2.2}$$

where the training set is defined by $\mathcal{D}$, and the number of the samples that the training set has by N. The input space can be analysed as vectors of D-dimensionality. For instance, the x_i can be features that are structured to define more complex objects such as images, time-series data, and so on. Also, each of these features can be considered as the height and the weight represented as vectors.

On the other hand, we can think of the output in a similar manner, where y_i can be variable that represents a restricted set, such as **categorical** or **nominal**, where $y_i \in \{1, ..., \mathcal{C}\}$; or that y_i is a **real-valued scalar**.

Therefore, two main formulation problems exist in Supervised Learning, **classification** and **regression** problem. When y_i is a categorical variable under some finite set, the problem is referred to classification; and when y_i output is expected to be a continuous variable, the problem refers to a regression problem.

For supervised anomaly detection approach, it is a big need for a vast amount of labelled data. The prepossessing to label data can be expensive and time-consuming. As a result, it limits the ability to exploit image data for anomaly detection. Furthermore, supervised anomaly detection approaches are not efficient for detecting **unknown anomalies** that the system has not seen before, due to the growing data and the uncertainty of predefined an anomaly, as human experts might fail to predefined all the possible anomalies.

2.2.2 Unsupervised Learning

The second type of Machine Learning, as is mentioned before, is called **Unsupervised Learning**. This approach aims to discover patterns in the data, where the main difference with Supervised Learning is that Unsupervised Learning does not make use of labeled data:

$$\mathcal{D} = \{x_i\}^N{}_{i=1} \tag{2.3}$$

Moreover, this problem is not defined in the sense of finding a specific pattern in our data space, and unlikely with Supervised learning, there are no labeled data to be compared with.

[2] An **activation function** regulates the firing strength of a neuron.

Some examples of Unsupervised learning are clustering, dimensionality reduction, density estimation, feature learning, and so on.

Unsupervised Learning aims to learn the **underlying distribution** of the data. For instance, when we are dealing with high-dimensional input, methods that reduce the dimensionality can be used. In this way, it is feasible to capture meaningful information. The intuition behind these methods is that even if the input space is high-dimensional, they might have a small degree of variability, which describes the latent (hidden) factors. For instance, it can be thought of "describing" different images of eyes, by referring to a few underlying latent (hidden) factors such as shape, lighting, and so on. As a result, these few underlying latent factors can depict the variability of the input space.

Last, a combination of both Supervised and Unsupervised Learning has shown improvements in training called **Semi-supervised Learning**, where only a small part of the labeled data needed for the training.

2.2.3 Gradient Descent

Gradient Descent (GD) [7, 8], based algorithms have as objective to minimise the error between the output and the actual target of a neuron. In other words, GD aims to minimise a function $f(\boldsymbol{x})$ by modifying $\boldsymbol{x}$ in such a way that the $\boldsymbol{x}^* = \arg\min f(\boldsymbol{x})$ [3].

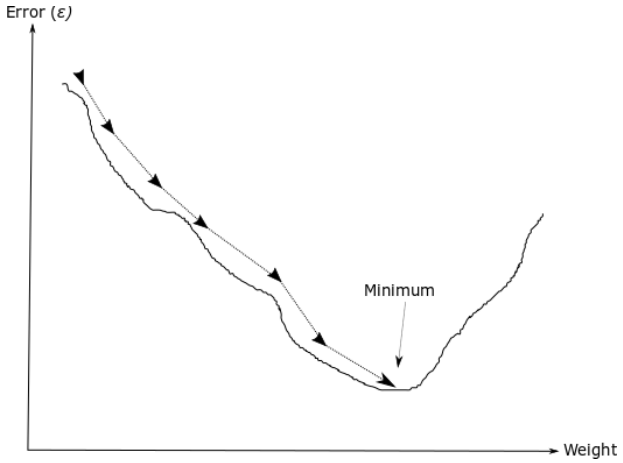


FIGURE 2.3: Gradient Descent
An examples of Gradient Descent over one weight.

Gradient Descent can be seen as **moving** downhill to a minimum, as is illustrated in figure 2.3. Mainly, this movement can be achieved by calculating the **derivative** of a function $y = f(\boldsymbol{x})$, denoted as $f'(\boldsymbol{x})$, over some epochs (iterations) with opposite sign. In general, the derivative will return the slope of the function at point x. In other words, it tells us how to make a slight change the weights to minimise or maximise the function $f(x + \varepsilon) \approx f(x) + \varepsilon f'(x)$. Furthermore, when the function has multiple inputs, the partial derivatives $\frac{\partial}{\partial_{x_i}} f(\boldsymbol{x})$ must be calculated. The **partial derivative** of a function is denoted as $\nabla_x f(\boldsymbol{x})$ and is a vector which contains all the partial derivatives. Gradient Descent moving to the minimum by calculating

$$x' = x - \varepsilon \nabla_x f(x), \tag{2.4}$$

where ε denotes to the learning rate, which determines the step size forward to the minimum, usually, the step size is a small constant value. Overall, when the gradient is equal to zero for each element, then Gradient Descent has converged.

[3] $\boldsymbol{x}^*$ asterisk refers to minimise or maximise a function

2.2.4 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) separates the training in batches instead of taking the whole training set and compute the loss, compared to traditional Gradient Descent methods, as discussed in 2.2.3. In this way, SGD reduces the computational complexity by sampling training examples as mini-batches $\mathbb{B} = \{\boldsymbol{x^1}, \ldots, \boldsymbol{x^{m'}}\}$ drawn from the dataset uniformly. The following algorithm presents the update process of SGD over iteration m. As we increase the size of the batches, the training process is being slower.

Algorithm 1: Stochastic Gradient Descent - SGD (reproduced from [8])

1 Initialise θ parameter ;
2 Learning rate ε_k ;
3 **while** *stopping criterion not met* **do**
  Sample a mini-batch m from training set corresponds to $y^(i)$
  Calculate gradient: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i^m L(f(x^{(i)}; \boldsymbol{\theta}), y^{(i)})$
  Update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \varepsilon\hat{\boldsymbol{g}}$
4 **end**

Moreover, SGD adds noise to the training set as randomly select the batch for the training, which is not disappearing even if SGD finds the minimum. Overall, the noise that SGD introduces can prevent overfitting, but as the batch-size decreasing the training process will become slower. Commonly, the batch size is 32 or 64 for the training process.

2.2.5 Momentum

An improvement of Stochastic Gradient Descent (SGD), which called **Momentum**, aims to leverage the slow convergence of SGD due to the fluctuation of steps that SGD has, for different gradient values. Momentum introduces a past gradient where the idea is to accrue an exponentially decaying moving average of prior gradients and move in their direction. The way that momentum works is by introducing two hyperparameters, $\boldsymbol{u}$ for velocity and $\boldsymbol{a}$ for the speed. Speed hyperparameter controls the contribution of prior gradient decay. Figure 2.4 illustrates an example of Stochastic Gradient Descent (SGD) with and without momentum, where green line represents the SGD using momentum, whereas the black arrow represents the step at each point that SGD takes.

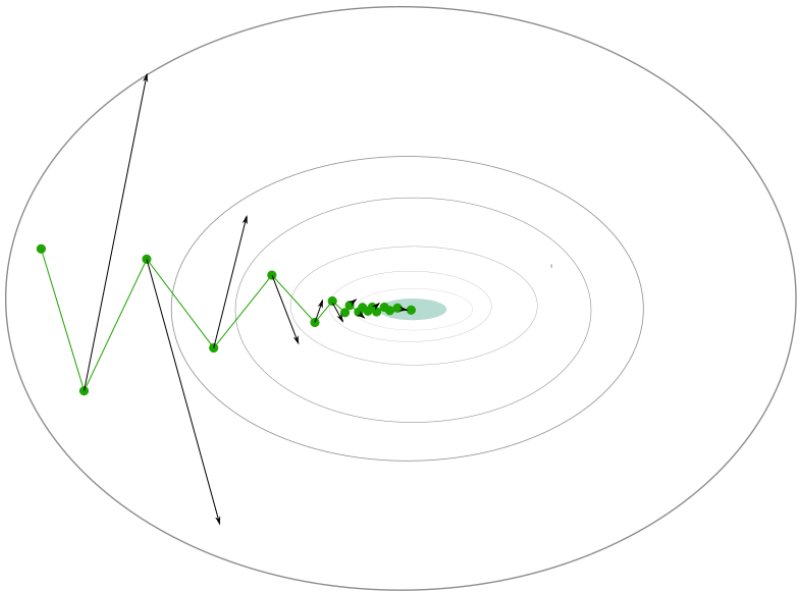


FIGURE 2.4: An example of Stochastic Gradient Descent with and without Momentum

Furthermore, there is an additional version of stochastic gradient descent with momentum called **Nesterov Momentum** [8], which aims to evaluate the gradient after applied the present $\boldsymbol{u}$ by adding a *correction factor*. In practice, this version does not improve the convergence rate.

2.2.6 Adaptive Learning Rates

Cost functions are hypersensitive in some directions of space parameter that compared with others. Furthermore, hyperparameters such a learning rate influence significantly model performance. SGD with momentum aims to tackle those problems but do not introduce any extra hyperparameters. Adaptive learning algorithms [8] such as **RMSprop** and **Adam**, add a separate learning rate for each parameter. In this way, the model can learn the directions of space parameter.

During the training, these learning rates are being adapted. RMSProp makes a use of an exponentially weighted moving average, which gives a smoother converge. A recent successful algorithm, called Adam, has been introduced to improve the training by combining in a sense momentum and RMSProp algorithm. Most of the implementation in this thesis use Adam as an optimiser, where more information can be found in [14].

2.3 Deep Learning

Deep Learning is a subgroup of Machine Learning and has been used the recent years successfully in various applications which deals with problems such as capturing complex **factors of variations**. For instance, we can think these factors as a piece of information that describe positions of objects on an image, posture of individuals, angle, different timestamps during a day, and so on. The way that Deep Learning accomplishing that is by splitting the complex problems into more straightforward problems. For example, Deep Learning learns to present the data in a nested ranking by using layers of the Neural Network.

For instance, one form of Deep Learning is the Feed-forward Neural Network, as a section 2.3.1 describes. Overall, Deep Learning has been successful in dealing with complex real-world problems such as fraud, cyber-intrusion, medical anomaly, sensor networks anomaly, video surveillance, IoT anomaly, and so on. Traditional algorithms fail to capture complicated structures, and as data increasing, traditional techniques are not able to handle anomaly detection in the future. Furthermore, Deep Learning dealing with such problems by using Unsupervised Deep Learning Models, and there is no need to manually feature labelling as the model can learn hierarchical discriminate features from the data. Last, the limitation of definite the healthy representation boundaries is often a challenge in both traditional and Deep Learning approaches.

2.3.1 Feedforward Neural Networks (FFNNs)

Feedforward Neural Networks (FFNNs) or **Multilayer Perceptrons** (MLPs) [8] aims to approximate some complex function f^*. FFNNs accomplish this approximation by defining a mapping to that function $f_{\boldsymbol{\theta}} : \boldsymbol{X} \rightarrow \boldsymbol{Y}$, controlled by parameters $\boldsymbol{\theta}$. For instance, in the case of a classification problem where the model's input refers to $\boldsymbol{x}$ and model's output to a category $\boldsymbol{y}$ this mapping is defined by $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})$ [4]. During feedforward propagation, the input data pass through the network and then produces outputs, but this not enough as the model has to learn, which of these parameters $\boldsymbol{\theta}_{optimal}$, returns the optimal result which approximates that complex function $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}_{optimal}) \approx f^*(\boldsymbol{x})$.

The way that FFNNs can learn these parameters is by using **Gradient Descent** and update the weights of the network through **backpropagation**, which opposite to feedforward propagation, allows the information to flow backwards from the output to the input layer. In other words, it can be thought as comparing the output from the feedforward process to the desired output by calculating a **loss function** (cost function) as is denoted in equation 2.5.

[4] $f(\boldsymbol{x}; \boldsymbol{\theta})$ sometimes can be written as $f(\boldsymbol{x})$

$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{m=1}^{m} L(x^{(i)}, y^{(i)}, \boldsymbol{\theta}) \tag{2.5}$$

The next step is to update the weights of the neural network in such a way that minimises the $J(\boldsymbol{\theta})$, which means the difference between model output and expected output. This comparison, which is performed by the $J(\boldsymbol{\theta})$, measures mathematically of how far is the network in that specific case. Furthermore, information through FFNNs flows in one direction, and there are no feedback connections.

Overall, FFNNs are essential as is the base for many successful networks such as **Convolutional Neural Networks** in object recognition, and **Recurrent Neural Networks** in natural language applications, which is an expansion of FFNNs, where feedback connections exist.

2.3.2 Convolutional Neural Networks (CNNs)

Convolution Neural Network (CNN) is a specific kind of Neural Network for processing data such as time-series (1D) or images (2D). The intuition behind CNN is to learn the invariance between different features. In other words, CNN allows to learn a non-linear combination of the input space and extract meaningful features. For example, individual pixels of an image does not provide any useful information, but different combinations of pixels can provide semantic meaning. The CNN process includes a kernel which is moving thought the entire image, the step size that this kernel moves during the training are called **stride**, which composes a way of subsampling. The convolutional operation aims to downsample the image, and the kernel can find features in the image.

The intuition behind CNNs is to learn features via a mathematical operation called **convolution**, where a convolution operation is denoted with an asterisk. For instance, in the case of an image $\boldsymbol{I}$, where both $\boldsymbol{I}$ and $\boldsymbol{K}$ are two-dimensional arrays; the convolution operation is denoted as:

$$S(\boldsymbol{i}, \boldsymbol{j}) = (\boldsymbol{I} * \boldsymbol{K})(\boldsymbol{i}, \boldsymbol{j}) = \sum_{\boldsymbol{m}} \sum_{\boldsymbol{n}} I(m, n) K(i - m, j - n) \tag{2.6}$$

As is mentioned in [8] (Chapter 9), the convolution operation is commutative. Therefore, it can be written:

$$S(\boldsymbol{i}, \boldsymbol{j}) = (\boldsymbol{K} * \boldsymbol{J})(\boldsymbol{i}, \boldsymbol{j}) = \sum_{\boldsymbol{n}} \sum_{\boldsymbol{m}} I(i - m, j - n) K(m, n) \tag{2.7}$$

A similar implementation called **cross-correlation** as is presented in equation 2.8, and is implemented in most Machine Learning libraries. The difference between cross-correlation and convolution is that the kernel is flipped.

$$S(\boldsymbol{i}, \boldsymbol{j}) = (\boldsymbol{J} * \boldsymbol{K})(\boldsymbol{i}, \boldsymbol{j}) = \sum_{\boldsymbol{m}} \sum_{\boldsymbol{n}} I(i + m, j + n) K(m, n) \tag{2.8}$$

Therefore, the operation will start from the top-left of an image compared with convolution which starts the strike as is illustrated in figure 2.5. This example presents a convolution process of a matrix size 6x6 with kernel size 3x3 and stride equal to one. The operation starts from the top-left and calculating the element-wise product between the kernel and the overlapped area.

Step 1: 0*0+0*0+0*1+0*0+0*1+1*0+0*1+1*0+0*1 = 0

Step 2: 0*0+0*0+0*1+0*0+1*1+1*0+1*1+0*0+0*1 = 2

Step 3: 0*0+0*0+0*1+1*0+1*1+0*0+0*1+0*0+1*1 = 2

The final matrix for all steps is represented in the output matrix of figure 2.5. The output of this process is called **output feature map**.

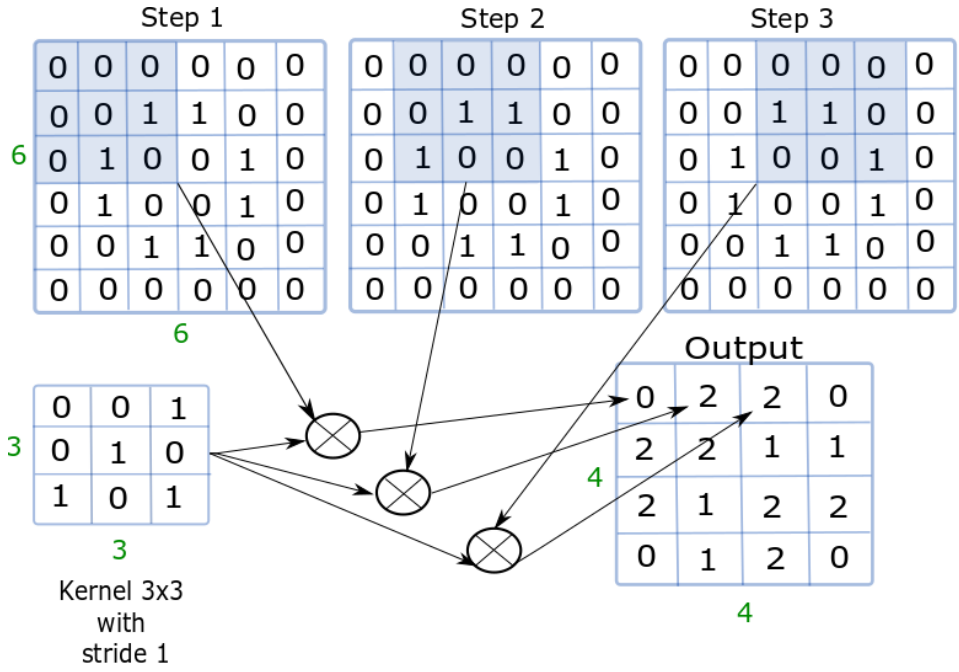


FIGURE 2.5: An example of a Convolution Operation

Furthermore, convolution operation allows to downscale in the input space. In this way, we can split the representation into smaller representations. The **stride** is responsible for the amount of that downscaling. The **kernel** is responsible for identifying some features, such as edges, angles, shapes and, and so on. During training, the multiple kernels are applied to learn different representations, as different kind of kernels that exist. Therefore, each kernel can contribute to different learning purpose.

Moreover, a typical CNN has three steps. The first steps consist of convolution operations, as we described above, where the result is a linear combination of activations. The second step refers to as detector step/layers and aims to break this linearity by using a nonlinear activation such as ReLU. And thirdly, there is a **pooling step**, where the size of the feature map is reducing. It can be thought as summarise the subregions of the feature maps. Pooling aims to create a representation that is invariance for input translation.

In other words, invariance means that outputs of pooling do not change when input is translated. For instance, it is useful in the case that someone wants to answer "if there is a dog in an image", without carrying so much on where exactly is the dog on the image. Moreover, invariance can be helpful when images present a rotated object. Most common used pooling operation is Max Pooling and Mean Pooling. More detailed information about CNN can be found [12].

2.3.3 Deep Residual Learning

Deep Residual Learning a state-of-the-art approach which aims to tackle the problem of training degradation, as the depth of the network is increased, and the accuracy of the training model is saturating for many challenging tasks such as object detection, image classification, and so on. Another way to think this problem is, as the Neural Network propagates the gradients thought all this layer and the layered complexity is increasing, the harder the propagation becomes because the derivatives of each parameter concerning the training loss start diminishing, also known as the **vanishing gradient problem**. In order to resolve the breakdown of deeper networks, He et al. [15] introduces **Deep Residual Learning**, which allows CNNs to have a

deeper structure of 100 to over 1000 layers. A residual unit that has an identity mapping is described by equation 2.10, where the input defined as $\mathbf{X}^{(l)}$, the output as $\mathbf{X}^{(l+1)}$ and $\mathcal{F}$ is the residual function.

$$\mathbf{X}^{(l+1)} = \mathbf{X}^{(l)} + \mathcal{F}(\mathbf{X}^{(l)}), \tag{2.9}$$

The intuition behind this approach is that instead of having only a stack of layers, an **extra shortcut** or **skip connection** is added before the activation to the final result, which allows explicit introduce the desirable (underlying) mapping, denoted as $\mathcal{H}(x)$, where x refers to the input of the initial layers. Therefore, each of these shortcuts or skip connections are referred to as **Residual Blocks**. In this way, multiple residual blocks can exist; these nonlinear residual block can adjust to other layers. The central hypothesis in the approach is based on that if more than one nonlinear layers can approximate a complex function, then can also approximate the **residual function**.

Furthermore, it is much easier to optimise the mapping of the residual $\mathcal{F}(x) = \mathcal{H}(x) - x$ than the initial mapping $\mathcal{F}(x) + x$. Moreover, both formulation should be able to approximate the asymptomatic wanted function. However, as is mentioned in [15], multiple nonlinear layers can affect the approximation of the identity mapping negatively. Residual learning aims to approximate the identity mapping by allowing to force the weight of the multiple nonlinear layers to zero. In this way, it is easier to approximate the function by using the reference to the identity mapping compared with approximating an unknown function. Moreover, a residual block is denoted as

$$y = \underbrace{\mathcal{F}(\mathbf{x}, W_i)}_{\substack{\text{residual} \\ \text{mapping}}} + x \tag{2.10}$$

The ResNet architecture adjusts every some stacked layers a residual block and is mainly inspired by [16]. Figure 2.6 illustrated two residual blocks, which are adjusted in a stacked of successive layers.

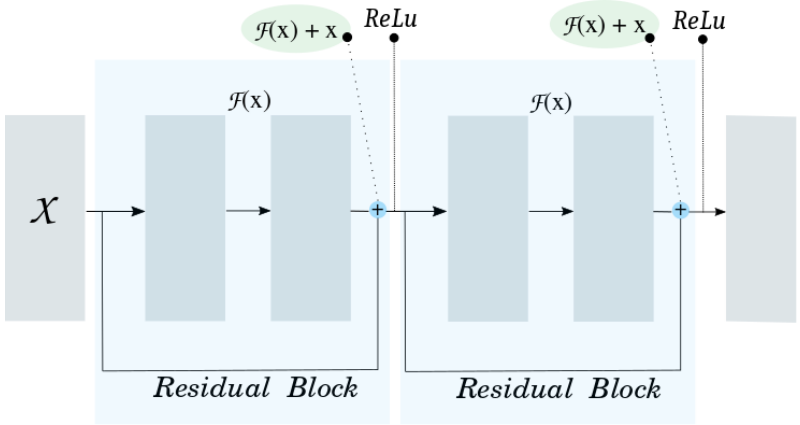


FIGURE 2.6: An example of two Residual Blocks

Overall, residual learning is highly successful as it is a state-of-the-art approach for learning deeper neural networks. In this project, both methods for the next timestamp prediction model and the anomaly detection model make use of residual blocks. More information about the implementation can be found in [15] and the source code [5].

[5]ResNet Github code: https://github.com/KaimingHe/deep-residual-networks

2.4 Maximum Likelihood Estimation

Maximum likelihood principle can derive functions which are good natural estimators. The intuition behind Maximum Likelihood is the estimation of a probability distribution by a model which is controlled by a $\boldsymbol{\theta}$ parameter. For instance, over a set of training examples $\boldsymbol{m}$, where $\mathcal{X} = (x^{(1)}, x^{(2)}, x^{(3)}, ..x^{(m)})$ which were individually drawn from the real data generating distribution $p_{data}(\boldsymbol{x}; \theta)$.

In this way, $p_{model}(\boldsymbol{x}; \theta)$ defined as a parametric distribution controlled by parameter $\boldsymbol{\theta}$; which describes where the data concentrated in the space. Moreover, we want to resolve the problem of $p_{model}(\boldsymbol{x}; \theta)$ estimating the true probability distribution by computing that probability over $\boldsymbol{x}$. Generally, the likelihood function is defined as $\mathcal{L}(\theta, \mathcal{D})$ and quantifies the compatibility of occupancy of $\mathcal{D}$ (referred to 2.3) based on any choice of θ.

$$\mathcal{L}(\theta, \mathcal{D}) = \prod_{i=1}^{m} p_{model}(x^{(i)}; \theta) \tag{2.11}$$

Additionally, Maximum Likelihood aims to estimate the parameters that are maximising the model's likelihood on the training dataset; same as the estimation of Normal Distribution (Gaussian Distribution). In the assumption that $\boldsymbol{x} \sim \boldsymbol{p_{\text{data}}}$ is distributed according to a Gaussian distribution, figure 2.7 illustrated an example of Gaussian Distribution.

$$\mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(x-\mu)^2 \Big) \tag{2.12}$$

where $\boldsymbol{\mu}$ refers to mean, and $\boldsymbol{\sigma^2}$ refers to variance as illustrated in figure 2.7. Moreover, log space simplifies the expression for the likelihood derivatives and "protect" of multiplying tiny probabilities together.

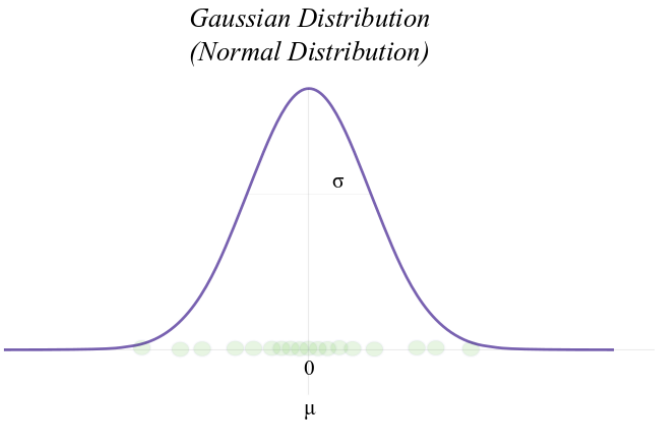


FIGURE 2.7: An example of Gaussian Distribution (Normal Distribution)

$$\boldsymbol{\theta}^* = \arg\max_{\theta} \prod_{i=1}^{m} p_{model}(x^{(i)}; \theta) \tag{2.13}$$

On the whole, it is possible to observe equivalent results more advantageously by taking the log of that likelihood and transforming equation 2.13 into a sum:

$$\boldsymbol{\theta}^* = \arg\max_{\theta} \sum_{i=1}^{m} log\, p_{model}(x^{(i)}; \theta) \tag{2.14}$$

Dividing by $\boldsymbol{m}$ results a criterion expressed as expectation over $\hat{p}_{data}$, refers to the empirical distribution determined from the training set.

$$\boldsymbol{\theta}^* = \arg\max_{\theta} \mathbb{E}_{\mathrm{x}\sim\hat{p}_{data}} log p_{model}(x;\theta) \tag{2.15}$$

The density function has to sum to one, which means that is not feasible to assign an infinite number of probability, where one points forces up the probability it unavoidably forces down in other positions. Equation 2.15 calculates the maximum density function for all the data in the space. Summarising topics of Maximum likelihood:

- An equivalent to Maximum likelihood estimation is minimising KL-divergence 4.6.
- An alternative form of Maximum likelihood can be obtained by taking the logarithm of the likelihood 2.14.

2.5 Density Estimation

In machine learning, **Density Estimation** [6] problems are required to learn a function $P_{model} : \mathbb{R}^n \rightarrow \mathbb{R}$. In the case that **x** is a continuous random variable, the probability distribution is described using **probability density function**. On the other hand, in the case of $\boldsymbol{x}$ is a discrete random variable, the probability distribution is described as **probability mass function**, in the space from which the examples originate. **Probability density function** does not provide a state-specific probability; instead of, it provides the probability of landing over a region with infinity tiny possible values.

Chapter 4, describes Generative Models, which can be performed as density estimation in a collection of points and allows to explicitly capture the distribution, which is intractable computationally in practice for the required operations on $P(\boldsymbol{x})$; as mentioned in [8]. Chapter 4.6 describes Generative Adversarial Networks, which use a different approach that implicit capture the distribution, in a way that improves the performance. More specifically, the next two subsections are discussed some of the main differences between explicit density and implicit density models.

2.5.1 Explicit Density Models

Explicit (prescribed) **probabilistic models** define a **log-likelihood** [7] function controlled by parameter θ. Also, the distribution is determined based on an explicit parametric specification of $\boldsymbol{x}$, where $\boldsymbol{x}$ refers to some observed random variable. Most models in Machine Learning appear in this explicit estimation of log-likelihood.

The category of explicit density models are unambiguous in the meaning of calculating the Maximum Likelihood, and the model's definition that refers to the density function is merged within likelihood expression. A limitation of explicit density models is the difficulty of design a model that can catch the whole complexity of the data distribution and preserving computational tractability. As is described in [4], there are two main approaches to deal with that. Under the first approach, there are models which are aimed to through their structure to be **computationally tractable**.

An example, of this models, are **Fully Visible Belief Networks** [8] (**FVBNs**), but even if these models aim to tackle the problem of capturing the complexity of the distribution by using the

[6] **Density estimation** is a process based on observed data, where the goal is to find an approximation or estimation of that data.

[7] Commonly, log-likelihood function $log q_\theta(p_x)$ is referred to Maximum Likelihood Estimation

[8] **Fully visible belief networks** refer to a special case of sigmoid belief networks [8], where there are no latent variables, therefore, there is no need for marginalised hidden - latent variables beyond the limits of the likelihood.

chain rule for decomposition of a probability distribution from a high dimensional vector to a product of one dimensional, it is still computational expensive as sample generation allows just one entry at time. Under the second approach falls the category of **Nonlinear Independent Components Analysis** (**NICA**), where the density p_x is tractable, in the case that both p_z and the **jacobian**[9] is tractable. This approach requires the same shape for both p_z and p_x. Overall, even though explicit density estimation can be highly effective, it is difficult to have a tractable density and therefore, Explicit Density Models are limited.

Furthermore, an additional implementation which makes use of approximations density in order to maximise the likelihood falls under the category of either variation approximation (deterministic approximation) or stochastic approximation. Section 4.5 describes **Variational Autoencoders**, one of the most popular approaches of variational learning.

2.5.2 Implicit Density Models

Opposed to explicit (prescribed) models, Implicit Density Models can generate samples by defining a stochastic procedure. More specifically, implicit Generative Models aim to avoid density intractability by using likelihood-free inference. Likelihood-free inference goal is to compare generated samples with true distribution samples. Next, given the information on the comparison, the generative model is training by adjusting the parameter. This process is known as **Density Estimation by comparison**. There are four ways of dealing with the problem of density estimation by comparison, as mentioned in [13]:

1. **Class-probability estimation** based model distinguish between observed and generated samples by creating a classifier. This density-based method is used for learning implicitly with generative models such as Generative Adversarial Networks (GANs).

2. **Divergence minimisation** measures the divergence between true and generated samples. It makes use of a function called f-divergence such as KL-divergence to compel the learning of the generative model.

3. **Ratio matching** directly minimises the difference between estimation and true density using **Bregman divergence** $\mathcal{B}_f$. Bregman divergence (distance) aims to measure the difference between two points. Furthermore, these points can interpret as probability distributions over some observed data. In that way, Bregman divergence results in a statistical distance.

4. **Moment matching** is equivalent to evaluate the equality between the moments of the true samples and the generated samples.

The objective of implicit density is to approach the data distribution, not in a direct way but to learn how to "draw" the distribution. Generative adversarial networks (GANs), as we will see in section 4.6, are designed to deal with the limitation of **Fully visible belief networks** (FVBNs) and generating in parallel, as is described in section 2.5.1. Additionally, GANs allows the use of higher dimension of p_z compared with NICA, as mentioned in section 2.5.1. Furthermore, Boltzmann Machines make a use of Markov chain methods for both training and generating samples. However, for multi-step Markov chain, the cost of producing samples becoming higher. Generally, Markov chain models fail to map to high dimensional spaces. Overall, GANs were introduced to tackle these problems such as the intractability of Markov chains, the usage of a variational bound of Variational Autoencoders (VAEs), and so on.

[9] **jabobian** refers to both squared matrix and its determinant (source: Wikipedia)

Chapter 3

Deep Spatio-Temporal Prediction

This thesis intends to combine both prediction and anomaly detection model, as previously mentioned in section 1. In this way, we can predict the next timestamp of all the cells, and use the anomaly detector on the predicted images as a threshold for the true values of a specific timestamp that we are evaluating, similar to figure 1.4. Furthermore, these models can be combined to predict potential failures in the system.

- The first section 3.1 and 3.2 is based on [17, 18]
- The second section 3.3 is based on [19]
- and the last section 3.4 is based on [20]

3.1 Formulation of the Spatial-Temporal Prediction Problem

ST-ResNet [18] and DeepST [17] use two definitions according to the formulation of the spatial-temporal prediction problem. The first one defines location concerning different levels of details and the semantic meanings, whereas the second definition is according to the measurement objective for regions. For instance, crowd flows, air quality and vehicle rent can be some of the measurements for regions. Moreover, in both studies city was divided into a grid map of $I \times J$, where each grid denotes a region based on longitude and latitude. Second, they choose to use crowd flows as measurement, where the movement of crowds is referred to as $\mathbb{P}$, which is a collection of trajectories [1] at a specific timestamp. Then two types of crowds are calculated, inflow and outflow, for a grid-region and for each k^{th} timestamp by using the following two formulas, respectively.

$$x_t^{in,i,j} = \sum_{Tr \in \mathbb{P}} |\{k > 1 | g_{k-1} \notin (i,j) \wedge g_k \in (i,j)\}| \tag{3.1}$$

$$x_t^{out,i,j} = \sum_{Tr \in \mathbb{P}} |\{k > 1 | g_k \in (i,j) \wedge g_{k+1} \notin (i,j)\}| \tag{3.2}$$

where g_k is the geospatial coordinate, Tr is a trajectory in $\mathbb{P}$. All the inflow/outflow for every interval in a grid $I \times J$ is denoted as $X \in \mathbb{R}^{2 \times L \times J}$. Overall, the problem description is to predict X_n given historical observation X_t where $t = [0, ..., n-1]$.

3.2 ST-ResNet

ST-ResNet [18], focus on the prediction of flow crowds in different regions of a city. This is important as the traffic management of a city is crucial for keeping a "healthy" network for the public safety of the citizens. Similarly, the objective of this project is to predict the status of the network and check if there are any unwanted events. Therefore, predicting the next timestamp of the network can secure that the network always remains stable and preventing adverse events by checking if the predicted state of the network is similar with the actual state of the network. Therefore, ST-ResNet can be used for predicting the network flow and assessing the behaviour of the network in such a way that the networks remain stable.

[1] Trajectories of crowds lies within the map grid $I \times J$.

Due to the complexity of the prediction, as it is affected by multiple factors such as events, internal region traffic and external factors such as weather, weekdays, weekends, and so on. ST-ResNet proposes a residual NN framework to handle spatial-temporal data.

Moreover, ST-ResNet [18] splits the time to **closeness**, **period** and **trend**. Each of these properties belongs to tree-based residual convolutional units, which enables the model to aggregate these properties of closeness, period and trend. Respectively, these aggregated values are merged with *exterior factors*, such as the day, timestamp, the weather conditions, and so on. ST-ResNet is used to predict feature states of each part of the city and prevent any potentially dangerous situations.

3.2.1 Prediction of Flows

The paper splits the predictions into two flows called **inflow** and **outflow**. Inflow refers to the total traffic entering a region for each time interval. On the other hand, outflow denotes the traffic that is leaving a region during an interval. The transition of flow is tracked from both inflow and outflow. They measure inflow/outflow by available data such as pedestrians, the number of cars, and so on. Is important to mention that mobile signals and GPS trajectories can be used for measuring pedestrians and vehicles, respectively, as mentioned in [18].

3.2.2 Simultaneously Forecasting

There are three main factors of complexity that affects the forecasting of inflow and outflow, **Spatial dependencies**, **Temporal dependencies** and **External influence**. Spatial dependencies aim to learn near and far dependencies between regions. In other words, it tries to identify regions that are affected not only from nearby regions but also from those which are distant. Temporal dependencies can occur at different time intervals during the day.

For instance, in morning hours many people go to work, therefore, it is expected that area in some particular regions like at the centre of the city it will be more crowed compared with other regions which are at a distance with the centre. The degree of influence that each region has can be varied by the closeness, period and trend.

On the other hand, it is expected for regular workdays to occur specific workflows in different regions, but there are cases where external factors such as weather, holiday events, metadata data like weekday/weekend can affect the behaviour of the crowds. ST-ResNet aims to accomplish the respective prediction of inflow and outflow, proposes a **Deep Spatial-Temporal Residual Network**, which makes a use of **Residual Unit** as is described in 2.3.3.

3.2.3 ST-ResNet Model

The model first transforms the inflow and outflow into a 2-channel image matrix. Next, it separates the images into three bits, according to the time axis which refers to recent (closeness), near (period) and distant (trend) time. The three parameters refer to time axis are $\mathbf{l}_c$, $\mathbf{l}_p$ and $\mathbf{l}_q$ accordingly. Each of them returns a tensor $\mathbf{X}_c$, $\mathbf{X}_p$ and $\mathbf{X}_q$, which has shape accordingly with the parameters in table 3.1.

For instance, if we consider the parameter of closeness as is depicted in figure 3.1, then the shape of the tensor will be $\mathbf{X}_c * \mathbf{f}$ where f is the number of the flow. Is important to mention that in ST-ResNet implementation, $\mathbf{l}_p$ refers to periodicity of a single day, whereas $\mathbf{l}_q$ refers to one week, which describes the trend in that week.

TABLE 3.1: ST-ResNet - Training Parameters

ST-ResNet	
Parameters	**Values**
Batch Size	32
Intervals	48
Learning Rate	0.0002
$\mathbf{l}_c$: *closeness*	{3,4,5}
$\mathbf{l}_p$: *period*	{1,2,3,4}
$\mathbf{l}_q$: *trend*	{1,2,3,4}
Map height	32
Map width	32
flows	2
L: Residual Units	12

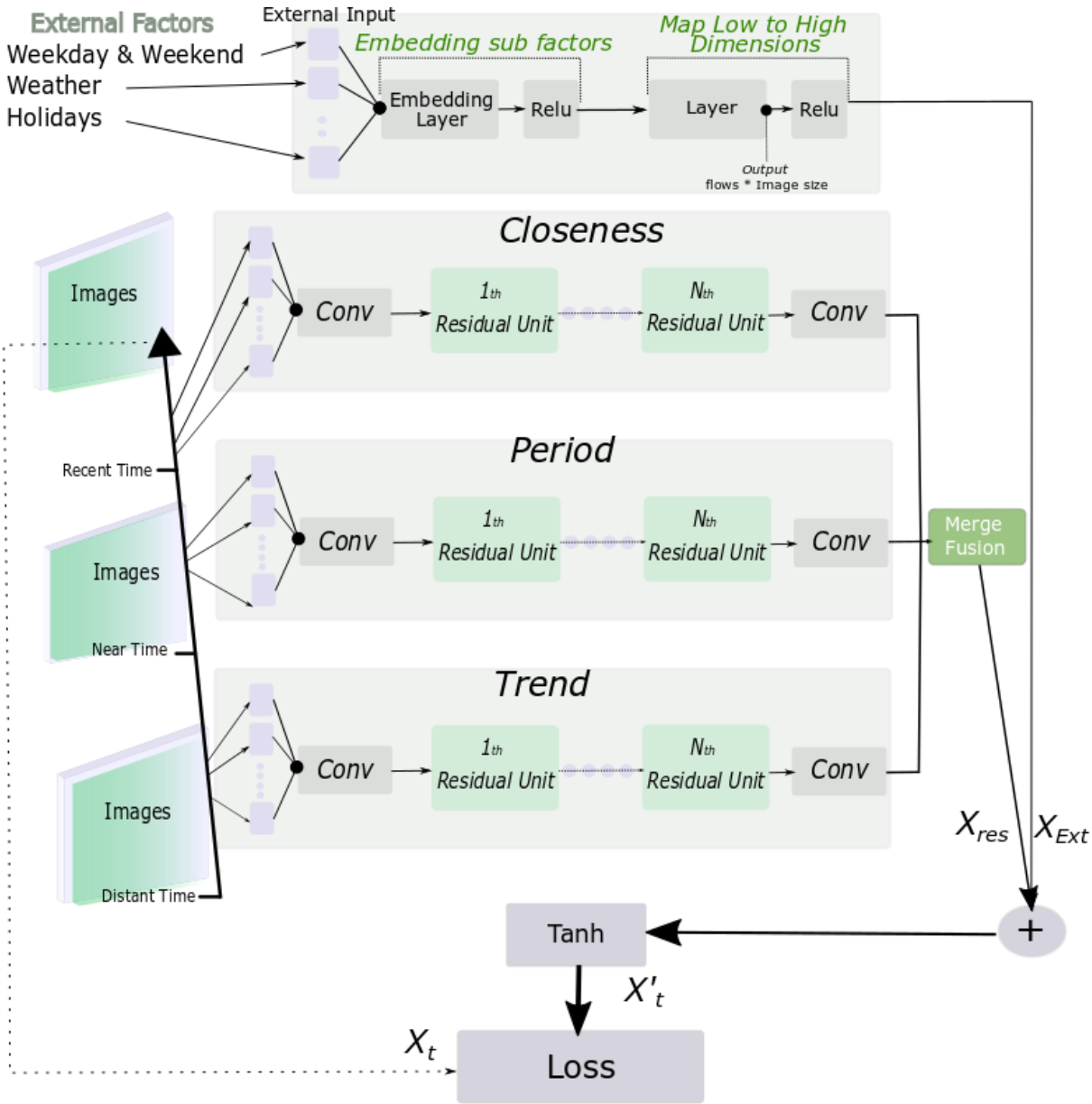


FIGURE 3.1: The Spatial-Temporal (ST-ResNet) - Architecture

3.2.3.1 Capture Close and Far Away Region Dependencies

Furthermore, the model makes use of three (3) components, which are fed accordingly. These components share the same structure, as illustrated in figure 3.1, which starts with a CNN and continues with residual blocks in order to capture the spatial dependencies between close and far distance. Finally, each of those components have a second CNN, which has as an input the residual output. The result of each of those components is $\mathbf{x}_c^{(L+2)}$, $\mathbf{x}_p^{(L+2)}$, $\mathbf{x}_q^{(L+2)}$ accordingly for **closeness**, **period** and **trend** where L is the number of residual units for each component and the number two refers to the number of the CNN. Finally, the model merged all the components by using a fusion function, which is an element-wise multiplication of the result of each component with the learned parameters. The operation is denoted by

$$\mathbf{X}_{Res} = \mathbf{W}_c \circ \mathbf{X}_c^{(L+2)} + \mathbf{W}_p \circ \mathbf{X}_p^{(L+2)} + \mathbf{W}_q \circ \mathbf{X}_q^{(L+2)} \tag{3.3}$$

3.2.3.2 External Components

Additionally, the model implements external factors, where it feeds them into a two-layer fully-connected Neural Network, as is illustrated in 3.1. The first layer is embedding all the sub-factors together follow by using **Relu** as an activation function. Moreover, the output of the activation is mapped into a higher dimension which has output the same as X_t. The output of the external components are referred as $\mathbf{X}_{Ext}$.

3.2.3.3 Fusion Region Dependencies & External Factors

Also, both region dependencies $\mathbf{X}_{Res}$ and external factors $\mathbf{X}_{Ext}$ are merged.

$$\hat{\mathbf{X}}_t = tanh(\mathbf{X}_{Res} + \mathbf{X}_{Ext}) \tag{3.4}$$

Finally, the predicted value 3.4 is used during the training, where the model minimising the **Root Mean Square Error** (RMSE) between predicted and true data matrix, as is denoted in equation 3.5, where $\boldsymbol{\theta}$ is referred to the learned parameters.

$$\mathcal{L}(\theta) = \|\mathbf{X}_t - \hat{\mathbf{X}}_t\| \tag{3.5}$$

3.2.4 DeepST

DeepST [17] is a prior Deep Learning approach for prediction based on Spatial-Temporal data due to the massive data generation of wireless communication technologies. Additionally, the architecture consists of three components, temporal **closeness**, **period** and **trend**. Moreover, DeepST aims to predict in real-time crowd-flows. This architecture was proposed before ST-ResNet 3.2, as is shown in table 7.2. Moreover, as ST-ResNet architecture outperforms DeepST and as ST-ResNet is a continuation of DeepST, from some authors of the DeepST implementation, this thesis focus only on ST-ResNet implementation. For more details about the DeepST implementation, refer to [17].

3.3 DMVST-Net

Deep Multi-View Spatial-Temporal Network for Taxi Demand prediction (DMVST-NET) [19] is based on taxi demand prediction such as STDN [21], proposed for both spatial-temporal relationships. Conventional methods are mostly based on time series forecasting techniques. These methods luck to model the non-linear material relation between spatial and temporal data. In this work, they used LSTM to handle temporal relation independently compared with traditional methods which consider only spatial relation using Recurrent Neural Networks. The model consists of three different view aspects, temporal, spatial and semantic view. The first one is modelling similarities between future and close-time points. The second one models

the spatial connection using Convolutions Neural Network, whereas the last one models the correlation between region sharing identical temporal patterns.

The intuition of DMVST-Net is to predict the requested number of taxis for a specific region during a timestamp in the future. DMVST-NET implementation consists of both CNN and LSTM for capturing both spatial and temporal complex dependencies. The authors claimed that including region for the prediction with little correlation to predicts hurts the performance of the model. Therefore, they used CNN for only nearby spatial regions. However, as this fails to capture the case that two far-away regions can have similar patterns, similarly discussed for ST-ResNet 3.2. The authors proposed a region graph which captures the semantic properties, where an edge of this graph refer to the similarity of demand pattern for two regions located far away from each other. Next, context features are created by a graph embedding model which combined vectors of the region, which are followed by a fully connected NN to predict the next timestamp.

Furthermore, as remote regions harm the model prediction, as was mention in [21]. Moreover, methods such as ST-ResNet, do not model the sequentially temporal dependencies, even if historical traffic images used for the prediction. In this way, the proposed model combines a Deep Learning model for both spatial and temporal sequentially chronicle relations.

3.3.1 DMVST-Net Framework

DMVST-Net framework consists of taxi requests defined as a tuple of timestamps, locations and identification number. Additionally, demand was defined as the request for each particular location over a time point, given the locations, referred to as L for 48-time intervals per day (similar to ST-ResNet). In our project, the taxi requests over one location can be considered as the PM counter over a specific location. Overall, the demand aims to predict the next timestamp $t+1$ given the historical information until $t+1$. Furthermore, external factors called context features can be applied as vectors referred to a specific location L and t time. The prediction function is defined as:

$$y_{t+1}^{i} = \mathcal{F}\ (\underbrace{\mathcal{Y}_{t-h,\dots,t}^{L}}_{\substack{\text{Historical}\\ \text{demand}}},\ \underbrace{\epsilon_{t-h,\dots,t}^{L}}_{\substack{\text{context}\\ \text{features}}})\ , \tag{3.6}$$

where, i denotes a location L and $t-h$ the initial time interval. The proposed framework consists of three view parts:

- **The Spatial view**, which is a local CNN and aims to capture close distance regions, where a location with the close neighbour's regions is treated as one image, where $\boldsymbol{i}$ denotes the centre of the image. The output of this model is a reduced spatial representation denoted as $\hat{s}_t^i$, where t refers to time and i to a region.

- **The Temporal view** consists of a Long Short-Term Memory (LSTM) model, which allows modelling sequential data, and has been proposed to tackle limitations such as vanish gradient of Recurrent Neural Networks (RNN). The way that LSTM work is by having a memory cell referred to as c_t, which is a collection of preview historical information. Furthermore, the LSTM make a use of a forget gate denoted as $\hat{f}_t^i$, where during the training the network erases or forgets some of the learned memories. Moreover, the output of the LSTM is controlled by an output gate denoted as $\hat{o}_t^i$, where the output of the LSTM is defined as:
$$\boldsymbol{h}_{t+1}^{i} = \boldsymbol{o}_{t+1}^{i} \circ tanh(\boldsymbol{c}_{t+1}^{i}) \tag{3.7}$$

- **The Semantic view** aims to capture locations which experience relevant functionalities and might have relevant demand patterns. The intuition is that the centre area might have higher demand during the morning hours compared with areas far away from the

centre. In this way, the likeness between regions is an embedded feature vector $\boldsymbol{m}^i$ passed to a fully connected layer to combine it with the whole network, defined as $\hat{\boldsymbol{m}}^i$. Last, the input to the LSTM consists of spatial representation $\hat{s}_t^i$ linked with the context features e_t^i.

$$\boldsymbol{q}_{t+1}^i = \boldsymbol{h}_{t+1}^i \otimes \boldsymbol{m}^i \tag{3.8}$$

Final, the **final prediction** $\boldsymbol{y}_{t+1}^i$ at time interval $t+1$ given the input data until time interval t is a combined of all three parts, defined by equation 3.8, passed to a fully connected NN. Overall, the boundaries of the model are defined by a **sigmoid** function used for the final layer.

$$\mathcal{L}(\theta) = \sum_{i=1}^{N}((y_{t+1}^i - \hat{y}_{t+1}^i)^2 + \gamma(\frac{y_{t+1}^i - \hat{y}_{t+1}^i}{y_{t+1}^i})^2) \tag{3.9}$$

The lost function is defined by equation 3.12. Additionally, for algorithm evaluation, the paper used Rooted Mean Square Error (RMSE) and Mean Average Percentage Error (MAPE).

3.4 Spatial-Temporal Dynamic Network - STDN

A Deep Learning model for traffic prediction, close related to ST-ResNet [18] and DMVST [21] is called Spatial-Temporal Dynamic Networks (STDN) [21]. The intuition of the model relies on that spatial-wise location dependency that is dynamic, which means that can be changed from period to period, and that different patterns appear during weekdays and weekends. More specifically, STDN models aim to handle both **temporal dynamics** and **spatial dependencies**s.

STDN [21] aims to handle spatial dependencies as dynamic as these dependencies can vary over time. In other words, historical period patterns can have some perturbations from different chronological periods, which means that they might follow periodical patterns which are not strictly periodic. To address the above problems, STDN [21] achieves to learn dynamic similarity between different regions by a flow gating mechanism using a CNN model. Additionally, a periodically shifted mechanism aims to control periodically extensive temporal changes by using a LSTM model.

Moreover, the STDN was implemented with a similar dataset as ST-ResNet. The city was split to a grid map $n = a \times b$, where n refers to a region. The traffic was defined as the movement of a starting region $y_{i,t}^{start}$ to the final region i for that time t. The formulation of the traffic prediction is similar to both ST-ResNet and DMVST-Net.

3.4.1 STDN Framework

The proposed framework consists of:

- Local Spatial Dependencies and **Flow Gating Mechanism** (FGM), where local CNN networks aim to capture the spatial trajectories. STDN frameworks, following DMVST-Net intuition, where regions with low correlation might hurt the prediction performance, where the regions of interest and the surrounding regions treated similarly, as it was described in 3.3. The final representation of region after passing the input through k convolution layers is denoted as $\boldsymbol{Y}_{i,t}^k$. However, local CNN reflects the historical traffic volume similarity, where the spatial reliance is stationary, which is not directly referred to the correlation among objective regions and its neighbours. On the other hand, traffic flow is another more straightforward way to represent correlation, where a **Flow Gatting Mechanism** (FGM) has been used for explicit capture of the information among the regions. Similarly, with ST-ResNet, the traffic flow was divided in **inflow** and **outflow**, where both were combined to an array, denoted as $\boldsymbol{F}_{i,t} \in \mathbb{R}^{S \times S \times 2l}$, where the number of flows per time interval is referred as $2l$. The spatial interaction was captured by a CNN with input the

combined array of flows for k layers, each of these layers returns a spatial representation, which is adjusted from the flow rate, denoted as $\boldsymbol{Y}_{i,t}^{k}$

- Short-term Temporal Dependencies and **Periodically Shifted Attention Mechanism** or **PSAM**. STDN address the vanishing gradient problem of long-term prediction models by explicitly model the same interval of a different day. However, traffic input is not exclusively periodic, as was mention in [20]. In this way, the authors propose PSAM, whose main focus is to address daily periodically shiftings. STDN tackles temporal shifting by choosing additionally time internals close to the target predicted time, referred to as Q. Specifically, the picked one hour before and after the target prediction timestamp, where LSTM is used by having as inputs the spatial representation combined with external factors and the previews combined information, in this way, $\boldsymbol{h}_{i,t}^{p,q}$ refers to the representation of preview day interval q to predict timestamp t in a region i.

Moreover, for each of the selected timestamp, the sum of the weight of the preview representations is defined by

$$\boldsymbol{h}_{i,t}^{p} = \sum_{q \in Q} \alpha_{i,t}^{p,q} \boldsymbol{h}_{i,t}^{p,q}, \tag{3.10}$$

where $\alpha_{i,t}^{p,q}$ refers to the "worth" or the importance of a specific time interval q by assessing the spatial-temporal representation learned by LSTM with preview stages $\boldsymbol{h}_{i,t}^{p,q}$. Final, another LSTM is used to maintain the periodical input representation. Therefore, the output for the target final interval is denoted by

$$\hat{\boldsymbol{h}}_{i,t}^{p} = LSTM(\boldsymbol{h}_{i,t}^{p}, \hat{\boldsymbol{h}}_{i,t}^{p-1}) \tag{3.11}$$

Furthermore, during training, both short $\boldsymbol{h}_{i,t}^{p}$ and long $\boldsymbol{h}_{i,t}^{p}$ term representation was combined, in order to denoted as $\boldsymbol{h}_{i,t}^{c}$. Last, the final prediction value determined by feeding the $\boldsymbol{h}_{i,t}^{c}$ to a fully connected layer, where the results for the initial and the last traffic quantity for each region is denoted as $y_{start,t+1}^{i}$ and $y_{end,t+1}^{i}$ accordingly with boundaries $[-1, 1]$. Moreover, the loss function used for training is defined as:

$$\mathcal{L} = \sum_{i=1}^{n} \lambda((y_{i,t+1}^{s} - \hat{y}_{i,t+1}^{s})^2 + (1-\lambda)((y_{i,t+1}^{e} - \hat{y}_{i,t+1}^{e})^2, \tag{3.12}$$

where actual data are denoted as $\hat{y}_{i,t+1}^{s}$ and $\hat{y}_{i,t+1}^{e}$ and λ is a constant that determines the influence between initial and final trajectory. Last, MAPE and RMSE evaluation methods used, similar to DMVST-Net.

Overall, DMVST-Net and STDN proposed that treating the whole city as an image and applying CNN might do not result in the best performance. Moreover, by having regions with low correlation might hurt the performance of the network. ST-ResNet uses CNN to capture the spatial information and overlooks the temporal sequential dependency. Therefore, both models DMVST-Net and STDN outperform ST-ResNet. However, in our implementation, ST-ResNet used as both DMVST-Net and STDN GitHub implementations were limited.

Chapter 4

Deep Generative Models

This section focuses on Deep Generative models exclusively and more specifically on Generative Adversarial Networks (GANs), which is the "heart" of this thesis. In this way, It is important to overview what Generative models are and what Generative models are aim to accomplish before broaden deeper in GANs. Therefore, this chapter goals is to provide a comprehensive overview of Generative Models and more specifically of Generative Adversarial Networks [1], where the last section of this chapter describes evaluation measurements based on GANs.

- Sections 4.1 - 4.6 is based on [8, 22, 23]. Moreover, section 4.6 - Generative Adversarial Networks is based on [4, 5, 8]
- Section 4.7 - DCGAN is based on [24]
- Section 4.8 - Wasserstein GAN is based on [25]
- Section 4.9 - Wasserstein GAN with Gradient Penalty (WGAN-GP) is based on [26]
- Section 4.10 - GANs Evaluation is based on Pros & Cons of GAN Evaluation Measures [6].

4.1 Introduction

In a network where normal and abnormal behaviour occurs, it is sometimes impossible to define precisely anomaly behaviours. Nevertheless, even if we assume that labelled data exist and an anomaly event happens, it might be **unknown**. In the case of an unknown anomaly, our model is not capable of detecting the abnormality as it is an undiscovered anomaly.

Unsupervised Learning [27] is an effective way to learn some data distribution by using Deep Generative Models [8, 28]. Moreover, Generative Models can be used for **Representation Learning** without any label data, which is commonly expensive and time-consuming for large companies dealing with **big data**. Furthermore, Deep Generative Models have been used in various applications such as image generation, speech generation, music synthesis generation, language generation, and anomaly detection [2, 3].

The intuition behind the generative models follows the quote of Richard Phillips Feynman, who received the Nobel Prize in Physics in 1965.

"What I Cannot Create, I Do Not Understand" -Richard Phillips Feynman

In this way, Generative Models aim to **"understand"** which features are important, and therefore how to generate them. More formally, the intuition behind generative models is such a tough training, that the model determines the semantic properties of the data distribution, which are generated. Hence, Generative Models are capable of learning the **underlying distribution** [2] of the normal data, based on the reconstruction error of the behaviour pattern. We can

[1] Note: Renom.js tutorial section was used as an external source for further assistant on GANs, DCGAN, WGAN, WGAN-GP and AnoGAN.

[2] The **underlying distribution** is a distribution of data which is associated with the data generative process. The data generative process means that an input has been generated by a probability distribution over a dataset. [8]

make use of that reconstruction error in order to identify an anomaly. Thus, the model can learn a manifold of the natural network patterns and use it as an anomaly detector.

Mainly, **Generative Models** aim to learn generated samples $p_{model}(x)$ analogous to the distribution $p_{data}(x)$ of the training data. As it is mentioned in [4], Generative models can be divided into two main categories. Generative Models which aim to explicitly describe and solve $p_{model}(x)$ by using **explicit density estimation** 2.5.1, whereas other Generative Models aim to sample from $p_{model}(x)$ by performing **implicit density estimation**, as is mentioned in 2.5.2.

Before explaining in more detail Generative Adversarial Networks (GANs), it is essential to explain the reason that GANs are currently so successful. In order to accomplish that, we have to explain some limitations of other approaches. Most commonly, we could have a NN which estimates the maximum likelihood of input space X explicitly and learns $P_\theta(x)$.

$$\int_x P_\theta(x)dx \tag{4.1}$$

However, equation 4.1 can be normalised to be equal to one, but even if that is a way to constrain it, it is intractable for higher spaces such as images.

4.2 Autoregressive (AR) models

A way of dealing with normalising constant in the probability distribution being intractable is by using a class of models called **Autoregressive** (AR). AR models factorise the distribution dimensional-wise. Precisely, in the case of an image, we can split the likelihood using chain rule into a one-dimension product 4.2, where each pixel is conditioned on all previous pixels.

$$p(x) = \prod_{i=1}^{n} p(x_n|x_1, \ldots, x_{n-1}) \tag{4.2}$$

In this way, the density estimation is tractable. Furthermore, those models need to be in order when it generates pixels. An example of an application called PixelRNN, which starts from the top-left and then sequentially generates pixels based on preview pixels. Because of this sequential process, the process is quite slow. Also, another approach that starts the generation, in the same way, is called **PixelCNN**. PixelCNN, instead of using an RNN model, it makes use of a CNN model which aims to approximate the next pixel value over surrounding pixels via a softmax loss.

AR models suffer in a real-time generation as too many forward passes through the NN causes slow generation. Furthermore, AR models do not learn what is vital in the input space, as there are only based on pixels. For instance, in network anomaly detection, we would like to generate images referred to a specific timestamp; therefore, we would like to learn some high-level characteristics.

4.3 Autoencoders

Autoencoders aims to learn a representation of the input by compressing the essential characteristics of the input space. Moreover, Autoencoders can learn this representation without any explicit labels. Autoencoders consists of an **encoder** which compresses the input and a **decoder** which aims to reconstruct the original data, where figure 4.1, illustrates the autoencoder structure. The encoder has to learn a lower-dimensional feature representation z. The bottleneck separates critical aspects of the data by balancing between compressibility and retaining variables that are behaviorally related.

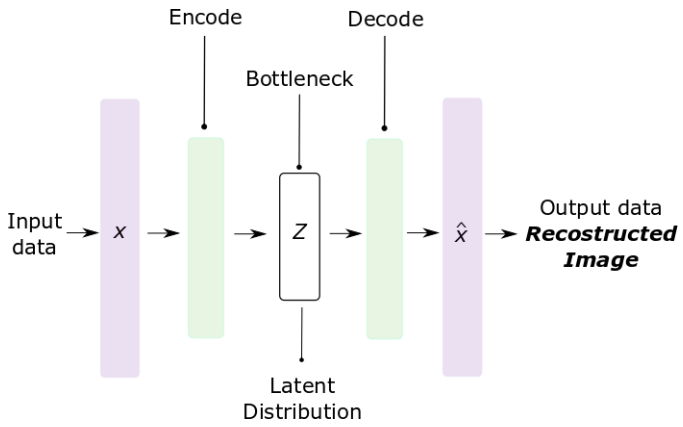


FIGURE 4.1: Autoencoder (AE) - Architecture

The encoder can be a CNN, which aims to compress an image to z representation. Next, the decoder makes use of a deconvolution where the inverse operation used to go from the low-dimension z to the original-dimension of the input. Moreover, the model can be trained by comparing the **Mean Square Error** (MSE) between inputs and the outputs.

$$L = (x - \hat{x})^2 \tag{4.3}$$

Overall, Autoencoders cannot interpret as pure Generative Models by using equation 4.3 loss, because the network is encouraged to compression rather than form an idea of feature representation. Therefore, our initial goal is to be able to sample from a model to generate data, to do that a different approach called **Variational Autoencoder** is introduced, where the idea of autoencoders twists to a probabilistic Autoencoder.

4.4 Kullback-Leibler (KL) Divergence

Probability theory allows measuring the difference between two probability distribution by using **f-divergence**. A specific group of Autoencoders called Variational Autoencoders make a use of a special case of f-divergence called **Kullback–Leibler divergence** (KL- divergence), also referred to as a ***relative entropy***. Therefore, before introducing Variational Autoencoders, it is essential to present KL- divergence.

As is mentioned in [8] to determine the uncertainty in the entire probability distribution someone can use the Shannon entropy in (also known as $\boldsymbol{H(P)}$):

$$H(x) = -\mathbb{E}_{x\sim P}[logP(x)] \tag{4.4}$$

Furthermore, in the case of having a variable x over two probability distribution $P(x)$ and $Q(x)$ we can compare the difference between those two distributions using Kullback-Leibler (KL) divergence:

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}[log\frac{P(x)}{Q(x)}] = \mathbb{E}_{x\sim P}[logP(x) - logQ(x)] \tag{4.5}$$

Also, KL divergence is non-negative, and when is equal to zero, the distributions are similar to discrete variables and nearly similar in the case of continuous variables. Additionally, even if KL-divergence seems that measure the distance between distributions, is not symmetric $D_{KL}(P||Q) \neq D_{KL}(Q||P)$, which means that taking the reverse distribution will result in different outputs. Later we will see that some Generative Models suffer from the limitation of choosing the right divergence as they learn to focus only on generating well a part of the input

space. Therefore, the models do not generalise well as it learns to generate only a part of the input distribution; this limitation is referred to as **Mode Collapse**.

A similar approach to KL divergence is the cross-entropy denoted in equation 4.6, which is equal to equation 2.14 (Maximum Likelihood Estimation). Moreover, we can achieve it by minimising the cross-entropy with respect to Q:

$$H(P,Q) = H(P) + D_{KL}(P||Q) = \underbrace{-\mathbb{E}_{x\sim P} log Q(x)}_{\text{minimise}} \tag{4.6}$$

More specifically, we can minimise the difference between $\hat{p}_{data}$ and p_{model}:

$$D_{KL}(\hat{p}_{data}||p_{model}) = \mathbb{E}_{x\sim\hat{p}_{data}}[log\hat{p}_{data}(x) - log p_{model}(x)] \tag{4.7}$$

As $\hat{p}_{data}$ do not refer to the model, we only have to minimise the model distribution $-\mathbb{E}_{x\sim\hat{p}_{data}}[log p_{model}(x)]$, in the same way as equation 4.6.

4.5 Variational Autoencoders (VAEs)

The idea of Variational Autoencoders removes the idea of the reconstructing loss, which outputs a restricted distribution and maximises the likelihood. Variational Autoencoders build a model that looks like an Autoencoder by assuming that the input data $\mathcal{D} = \{x_i\}^N{}_{i=1}$ is generated from some underlying unobserved distribution z such as Gaussian (normal) distribution.

In the case of probabilistic models, which works with the joint probability of both x and z, when we want to generate data, we can sample by taking the conditional probability equation 4.8, but for high-dimensional data the interval $p(x)$ (equation 4.1) of the posterior density is intractable.

$$p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(z|x)} \tag{4.8}$$

Explicit density estimation makes use of **approximation inference** (variational inference). VAE introduce an additional approximation inference network (encoder) which aims to encode $q_\varphi(z|x)$ and approximates $p_\theta(x|z)$. The additional encoder allows deriving a **lower bound** of data likelihood that is tractable. In other words, this approach transforming an inference problem to optimisation. Figure 4.2 illustrates the structure of a VAE probabilistic network.

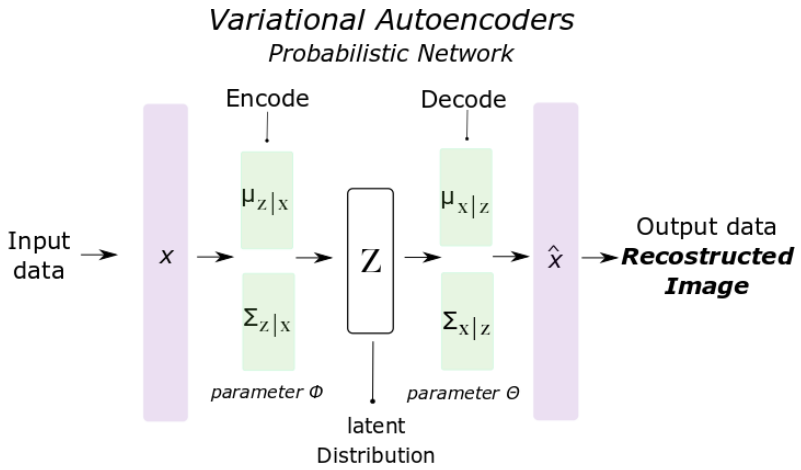


FIGURE 4.2: Variational Autoencoders (VAE) - Architecture

We can sample this density function as we would do with a typical Generative Model even though the structure is similar to Autoencoder architecture. Accordingly to this, encoder with parameters φ and decoder with parameters θ networks will sample from distributions $q_\varphi(z|x) \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$ and $q_\theta(x|z) \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$. Importantly, as randomly sampling from a distribution does not allow to backpropagate gradients, VAE makes use of a **reparametrization trick**, where a new parameter ϵ allows to reparametrized z to backpropagate.

Probability theory allows measuring the difference between two probability distribution by using f-divergence. Variational Autoencoders make a use of a particular case f-divergence called **Kullback–Leibler** divergence, as described in 4.4. Because of the unknown posterior probability, Variational Autoencoders are trained by maximising $\mathcal{F}(q)$, which is the **variational lower bound**, also known as Evidence Lower Bound (ELBO).

$$\underbrace{\mathbb{E}_{z\sim q_\varphi(z|x)} log p_\theta(x|z) - D_{KL}(q_\varphi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}))}_{\mathcal{F}(q)} \leq log p(x) \tag{4.9}$$

Overall, even if Variational Autoencoders tends to be an excellent way for manifold learning as encoder and decoder can parametrically be trained simultaneously to enable the encoder to capture the underlying distribution. VAE suffer in resulting blurry images, and that makes them inappropriate for anomaly detection compared with the state-of-the-art **Generative Adversarial Networks** (GANs) as will be discussed in the next section. Furthermore, more detailed information about VAE and a more in-depth explanation of the mathematics behind them, the reader can consult to [29].

4.6 Generative Adversarial Networks

Goodfellow et al. introduced **Generative Adversarial Network** (GAN) or Vanilla GAN in 2014 [5]. GANs has been used as a benchmark for a series of different approaches which targets different scientific areas, such as photo-realistic representation for computer games, motion in a video by reconstructing 3D models, recreate old videos with higher resolution, medical imaging for anomaly detection, and so on.

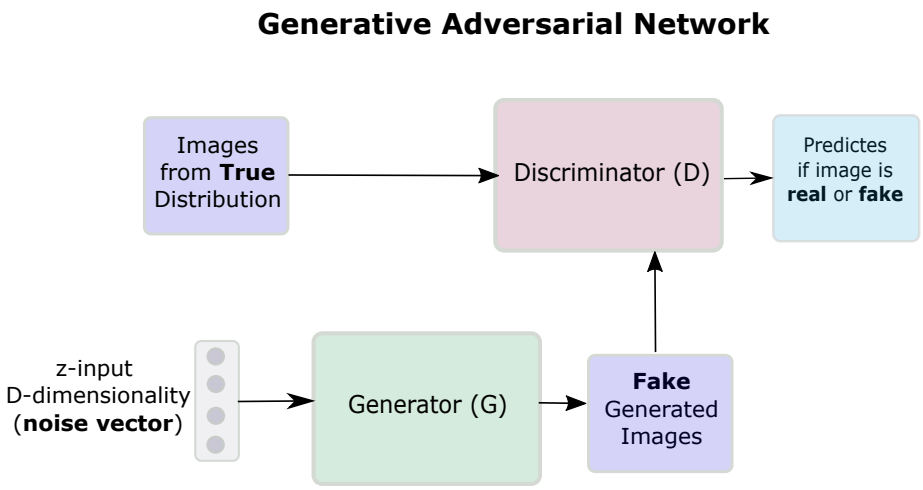


FIGURE 4.3: Basic Architecture of Generative Adversarial Networks.

GAN framework, as is illustrated in 4.3, consists of two parts, the **Generator** (G) and the **Discriminator** (D). The Generator (G), aims to augment the possibility of the Discriminator (D) making a wrong estimation. In other words, Generator (G) creates samples, close to the training distribution, which can fool the Discriminator (D). On the other hand, Discriminator (D)

determines if the samples are from the **latent**[3] or the **normal** (input) **space**. In other words, Discriminator (D) finds if the data are coming from the Generator (G) or not. As a result, Discriminator(D) can be used as a tool to identify real from fake samples.

4.6.1 Adversarial Nets Framework

Generative Adversarial Networks models use an **implicit** way for sampling over a probability distribution without explicit define a density function. The training process of GANs can be described as a **game theory** between Generator (G) and Discriminator (D) player. Moreover, the training process can be interpreted as a minimax game between the Generator and the Discriminator. Therefore, the relationship is **adversarial**, but during training, both models are **synergistic working**. In other words, Generator and Discriminator losses make a use of both parameters, but none of them cannot control other's parameters.

More specifically, Generator's loss $\mathcal{L}_{G_\theta}$ can update only $\boldsymbol{\theta_G}$ and Discriminator's loss $\mathcal{L}_{D_\theta}$ can update only $\boldsymbol{\theta_D}$. The stopping training process criterion is defined by the **Nash equilibrium**, where both $\mathcal{L}_{G_\theta}$ and $\mathcal{L}_{D_\theta}$ have to converge to a local minimum with respect to $\boldsymbol{\theta_G}$ and $\boldsymbol{\theta_D}$ respectively, where both Generator and Discriminator aim to learn a **differentiable function** [4].

On the one hand, **Generator** function $G_\theta(z)$ defines the **mapping to the data space**, where $z \sim p_z(z)$ following a predefined distribution. Therefore, Generators' objective is to learn to generate samples from the identical distribution as the true data $x \sim \boldsymbol{p_{data}}$ and fool the Discriminator. On the other hand, Discriminator can be though, as a classifier, where $D_\theta(x)$ is the probability that x is from the true training distribution and not from the generated. In this way, Discriminators' objective is to out approximate $D_\theta(x)$ to one when $x \sim \boldsymbol{p_{data}}$ and to approximate to zero when $D(G(z))$. The following equation presents a **minimax game** between Generator and Discriminator:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim \boldsymbol{p_{\text{data}}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{z\sim p_z(\boldsymbol{z})}[\log(1 - D(G(z))] \tag{4.10}$$

As mentioned in [5], due to confident of Discriminator rejecting Generators' samples early in the learning process, $log(1 - D(G(z))$ saturates. Therefore, the generator can be trained to maximise the logarithm $logD(G(z))$ instead of minimising the $log(1 - D(G(z))$. Furthermore, the minimax game has an optimal when $p_{data} = p_{model}$ and for a given G the optimal Discriminator is defined in equation 4.11; the proof can be found in [5], and it is essentially the fundamental key behind the approximation mechanism of GANs.

$$D^*_G(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{model}(\boldsymbol{x})} = \frac{1}{2} \tag{4.11}$$

Given D^*, we can rewrite the minimax game of equation 4.12 as

$$\begin{aligned} C(G) &= \max_D V(G,D) \\ &= \mathbb{E}_{x\sim p_{\text{data}}}[\log D^*_G(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D^*_G(G(z)))] \\ &= \mathbb{E}_{x\sim p_{\text{data}}}[\log D^*_G(x)] + \mathbb{E}_{x\sim p_{model}}[\log(1 - D^*_G(x))] \\ &= \mathbb{E}_{x\sim p_{\text{data}}}\left[\log \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{model}(\boldsymbol{x})}\right] + \mathbb{E}_{x\sim p_{model}}\left[\log \frac{p_{model}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{model}(\boldsymbol{x})}\right] \end{aligned} \tag{4.12}$$

[3] In Generative Adversarial Networks (GANs), **latent space** and **z-space** have the same semantic.

[4] **Differentiable function** refers in the existence of the derivative at each point in the domain space.(Source: Wikipedia)

Furthermore, the minimum of Generator $C(G)$ succeeds when $p_{data} = p_{model}$ and that is equal to $-log(4)$. We can simple prove that by applying $D^*_G(\boldsymbol{x})$ in equation 4.12

$$\begin{aligned} &\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log \frac{1}{2}] + \mathbb{E}_{x\sim p_{model}}[\log \frac{1}{2}] = \\ &= \log \frac{1}{2} + \log \frac{1}{2} = -\log(2) - \log(2) = -log(4) \end{aligned} \tag{4.13}$$

Then by subtracting equation 4.13 from $C(G)$ we get:

$$\begin{aligned} C(G) &= \mathbb{E}_{x\sim p_{\text{data}}}\left[\log \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{model}(\boldsymbol{x})}\right] + \mathbb{E}_{x\sim p_{model}}\left[\log \frac{p_{model}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{model}(\boldsymbol{x})}\right] \\ &= \mathrm{E}_{x\sim p_{\text{data}}}\left[\log \frac{\mathbf{2}p_{data}(\boldsymbol{x})}{\mathbf{2}(p_{data}(\boldsymbol{x})+p_{model}(\boldsymbol{x}))}\right] + \mathbb{E}_{x\sim p_{model}}\left[\log \frac{\mathbf{2}p_{model}(\boldsymbol{x})}{\mathbf{2}(p_{data}(\boldsymbol{x})+p_{model}(\boldsymbol{x}))}\right] \\ &= -\mathrm{E}_{x\sim p_{\text{data}}}[\log \tfrac{1}{2}] - \mathbb{E}_{x\sim p_{model}}[\log \tfrac{1}{2}] + \mathbb{E}_{x\sim p_{\text{data}}}\left[\log \frac{\mathbf{2}p_{data}(\boldsymbol{x})}{\mathbf{2}(p_{data}(\boldsymbol{x})+p_{model}(\boldsymbol{x}))}\right] + \mathbb{E}_{x\sim p_{model}}\left[\log \frac{\mathbf{2}p_{model}(\boldsymbol{x})}{p_{data}(\boldsymbol{x})+p_{model}(\boldsymbol{x})}\right] \\ &= -\log(4) + \underbrace{\mathbb{E}_{x\sim p_{\text{data}}}\left[\log\left(\frac{\mathbf{2}p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{model}(\boldsymbol{x})}\right)\right]}_{\text{KL -Divergence}} + \underbrace{\mathbb{E}_{x\sim p_{model}}\left[\log \frac{\mathbf{2}p_{model}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{model}(\boldsymbol{x})}\right]}_{\text{KL -Divergence}} \\ &= -\log(4) + KL\left(p_{data}||\tfrac{p_{data}+p_{model}}{2}\right) + KL\left(p_{model}||\tfrac{p_{data}+p_{model}}{2}\right) \end{aligned} \tag{4.14}$$

The two Kullback–Leibler divergences of equation 4.14 are equal to **Jensen Shannon divergence** (JSD) between two distributions. JSD is always greater or equal to zero $JSD\left(p_{data}||p_{model}\right) \geq 0$ and is equal to zero only when the model's distribution p_{mode} is equal to the true data distribution p_{data}. Therefore, the global minimum of equation 4.15 equals to $-log(4)$ only when the two probabilities are equal.

$$C(G) = -log(4) + JSD(p_{data}||p_{model}) \tag{4.15}$$

4.6.2 Training Process

The training process of the original implementations consists of **Stochastic Gradient Descent**, where there is a specific number of iterations, both **Generator** and **Discriminator** training in parallel. In a minimax game, the equilibrium can be though as finding a **saddle point** which is similar to minimising the Jensen-Shannon divergence.

The algorithm starts with a loop of K steps, where mini-batches are sampled from $z \sim p_z(x)$ noise prior distribution and $x \sim \boldsymbol{p_{\text{data}}(x)}$ data generating distribution. Next, the Discriminator is updated by Stochastic Gradient Ascending.

$$\max_D \mathbb{E}_{x\sim \boldsymbol{p_{\text{data}}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z))] \tag{4.16}$$

Additional, when the loop is finished, the algorithm resampling a mini-batch of m samples from $z \sim p_z(x)$ prior distribution and updating the generator by Stochastic Gradient Descent.

$$\min_G \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z))] \tag{4.17}$$

More information about the algorithm can also be found in the original implementation [5]. GANs are **asymptotically consistent**, which means if we find the equilibrium, the model

can successfully recover the actual data distribution. In practice it is challenging to train the models, as it might converge to a **local minimum** instead to a **global minimum** or the model will learn to represent only some parts of the training input which the model feels confident (Mode Collapse).

Furthermore, we would like to generalise successfully and learn even when samples are bad. However, in equation 4.17, the Generator performs best when the Discriminator outputs $D(G(z))$ is close to one and poor when is close to zero, which means that we favour the Generator when it is already good. To tackle this limitation, we are going to take the opposite direction, and instead of minimising Discriminator's likelihood of guess correct, we maximise the probability being wrong by updating generator using Stochastic Gradient Ascent.

$$\max_G \mathbb{E}_{z \sim p_z(z)}[\log(D(G(z))] \tag{4.18}$$

Now we have achieved the opposite results, which improves the Generator to learn more when the samples are still bad compared with equation 4.17. In general, many different approaches of GANs have been introduced by using different objectives. There are different approaches which suggest different updating techniques such as parallel updating of Discriminator and Generator, as is mentioned in [4]. However, there is no a specific rule which is the best approach.

Overall, Generative Adversarial Networks originally were not developed for large scale inputs even if the initial architecture of GANs makes use of convolution nets, DCGAN can scale GANs for generating higher scale output and better results.

4.7 DCGAN

Unsupervised Representation Learning with Deep Convolutional GAN (DCGAN) [24] has proposed a combination of CNNs and GANs for Unsupervised Learning. The model consists of DCGAN architecture improves the training stability compared with the initial GANs [5]. **DCGAN** architecture [24]:

- Replacing the pooling layers of the convolution with strided convolutions for the discriminator by jumping with stride higher to one and allows the networks to learn **spatial downsampling**. Second, the architecture introduces **fractional-strided** convolution for the Generator, which allows increasing the dimensionality by implementing null pixels among the original and opposite to discriminator to learn spatial upsampling.
- **Batch Normalisation** is used to stabilise the training process by normalising the activation. Additionally, prevent instability and learn the right mean and scale of the data distribution, the output of the Generator and the input of Discriminator do not make use of Batch Normalisation. Overall, Batch Normalisation improves the model generalise well and avoid mode collapse.
- **ReLU** is used as am activation function for each of its layers except the output (Tanh activation) of the Generator network. The Discriminator network consists of **leaky Relu** activation, which performs well with high-resolution modelling in constraint with vanilla GANs which make use of max out activation.
- Using **Adam** as an optimiser with learning rate equals to 0.0002 instead of the default 0.001 and **Momentum** equal to 0.5 instead of the default 0.9, results in higher stability.
- Removes hidden, fully connected layers.
- Training via SGD with mini-batch size equals to 128 by normalising all the weights from a normal distribution with zero-centred and standard deviation 0.2.

On the whole, **DCGAN** architectures offer a more stable way of training GANs. Also, both Generator and Discriminator can learn the hierarchy of representation of training input. DCGANs architecture has been used as an extension in many proposed implementations based on GANs such as AnoGAN [2].

Nonetheless, original GANs and the extensions such as DCGAN, which use the same formula such as loss, suffer from instability during the training, this is because during an initial state when the two distribution do not overlap and the discriminator does a good job there is no learning benefit. The ideal discriminator returns zero, forcing the gradient to become extremely small and therefore during backpropagation no learning taking place for the generator. An alternative divergence has been proposed, called **Wasserstein GAN** to tackle the limitation of JSD divergence.

4.8 Wasserstein GAN

As indicated in [25], original GANs are challenging to train; an alternative approach called **Wasserstein GANs** (WGAN) has been proposed to tackle some of the limitations of GANs such as Mode Collapse and Diminishing Gradient. WGAN make use of a different cost function called **Earth Mover's Distance** or **Wasserstein-1**, where **weight clipping** is introduced for approximating Earth Mover's Distance.

Additionally, WGAN keeps the same idea for both Generator and Discriminator. However, one difference is that the Discriminator acts more like a **critic** than a binary classifier (Discriminator) as the sigmoid of the last layer has been removed. Overall, WGAN provides a smoother distance measure compared with GANs, even when the distribution of the model and the real data do not overlap, which is not the case with GANs.

Contrary, Wasserstein-GAN tackles the difficulty of GANs models balancing the training, which requires a more approached design of the network architecture. Moreover, Wasserstein-GAN architecture reduces Mode Collapse problem and allows to compute the Earth Mover (EM) distance continuously during training. The intuition of the Earth Move distance is denoted by the numerous piles of soil, which have different weights (values). The goal is to move those weights in order to pair a target distribution. The Wasserstein distance is defined as

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\left[||x - y||\right] \tag{4.19}$$

FIGURE 4.4: An example of Wasserstein-GAN Transport Plan.

More specifically, in equation 4.19, the initial distribution is denoted as $\mathbb{P}_r$ and the target distribution as $\mathbb{P}_g$. Also, the Wasserstein distance between $\mathbb{P}_r$ and $\mathbb{P}_g$ is defined as the **infimum** or known as the **greatest lower bound**, where Π between $\mathbb{P}_r$ and $\mathbb{P}_g$ indicates the collection of all join distributions γ.

To sum up, Π consists of all the transport plans where $\gamma(x, y)$ refers to the "*soil mass*" that must be moved from x to y in order to convert $\mathbb{P}_g$ into $\mathbb{P}_r$. Therefore, WGAN results the transport plan with the **lowest cost** of all $\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)$. Figure 4.4 presents an example of a transport plan, where the purple area is shifting to the right in every step until it matches the target distribution. Moreover, it is worth to mention that different transport plans might exist at different costs. Furthermore, by using equation 4.19 we can calculate each step of example 4.4 as

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sum_{a=-\infty}^{\infty} \delta_a = \delta_{Step1} + \delta_{Step2} + \delta_{Step3} + \delta_{Step4} \tag{4.20}$$

$$W(\mathbb{P}_r, \mathbb{P}_g) = ||(3+0)-0|| + ||(0+3)-2|| + ||(4+1)-4|| + ||(0+1)-1|| = 3+1+1=5 \tag{4.21}$$

where $\delta_a = ||(\mathbb{P}_r(a) + \delta_{a-1} - \mathbb{P}_g(a)||$.

However, equation 4.19 is extremely **intractable**, as is mentioned in [25]. In this way, the authors used an alternative approach called **Kantorovich-Rubinstein duality** to approximate the Earth Mover's distance denoted as

$$W(\mathbb{P}_{data}, \mathbb{P}_{model}) = \frac{1}{K} \sup_{||f||_L \leq K} \mathbb{E}_{x \sim \boldsymbol{p_{\text{data}}}}[f(x)] + \mathbb{E}_{x \sim p_{model}}[f(x)], \tag{4.22}$$

where **supremum** or **least upper bound** of k-Lipschitz functions for a fixed parameter K. By transforming the non-tractable problem to a tractable problem and by assuming that we have a family of k-Lipschitz functions $\{f(x)\}_{w \in \mathcal{W}}$ parameterised by w, we can find the optimal f* which equals to trained network over optimal parameter w^*. Therefore, the WGAN loss function is defined as

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim \boldsymbol{p_{\text{data}}}}[f_w(x)]] + \mathbb{E}_{\boldsymbol{z} \sim p(\boldsymbol{z})}[f_w(g_\theta(z))] \tag{4.23}$$

In order to guarantee the K-Lipschitz is conditioned on f_w, the authors proposed **Weight clipping** which enforcing the K-Lipschitz by clipping the weights into a range [-c,c] after every update of the optimiser. Furthermore, another difference that is compared with vanilla GANs is that WGAN formula does not have a sigmoid activation anymore; this reflects the fact that equation 4.23 does not make use of logarithm term anymore. Additionally, WGAN proposed updating discriminator five times and then updating the Generator one time, in contrast to [4], which suggested updating both Generator and Discriminator equally.

As it is shown in [25], WGAN algorithm uses RMSPropm, which works as an optimiser for both generator and critic(discriminator), and for each critic iteration, it is clipping the weights of the critic $\boldsymbol{w \leftarrow clip(w, -c, c)}$. Additionally, WGAN Earth mover's distance tackles Mode Collapse and eliminates the gradient vanishing compared with vanilla GANs. Furthermore, with vanilla GANs, we do not have a clear way of measuring the performance of generated images, as GANs loss can measure how well the generator fools the discriminator but it does not measure direct the generated samples. Moreover, as we will see in section 4.10, there are alternatives ways to assessing **quantitatively** and **qualitatively** the GAN model.

In contrast, WGAN keeps the Discriminator in convergence, and by optimally estimating the EM distance during training acts as an indicator of the performance of the model as is closely correlated with the observed data, figure 4.5 presents the Wasserstein GAN architecture.

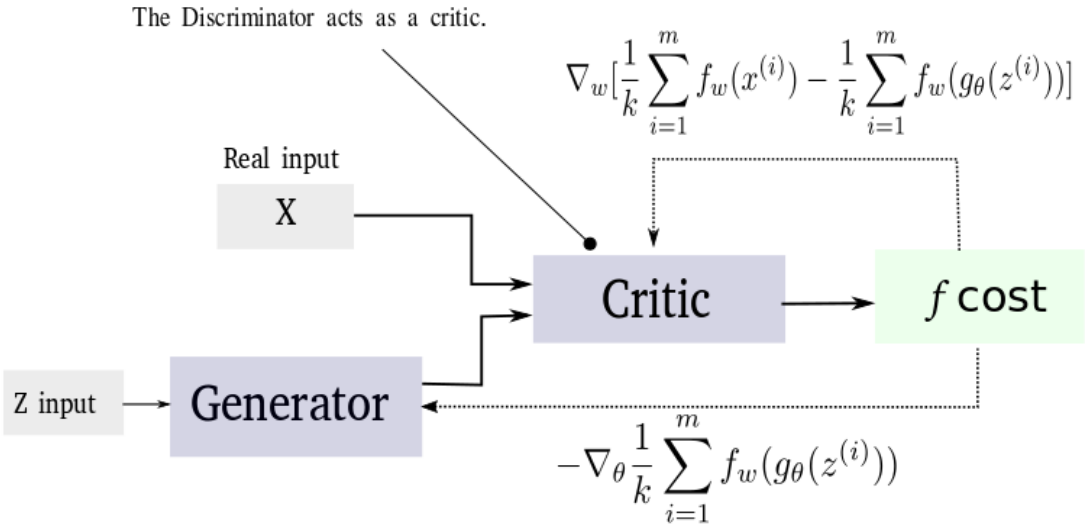


FIGURE 4.5: Wasserstein Generative Adversarial Network (WGAN) - Architecture

Overall, WGAN can be unstable for different clipping windows as it can be unstable for higher constants or it can vanishing its gradient for smaller constants. Next section, describes an improvement of WGAN proposed by Gulrajani et al.[26], which replaces weight clipping with gradient penalty.

4.9 Wasserstein GAN with Gradient Penalty (WGAN-GP)

As is mentioned in [3], weight clipping "*is a terrible way to enforce Enforcing Lipschitz*". For instance, when Weight Clipping is used on the Discriminator (critic) which can often lead to undesirable results as poor images or even converge failures, in the case that c hyperparameter is not tuned correctly. Improved Training of Wasserstein GANs [26] leverages these problems by introducing a **penalised** Discriminator norm of the gradient, concerning its inputs. The proposed method outperforms WGAN and introducing a more stable training process without the need for careful hyperparameter tuning.

Furthermore, improved training of Wasserstein GANs [5] has shown the optimisation difficulties by comparing different weight constraints and soft constraints which result in the same problematic response. A k-Lipschitz constraint implementation by using weight clipping leads to simpler functions. Those simpler functions lead to approximate elementary models which defining the optimal function and ignore higher moments of the actual distribution. Furthermore, without cautiously tuning of $\boldsymbol{c}$ hyperparameter, the gradient is at risk of exploiting or vanishing. The improved method consists of a soft constraint adaptation by keeping the original Discriminator (critic) cost and implementing a gradient penalty. The proposed method is based on that only if a differentiable function f has a gradient with norm no greater than one; then it is 1-Lipschitz. In this way, if f^* is differentiable, it holds equation 4.24, where $\boldsymbol{x_t = tx + (1-t)y}$ with $\boldsymbol{0 \leq t \leq 1}$ and $\boldsymbol{\pi(x = y) = 0}$. More details about the proof can be found in [26].

$$\mathbb{P}_{(x,y)\sim\pi}\left[\nabla f^*(x_t) = \frac{y - x_t}{||y - x_t||}\right] \tag{4.24}$$

Additionally, due to the intractability issue of enforcing everywhere the norm of the unit gradient constraint, as it was mentioned in [26], a soft constraint version used to penalise the

[5] Code for the WGAN-GP implementation: https://github.com/igul222/improved_wgan_training.

model on the gradient norm when it moves away from the objective norm. The improved version of WGAN is given by

$$L = \underset{\tilde{x}\sim\mathbb{P}_g}{\mathbb{E}}[D(\tilde{x})]] - \underset{x\sim\mathbb{P}_r}{\mathbb{E}}[D(x)]] + \underbrace{\lambda \underset{\hat{x}\sim\mathbb{P}_{\hat{x}}}{\mathbb{E}}[||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2]}_{\text{Gradient Penalty}} \tag{4.25}$$

where λ is a weight penalty factor, where $\lambda = 10$ used for the experiments in [26]. Moreover, below is given a more detailed explanation of the right-most part of the equation 4.25.

- The random interpolated image denoted as $\hat{x} \leftarrow \epsilon x + (1-\epsilon)\tilde{x}$, where ϵ is a random number sampled informally $\epsilon \sim U(0,1)$.
- The generated output denoted as $\tilde{x} \leftarrow Generator_\theta(z)$, where $z \sim p(z)$ is the prior distribution on the input noise z.
- The L2 norm on the Gradient is denoted as $||\nabla_{\hat{x}} D(\hat{x})||_2$

Batch Normalisation has been used for stabilising the training process for prior implementations such as vanilla GANs, DCGANs and WGAN. However, Batch Normalisation affects the mapping of the input to the output as it causes association among samples from the same batch. Because WGAN-GP individually penalises for each of its input the norm of critic's gradient, it replaces Batch Normalisation of the critic with **Layer Normalisation**. Overall, WGAN with gradient penalty has been already implemented for anomaly detection [3] as we will discuss more in section 6.6.

4.10 GANs Evaluation

Even if a wide variety of GANs has been introduced, there is not a robust way to **evaluate** GANs as different methods make uses of different objectives, whereas a variety of evaluation methods aim to assess the model **quantitatively** others seeks to assess the model **quantitatively**. However, in some cases due to the limitation of the objective function, these evaluation methods might bias models, which seems that performed well, but they have only learned a small section of the input space. For instance, some GANs tends to learn to discover a small variate of the input space and favour the Generator to generate samples that the Generator is comfortable. On the other hand, the quality measurement of the generated samples can differ for what humans would perceive and judge.

A recent research by Borji [6], aims to evaluate the **advantages** and **disadvantages** of Generative Adversarial Networks, separates the measure of GAN in two main categories, **quantitative** and **qualitative** measurement evaluation. Most of the research has focused on the quantitative measurement with 24 different measures, and 5 for qualitative analysis, as mentioned in [6]. GAN evaluation is an active research area, due to the limitation of this master thesis, it is not feasible to analyse each of these methods in detailed. However, in this section, we will discuss some of these methods briefly. For more information about the measurement evaluation methods, the reader can refer to [6]. Measures based on quantitative analysis:

- **Average Log-likelihood** aims to proffer an explanation of real-world data by using density estimation. More specifically, it measures the likelihood of the true distribution under the generated, as was mention in [6]. Moreover, with average log-likelihood is not feasible to answer questions such as when GAN model has memorised the true data distribution or if the model has learned to represent the whole space of the true distribution. Overall, **Coverage Metric** is claimed to be more interpretable compared with Average Log-likelihood, where a metric is used which use the true mass probability of the input data coated with the P_{model}.

- **Inception Score** (IS) [30] is one of the most **commonly used** measurement method, which aims to seek both diversity and quality properties of the generated samples. Inception score measures the Kullback–Leibler divergence among marginal $p(y)$ and conditional distribution $p(y|x)$, where high entropy for marginal denotes higher diversity among the classes, and a low entropy for the conditional distribution denotes the ability to assign correct labels to the generated examples. However, the Inception Score has some limitation. For instance, IS can memorise parts of the training examples and can be unable to identify overfitting. Second, IS takes into account only the generated samples, in this way IS can be fooled for another model which can generate similar samples. Moreover, IS is not **asymmetric measurement** and can also be influenced by the input size of the image. On the other hand, a **modification of IS** (m-IS), which introduces a cross-entropy aims to improve the diversity among the samples from the same class. Furthermore, **mode score** introduces the empirical distribution, which is not considered in the initial IS, but has been proved that mode score and IS is equivalent, as is mentioned in [6]. Last, AM scores suggesting that when the data distribution of the samples are not equally distributed, the entropy $\mathcal{H}(x)$IS is not any more suitable. Therefore, AM score introduces the label of the training dataset by using Kullback-Leibler (KL) divergence between y_{train} and y combined with the entropy of the prediction.

- **Fréchet Inception Distance** (FID) [31] takes the feature space by using a layer of generated samples. Then FID measures the distance between the true samples and the generated by summarising the statistics as multivariate Gaussian and calculating the mean and the covariance image-size. Therefore, the Fréchet distance or the Wasserstein-2 distance can **measure the gap** between **real** and **generated** image and quantify the quality fo the generated images, while IS and AM score measures the diversity and the quality for only the generated images.

- **Maximum mean discrepancy** (MMD), similar to 4.8, measure the likeliness of two distribution, the real P_r and the generated P_g. Moreover, kernel MMD measure the square of MMD over some defined kernel function such as Gaussian. Kernel MMD can be used for the training purposes similarly as 4.8.

- **Birthday Paradox Test** calculates the support [6] size of a discernible distribution by enumerating the duplicates. As mentioned in [6], it is not possible to find precisely the duplicates. Therefore, L_2 norm used for finding near-duplicates. On the whole, this approach is feasible for detecting GANs mode collapse.

- **Classifier Two-sample Tests** (C2ST) aims to figure whether two samples are sampled from the identical distribution by using a binary classifier, where different binary classifiers can be used. An example based on C2ST is called 1-Nearest Neighbor classifier, which computes the leave-one-out accuracy (LOO) accuracy.

- **Classification Performance** is a technique to use a linear model to evaluate the performance over an unsupervised feature extractor of a labelled dataset. For instance, as is mentioned in [6], we could assess a trained DCGAN model qualitatively, by using the trained feature of the Discriminator on ImageNet[7] to train a linear L2-SVM [8] for classifying CIFAR-10[9]images.

- **Boundary Distortion** aims to measure the difference between a generated samples and covariate shift, in which the probability mass of the generator is located on some modes of the true distribution, using classifiers.

[6]The **support** of a true value function is a subgroup of the domain which includes the elements which have not been mapped to zero (Wikipedia)

[7]ImageNet is a dataset with approximately 15 million annotated images.

[8]For more information about linear Support Vector machines refer to [32]

[9]CIFAR-10 is a 32x32 pixel size dataset with ten different classes, where each class contains 6000 images.

- **Number of Statistically-Different Bins (NDB)** measure the diversity of both generated images and mode collapse. The method compares the number of samples in a specific bin, where the assumption is that two set samples from the identical distribution should be the same as noise sampling. NDB measurement is applied directly on a pixel-wise level in contrast with IS. Therefore, an advantage of NDB is that it allows assessing images pixel-wise.

- **Image Retrieval Performance** aims to identify samples in the training set, which have been modelled from the network unsuccessfully.

- **Generative Adversarial Metric (GAM)** is a combination of two GANs, which are competing each-other by shifting Discriminators or Generators and computing the likelihood ration between these models. However, there are two limitations with GAM due to the restriction of the performance for both Discriminators as the defined by a constraint and as there are many swapping between the Generator and the Discriminator the computational cost is increasing. Furthermore, one approach based on GAM is called Generative Multi Adversarial Metric (GMAM) [33], where the inspiration of GAM and GMAM called **Tournament Win Rate**, and **Skill Rating** and aims to place each player either as Discriminator or as Generator. The result of this tournament is defined by two tournament win rate and skill rate.

- **Normalised Relative Discriminative Score (NRDS)** this measurement method is closely related to [34], where the central idea is that whether the generated samples are close to the true distribution, more epoch would be needed in order to discern the fake from the real samples. The discriminator network shares its weights with a test Discriminator network and for each different generated model that is comparing with. The decrement rate is calculated as the area under the curve $A(C_i)$, and the final NRDS is the final decrement rate over all the preview $A(C_i)$.

- **Adversarial Accuracy and Adversarial Divergence** aims to use two different trained classifiers to estimate $P_r(y|x)$ and $P_g(y|x)$. Next, both classifiers used for computing the accomplished the *adversarial accuracy* over a validation set and the *adversarial divergence*, which computes the KL divergence 4.4. Nonetheless, a limitation with this approach is that both real and generated images must be labaled.

- **Geometry Score** is a more sophisticated approach which is up to analyse the geometrical properties between the generated samples and the underlying data manifold. The detailed implementation can be found [35, 32] and the code [10], too. Furthermore, the paper presents a comparison between WGAN (4.8) and WGAN-GP(4.9) over one digit of MNIST 5.1 dataset, where WGAN with gradient penalty indicates a better performance. Additionally, the method proposed a combination with FID because geometry score considers only the topological properties.

- **Reconstruction Error** measures the reconstructed error, such as L2 norm, between true input data and generated. However, this approach is quite slowly as there is no exact reference to the z^* [11]. Therefore, the approach is slow as it starts by having zero vector and optimising the reconstructing error between generated and real input, which is not computed from the FFNN 2.3.1.

- **Image Quality Measures** aims to assess the quality of the generated samples by using

  - **SSIM** which is an individual scale measurement that aims not to consider features that are not perceptual interesting for the human eye. SSIM compares two images over

[10] The github of the **Geometry score** implementation: https://github.com/KhrulkovV/geometry-score

[11] z^* denotes the optimal z for a given x

luminance, *contrast* and *structure*. Next, the three quantities are merged to construct the SSIM score.

- **PSNR** which assess the quality of the generated sample by measuring the highest signal-to-noise ration between two black and white images, where high PSNR values denote better-generated samples quality.

- **Sharpness Difference** (SD) aims to assess the sharpness loss during the generation process.

- **Low-level Image Statistics** aims to evaluate the similarity of low-lever statistics based on that statistics remains the same when natural images are scaled, as is mentioned in [6]. A proposed method for evaluating GANs is proposed by Karras et al. [36], where for the assessing four statistics are involved, the mean power spectrum, the count of related components given an image, the distribution of arbitrary filter responses, the contrast distribution.

  A critical aspect of the proposed research is that even if the generated samples can be captured the distribution and results high quality, images generated by DCGAN, WGAN and VAE have shown that the generated images do not have scaled invariant magnitude for the mean power spectrum, which means that additional structures exist from the deconvolutional operations. Furthermore, it is feasible to use lower-level statistics to optimise the discriminator of a GAN network, assessing whether the statistics of the generated samples matched with the real data distribution. This method is known as **feature matching** [30], and it will be discussed more in 6.5.1.

- **Precision, Recall and F1 Score**, as mentioned in [6], Lucic et al. [37] proposed to quantitatively measure the extent of overfitting in Generative Adversarial Networks (GANs) by computing Precision, Recall and compute F1 score (7.3).

  - **Precision** intuition is to capture the quality of the generated images. Furthermore, the precision is determined as a part of the generated images, where the distance to the manifold is tested under some threshold.

  - **Recall** aims to evaluate the true distribution ration that has been covered from the learned distribution. Therefore, Recall is set as the part of the test data, where the L2 distance to the generated samples is below the threshold. Moreover, Inception Score (**IS**) does not penalise the GAN model when it does not learn all the variate of the modes in the training distribution. Therefore, it only captures the precision, as was mentioned in [37]. Furthermore, both Recall and Precision can be captured by *Fréchet Inception Distance* (FID) score. This approach uses a known data manifold, and the distance between the data manifold to the generated images is computing by inverting the generator as it learning the latent representation of z from all the input images. Overall, for generated samples close to the manifold, the *positive predictive values (PPV)* or *Precision* is high. Similar, a high score of Recall determines that the model can successfully recover the true distribution.

On the other hand, **qualitative** measurements by humans are one of the most common way of examining GANs qualitatively. However, visual examination is an expensive and a slowly process which is difficult to reproduce from the research community. Furthermore, for models which have learned to generate a part of the training dataset, is biased as a big number of sample evaluation is needed to be sufficient. Next, five qualitative measures are discussed, which aim to examine the generated samples over the learned latent space qualitatively.

- **Nearest Neighbors** presents generated samples close to the nearest neighbours beside the training input, where overfitting can be detected. However, there is a less significant perceptual disturbance due to that small changes to the images, which can lead to substantial

changes in the Euclidean distance that Nearest neighbour is usually based. Moreover, GANs that memorise the training set can fool the raters, where tackling this limitation is merely presenting more than one Nearest Neighbor.

- **Rapid Scene Categorisation** is a method which aims to characterise an image fake or real based on the fast perceptual ability of humans under some time constraints. Since this method is not well defined, as the critics are not specialised human raters, it makes it difficult to evaluate the model. Furthermore, this approach is more focusing on assessing the diversity of the generated samples. Overall, the evaluation might be biased in models that memorising the training set.

- **Rating and Preference Judgment** aims to ask participants to rate the truthiness of the generated images. In this way, experiments on multiple networks with different loss function can be used to determine whether the reconstruction the pixel-wise loss optimisation NN is better than a perceptually optimised NN. For more information about the experiments, the reader can refer to [38, 6].

- **Evaluating Mode Drop and Mode Collapse** aims to measure the mode by computing the distances among generated samples and modes centre. As is mentioned in [6], GANs failing to capture the whole input space, whereas being able to generate samples close to input space. **Mode collapse**, also known as **Helvetica Scenario**, generate the same samples for different latent variable z. Moreover, Mode Drop occurs when it is hard to represent some of the training inputs. Therefore, the model does not learn ambiguous inputs by ignoring them. Furthermore, by having a generated dataset, where both true training input distribution and modes are known, can be used to measure quantifiable the mode collapse. Furthermore, a similar approach, as is mentioned in [6], aims to examine how well the generated samples are distributed starts by having a well-balanced dataset with an equal number of images per class.

- **Investigating and Visualising the Internals of Networks** focusing on assessing the way that GAN learning and what precisely the NN is learning. In this way, this approach aims to investigate the scene of the latent space and the inward characteristics of the GANs.

  - ***Disentangled representations*** methods are focusing on measuring the interval representation and the existence of a semantic relationship between different direction in the latent space.
  - ***Space continuity***, close related to *disentangled representations*, the objective is to analyse the level of detailed that a model can capture, for different seed vectors z_i. Therefore, if both generated images are reasonable good on a linear joined representation between two vectors, then it is an indication the model captures the interval structure, without memorising the training input. White [39] has proposed a method for preventing deviation by replacing the linear interpolation with spherical interpolation.
  - ***Visualizing the discriminator features*** intuition is closely related to research on both representation and learned features from CNN, where Springenberg et al. [40] based on 4.7 have shown a learned feature hierarchy. Last, t-SNE [41] objective is to visualise the learned latent space z into a 2D representation, has been used in AnoGAN also to visualise the embedding of normal and abnormal images over the feature representation of discriminator's last convolution layer.

Overall, finding a suitable evaluating measure for GANs is a highly active area, in this section discussed an overview of the most recent evaluation method. Moreover, chapter 7 Experiment Setup and Results discussing evaluation methods that have been used in this project, where WGAN loss and F1-score used for quantitative assessing the model and investigation and visualisation of the internals of network used for evaluating the model qualitatively.

Chapter 5

Datasets

5.1 MNIST

MNIST is a dataset that contains grey-scale handwritten digits (from 0 to 9). MNIST has been used as a test baseline for many machine learning experiments. MNIST contains 60.000 examples of handwritten digits and 10.000 samples for testing. Furthermore, MNIST has a fixed size of 28x28 pixels for all the digit images. Because MNIST is a labelled dataset, it can be used for both supervised and unsupervised tasks. Therefore, one potential usage is to split the dataset into two categories; where some of the numbers refer to normal and some of them to abnormal (anomaly) behaviour. Next, we can use the normal numbers to train the network and then use the other number to test the anomaly detection model.

5.2 Converter Framework

This section introduces a python framework created for generating the input data used for the experiments. Furthermore, the input data consists of measurements that are captured from the networks for every 15 minutes interval. Moreover, each of these measurements refers to different locations in an area. This framework intends to convert all these measurements for different location into an image.

5.2.1 Performance Measurement Counters (PM Counters)

The number of traffic times events have occurred within a specified period referred to as Performance Measurement Counters. PM counters are used to generate Key Performance Indicators (KPIs). PM data are generated by the network and then stored as an XML file. Final, these data transferred to external databases, where can be fetched.

5.2.2 Data Preprocessing

The input data consists of multiple cells, which exist in different locations in a city. Moreover, each of these cells need to fetch the necessary information to generate the input data. Therefore, two programs have been created for both fetching and prepossessing the data. The *first program fetching* the data and the *second program (converter)* used to *check* the fetched data and then *generate* the input image for each day. Last, there is a functionality that checks for missing data and then reports those which are missing to be fetched again. In this way, we ensure that all of the images for the whole period consists of the same amount of cells. Figure 5.1 illustrates the steps that are taken to generate the images.

5.2.3 Fetch Data

First, R-Studio used to fetch the data. More specifically, the R program has a query which starts with a loop for all the desired cells which exist in a particular area. Next, it fetches the PM counters data for a period. In other words, the cell folder contains for each day a CSV file. Then, each of these files includes all the PM counters for that cell for 96 timestamps. Overall, in this project, 1230 cells were fetched.

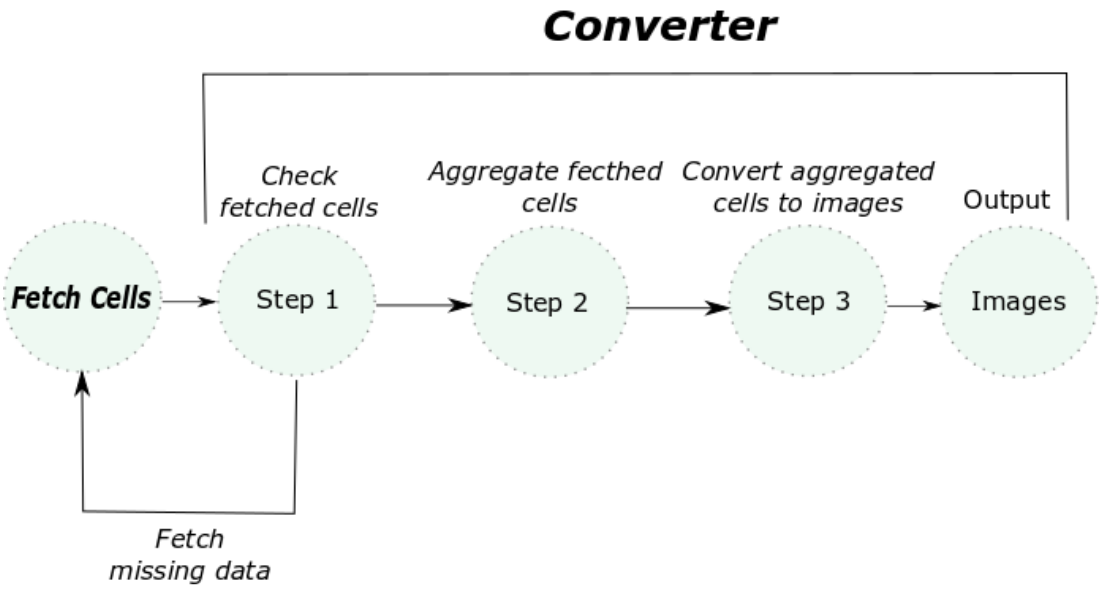


FIGURE 5.1: Fetching and Converting Process

The process of generating images consists of two phases. The first phase consists of the fetching cells. Then the second phase includes the converter, which splits into three steps. The first step is for checking the fetched cells, and it results in both information about correct and missing cells. In the case of missing cells, it re-runs the process of fetching only for the absent cells. The second steps aggregate all the cells per day and per timestamp. Therefore, inside each of the days-folder, there 96 CSV files (for 15 min interval per day), that contains all the cells information. The final step generates for each day 96 images.

5.2.4 Convert Data

The converter was created in Python and contains three main functionalities. The first one is for testing the fetched data; the second one is for generating the CSV files to prepare the needed information for the converter to produce the images. The last functionality creates the input images, where **Object-Oriented Programming** (OOP) methods have been used for expansibility, reusability, simplicity, maintainability, and so on. In other words, it is not required to run all the steps every time to generate images. It is possible to run the last functionality, which produces images by reusing the aggregated files.

Furthermore, a setting JSON file is created to improve the flexibility to work with different functionalities. In the setting file, it can be set various settings to activate different features. For instance, the start-end date and the number of fetched days to be converted. The file type that the converter will read and aggregate the cells. Furthermore, the three steps are shown in figure 1, can be activated or deactivated by setting true or false. Moreover, parameters can be set there, such as plot image during the generation, file type, image height and weight in pixels. The following figure illustrates the settings JSON file of the converter.

```
"settings_file": "../experiments/settings/",
"start_date": "2018-10-01",
"end_date": "2018-12-11",
"plot_image_data": "no",
"file_type": "csv",
"pm_counter": "CPU_load",
"number_of_days": 120,
"check_fetched_cells": false,
"converte_cells_to_excels": false,
"converte_cells_to_images": true,
"image_height": 64,
"image_width": 64,
"interpolation_method": "none",
"dpi": 100,
"plot_point_size": 1
```


FIGURE 5.2: Converter - JSON Setting file

5.2.5 Converter - Check the Fetched Data

Working with real-world data can be challenging. In this project, our input data contains more than one thousand cells. As a result, fetching data and generating the inputs images is time-consuming. The intuition behind the first steps is to eliminate the time for fetching all the cells again. The first step returns an error message when something is missing and saves this information in a file.

Additionally, the first step of the algorithm 2, returns a file with all the cells passes the testing. Then this file is used to continue with the creation of the aggregation of those cells. This functionality is essential; otherwise, we could end with missing cells for some days. Besides, it returns two excel files, one contains the missing cell data, and the second one contains the correct cell data. Moreover, the converter later will be based on the right data to create input images. Someone can think about "why we need then the missing excel file". The answer is that we can fetch again only the missing data instead of fetching all the cells again.

A separated class created, which checks all cell names folders, for three cases. Before start checking each folder for each case individually, it creates two empty pandas data frames [42] for the missing cell name. Next, it starts by checking if the folder is empty, and if it is empty, it sets the information into the pandas data-frame. Second, it checks if the folder is not empty, and the length of the files contains are less than the number of days that are fetched. This number can be set in the setting files and applied to the whole program. If this is true, it sets the pandas data-frame with the specific information. In the case, that everything is correct inside the cell folder, it sets to the data-frame which contains the right cells. Algorithm 2 presents the

functionality of the first step.

Algorithm 2: Check Cell Folders for missing and correct data

Data: Excel files for each Cell Names in a period, directory is a global variable
Result: **This algorithm checks if missing data exist in a period. Then it generates one excel file for missing cell data and one for correct.**

```
1  Initialisation;
2  count_empty, count_missing, count_correct, loop_count = 0;
3  missingCell_DF = create_empty_cell_df()
4  correctCell_DF = create_empty_cell_df()
5  for each cell name do
6      loop_correct += 1 list = [cell_name,length_of_Files latitude,longitude]
7      length_of_Files← length_of_Files_in_a_folder()
8      if file path does not exist then
9          data_frame.saveDF(single_time_iteration)
10         count_empty += 1
11         missingCell_DF.set_data_frame(list)
12     else if the folder exists and number of days less than the expected then
13         data_frame.saveDF(single_time_iteration)
14         count_missing += 1
15         missingCell_DF.set_data_frame(list)
16     else if the folder exists and the number is above the expected then
17         data_frame.saveDF(single_time_iteration)
18         count_correct += 1
19         correctCell_DF_DF.set_data_frame(list)
20     Print Information about missing and correct cell data
21     missingCell_DF.saveMissingCellNamesDF(directory)
22     correctCell_DF.saveCorrectCellNamesDF(directory)
23 end
```

Overall, in the case that we have missed data, the generated excel file is used by another *R* file, which fetching again the absent cells. After fetching the absent cells, we can use the excel file, which contains the correct cell names to process to the *second step* of the converter, as is shown in figure 5.1.

5.2.6 Converter - Convert Cell Folders to Individuals Days

After a successful checking phase, we can aggregate all cells together. More specific, each of the folders corresponds to a particular day contains 96 excel files. Each of these files contains for a particular timestamp aggregated all the cells. Each day folder includes 96 excel files as an interval, also known as **Result Output Period** (ROP), in the network is every 15 minutes. Therefore, 96 ROP exist for each day. Algorithm 3 starts with a loop for each day in the defined period. Then, for that particular day, it loops for 96 intervals. Next, it initialises a class to keep all the data. Moreover, the algorithm calls a function that reads the corrected cell names for that specific day and timestamp.

In further detail, inside that function, a try-catch starts by reading the cells data and then checking if there are any missing data. After finishing the loop, it returns the data frame with all the cell names and two boolean variables, missing data and empty folder. If both are equal to false, it means that there are not any missing data and not any empty folders for all cells. In the case of non-missing data and non-empty folders, the algorithm saves the data frame as an

excel file. Else, it returns an empty data frame and both missing and folder empty as true.

Algorithm 3: Convert Cell Folders to Individuals Days

Data: The cell names for the loop are coming from the generated correct cell excel file.
Result: **This algorithm generates for each cell folder and a period, a folder for a particular day which contains 96 Excel Files for a 15-minute interval in a day.**

```
initialisation;
for a single day in a period do
    for a single time iteration in a time range of 15 min(ROP) do
        data_frame_class ← initial_data_frame_class()
        for each cell name do
            try:
                data_frame_class, empty, missing_data ←
                  for_a_cell_name_and_a_specific_rop_read_data()
            catch ValueError e:
                Print e
            end
            if not empty and not missing_data then
                data_frame_class.saveDFsingle_time_iteration
        end
    end
end
```

5.2.7 Converter - Image Generation

The converter in the third step generates the input images, as is presented in figure 5.1. The primary purpose of the converter is to generate an image for every 15-minutes in a day for a period.

Algorithm 4: Convert Cell Folders to Images

Data: path example: ../data/2018_10_29/2018_10_29_00_00_00.csv
Result: **This algorithm generates the for each day 96 images for a 15-minute interval. The result contains folders for a period, where each contains 96 images. Furthermore, it is possible to set different**

```
initialisation;
for a single day in a date range do
    sum_of_files = sum_of_files(file_path)
    if sum_of_files == 96 then
        for each timestamp in a single day (96 intervals(ROPs) do
            path_of_a_timestamp_for_a_specific_day
              ← set_specific_path_of_a_single_day_and_timestamp.
            if file path exist then
                data_frame
                  ← read_data_frame_data(path_of_a_timestamp_for_a_specific_day)
                folder_path = sets the folder for save the images
                image_file_path = sets the path of the images
                save_as_image(data_frame,folder_path,image_path)
            else if the folder exist and number of days less than the expected then
        end
end
```

Algorithm 4, starts with a loop over all the data range. Then, algorithm 3 starts reading each file for a specific day, and it checks if the number of files inside the folder is equal to 96 intervals.

Then, it makes a loop for all the 96 files. For each of these files, it makes use of **sklearn**[1] minimax scaler preprocessing to normalise the data. Then it takes the mean of all the cells with the same latitude and longitude and groups them together by using Pandas data frame, and it saves the figure.

5.2.8 Generated Input Dataset

The generated input contains images which represent a specific timestamp of the network. Furthermore, points represent the latitude and the longitude of multiple cells and the density of the colour that each each point represents the PM counters, where for the training process, grey-scale images used. Figure 1.1 shows an example of a generated image.

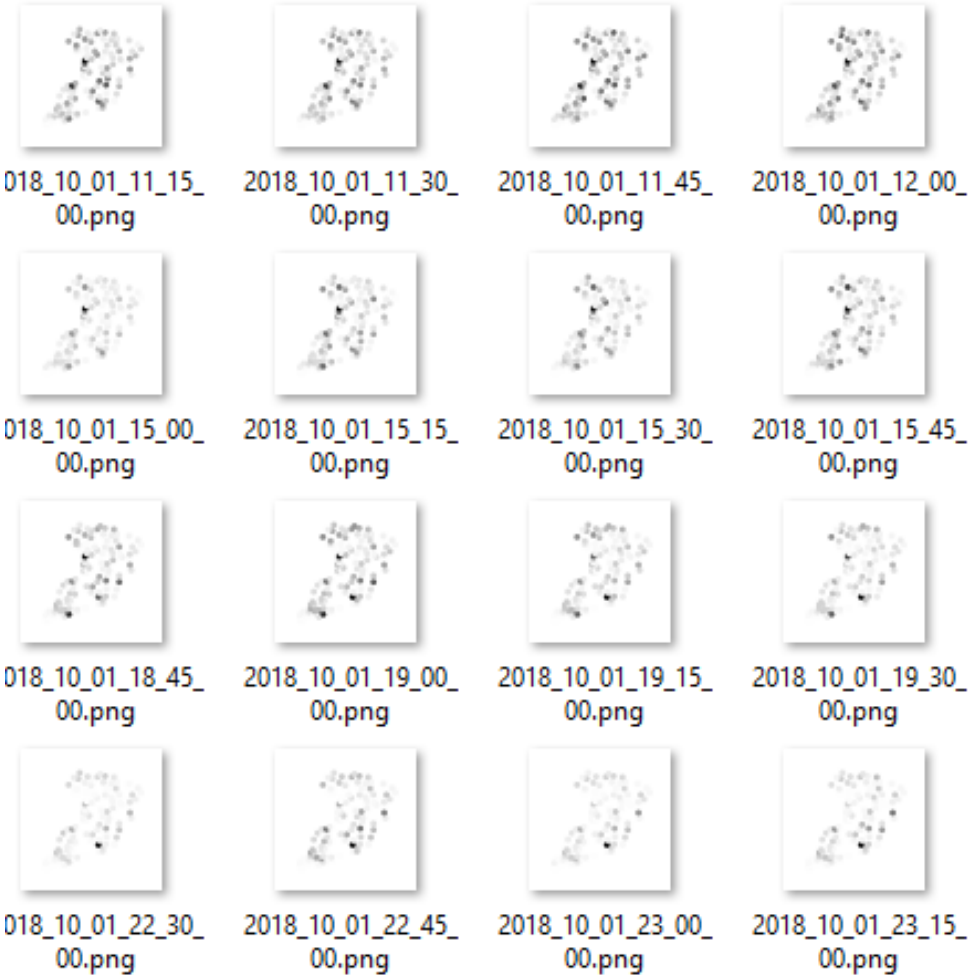


FIGURE 5.3: Converter Framework - Generated grey-scale (64x64) Images

The image represents generated grayscale images. As we can see the overall value of PM counters follows some pattern. As a result, during the working hours, the demand was higher than the hours around eleven o'clock. Therefore, it is expected that the density of the points is changing accordingly during the day.

Furthermore, the generated images were *TIFF* imaged with size pixel size 32x32 and 64x64. Moreover, *TIFF* images do not lose any information compared with *JPEG* files and *PNG*. Thus, *PNG* are good for areas with similar colours density. One reason for using *TIFF* in our implementation, was because points are representing different aggregated cells, and the density of each of these grayscale points must be high resolution. Moreover, we trained the model with

[1] **sklearn** is a python library for machine learning. (Wikipedia)

PNG images and *TIFF*, where better results observed by using *TIFF* images. In this way, *TIFF* images were used for the training process in this project.

Last but not least, as the converter has full control of the locations and the values assigned in each location. In this way, it is possible to generate anomalous images for testing purposes, as the input data that used in this project represent a normal network without abnormalities. Therefore, a functionality that creates abnormal images by assigning random values to each location has been created. Figure 5.4 illustrates an example of a generated anomalous image.

FIGURE 5.4: Converter Framework - Anomalous Generated grey-scale (64x64) Images

Over, the converter can generate images with flexible size, and different representation can be implemented. For instance, interpolation methods can be used like kriging (Gaussian process regression). In this way, the converter allows creating different representations for flexible experiments.

Chapter 6

Anomaly Detection Approach

This section discusses the anomaly detection approaches that are based on GANs. It starts by introducing an overview of anomaly detection methods, where 6.1 and 6.2 are based on [43]. Next Section 6.3 is based on [43, 44]. Furthermore, the section discussing two approaches, AnoGAN 6.5 and f-AnoGAN 6.6, which are anomaly detection methods based on GANs 4.6.

6.1 Anomaly Detection

Nowadays, there is a high demand in the concern of security in networks. Because, any attack or anomaly in the system can affect various domains such as economic issues, privacy and security of data to national security, and so on. As real-world networks becoming more complex is even harder to maintain the stability of systems. The increasing number of interconnected devices, services, users, and so on; have been increasing the complexity of the network rapidly the recent years.

Accordingly, the domain of anomaly detection is an active and broad research area, where an important task is to define anomalies which have not seen before. Thus, next-generation networks require even more complex integration with emergent domains like the Internet of things (IoT) becoming a need for future societies. Therefore, anomaly detection is necessary as abnormalities might occur more frequently, whereas challenges such as maintaining the stability of such a network can be a very challenging task.

6.2 Network Anomaly Detection

Network anomaly detection refers to the problem of identifying patterns in data that do not conform to the expected behaviour occurring on a network, where a network anomaly refers to a faulty behaviour in the software system such as unusual peaks and drops of traffics, where software issues may cause that. For instance, anomalies in a network system might arise after a software upgrading simultaneously in numerous devices.

Therefore, it is essential to maintain the stability of the network at a higher level. In other words, detect abnormalities in a whole area, where thousands of device exists, before starting to identify individual cell anomalies. In this way, this project aims to detect anomalies at a higher level in the network, where multiple definitions exist for anomaly detection[45, 43], where the solution to identify the abnormality is coming from the concept of the normality.

As is mentioned in [3], **anomaly detection** aims to identify data that does not "follow" the normal distribution and **novelty detection**, aims to identify new unobserved data which are not included in the training input. Because both detection methods share the same foundation is it crucial to use an approach which can be used to identify both cases. As we will see later in sections 6.5 and 6.6, Generative Models can be used for both **anomaly detection** and **novelty detection**. On the other hand, one crucial fact that contributes most in choosing a Deep Learning architecture is the nature of the data. We can classify input data into sequential and non-sequential. For instance, time-series, voice, and text can be considered as sequential whether images are non-sequential data.

Moreover, time series can be classified into **univariate** and **multivariate** time series. Univariate works only with a single variable. Whereas, multivariate time series concentrate on

multiple features. There are three groups to separate the above two categories, point anomalies, contextual anomalies and collective anomalies, as is described in [43].

6.3 Anomaly Detection with Deep Learning

Deep anomaly detection models can be divided into three categories. The first category is **Supervised Deep Learning anomaly detection** methods, which are not popular due to the fact of lack of availability of training samples, which are labeled. Furthermore, abnormal behaviours are often a rare event, and supervised methods lack in performance. Because there is an imbalance between the classes, where the total number for normal data is far more than the abnormal, the second category is **Semi-Supervised Deep Learning anomaly detection** approaches and is widely used as this category needs only a small sample of anomalies to work. The last category is **Unsupervised Deep Learning anomaly detection** approaches use an automatic way of labelling data. Overall, unsupervised techniques outperform existing methods like principal component analysis (PCA), Support vector machines used for health and cyber-security.

As is described in 6.2, anomalies can occur in the data, but sometimes new undiscovered abnormalities might appear. Deep Learning is a subgroup of machine learning, which can identify unknown patterns; and has been successfully used the recent years in various application as is mentioned in 2.3. The recent success of Deep Learning due to that can handle big data which traditional Machine Learning algorithms cannot have increased the popularity in various domains such as Medical, IoT, cybersecurity, and so on.

When we are performing anomaly detection is vital that the normal behaviour of the network or the expected behaviour is known. The Deep Learning model has to learn how normal behaviour looks like before start performing anomaly detection. This ability of identification of new unobserved patterns called **Novelty Detection**. When new unseen data occur a **novelty score** can be used. Using an agreement threshold score for the decision, when the score departs from the decision score can be considered as abnormal.

Supervised Deep Learning anomaly detection can be more accurate compared with and **Semi-Supervised Deep Learning anomaly detection**, as is mentioned in [43]. However, the need for labelling data for both normal and abnormal; and the failure to separate abnormalities when the feature space consists of non-linearity and complexity makes it less attractable for anomaly detection. **Semi-supervised deep learning anomaly** assumes that one label class exist. Furthermore, it makes assumptions to score an instance as abnormal based on points that are close both in the input and learned feature space. Additionally, the **deep hidden layers** allow a robust discriminative attribute for separating normal from abnormal data. Generative Adversarial Networks (GANs), as is described in 4.6, trained in both Semi-Supervised and Unsupervised Learning have shown promising results in the area of anomaly detection. A potential disadvantage is that fewer anomalous behaviours might not possible to identify.

6.4 Deep Hybrid Models (DHMs)

The main propose of **Deep Hybrid Models** [43] (DHMs) is to be used as a feature extractors. Autoencoders are primarily used as feature extractors as they can learn the hidden representation. As a result, these features are used as an input for anomaly detection algorithms. In general, Deep Learning approaches are widely used as feature extractors. DHMs are consists of two steps the robust feature extractor and the anomaly detection model, where the first aims to separate irrelevant feature which can hide a potential anomaly. The second aims to model on high dimensional space an anomaly detector. This hybrid models have higher computational complexity as are consists of multiple models but are suitable for reducing the dimensionality curse as they make a use of feature extractors. Therefore, DHMs are scalable and can be faster as these models can reduce the input space significantly.

6.4.1 Results of Anomaly Detection

The output of anomaly detection algorithms is either an **anomaly score** or a **label** which can represent anomaly, normal or system alarms. The anomaly score is based on the outliers. For example, the anomaly score can be computed on an image pixel-wise. Next, a threshold (decision score) is defined for decision deciding on some expert. In general, the decision score is more valuable than a binary classification. In the Unsupervised Leanring anomaly detection, the residual vector (reconstruction error) is measured and then is used from domain experts to label abnormalities.

6.4.2 Unsupervised Deep Anomaly Detection

Unsupervised Deep anomaly detection has shown a state-of-the-art performance, among models such as AE, VAE, GANs, and so on. Autoencoders have been used widely as cardinal Unsupervised architecture in anomaly detection. The intuition behind unsupervised methods for anomaly detection is that normal data can be separated from abnormal data in the input space. Lastly, unsupervised Deep Learning algorithms can provide an anomaly score based on densities or distances of the inner properties which deep learning approaches aims to learn.

Unsupervised Deep anomaly detection challenges consist of the complexity of learning higher-dimensional data. Another common issue is that compressing the data by dimensional reduction requires to adjust a hyper-parameter which influence the optimal result. Last, Unsupervised Deep Learning techniques are delicate to noise input, where that noise can reduce the accuracy of the model.

6.5 AnoGAN

Supervised Learning for detecting abnormal imaging markers related to a disease is challenging due to the difficulty of annotating. Additionally, Supervised Learning is challenging due to the need for a wide variate of labeled data which makes it an inapplicable method. In this way, Unsupervised anomaly detection with GANs to Guide Marker Discovery or **AnoGAN** uses Unsupervised Learning for anomaly detection on imaging data. The model consists of a Deep Convolutional GAN network, which is based on DCGAN (see 4.7).

Moreover, it is crucial to identify as to quantify these abnormal markers (disease or in our case failures of the network), as it is equally important for observing the diseases progress and dealing with those diseases. The main idea of AnoGAN is to learn a **manifold** $\mathcal{X}$ of healthy data images and then identify pixel-wise the anomaly by combining an anomaly score. The way that learning the manifold is by mapping from the imaging data to the latent space.

On the other hand, AnoGAN can be used in a relevant way for network anomaly detection. For instance, as medical imaging data allows the detection and the treatment of diseases, it is feasible to convert network time-series data to images, as it was discussed in 5.2, where having these network imaging data can activate the same procedure, as detecting anomalies on medical data, and treat the abnormalities of the network as "*diseases*". For instance, by identifying the abnormal area pixel-wise, we can root case where abnormalities occur. Furthermore, novelty detection can also be applied, as it has the same methodological foundation compared with anomaly detection, as is mentioned in the improved version of AnoGAN, called f-AnoGAN [3].

AnoGAN implementation uses the original loss of the initial GAN implementation [5], as described in 4.6, to learn the normal anatomical variance, which refers to the healthy input data $\langle I_m \rangle$. More precisely, the input $I_m \in \mathbb{R}^{axb}$ denotes the intensity images [1]. On the other hand, during testing, $\langle \boldsymbol{y_n}, l_n \rangle$ is given, where y_n denotes images which are not used during the training and l_n denotes the labeled image by a single binary value, which defines an image as normal or abnormal.

[1] An **intensity image** indicates a data matrix, where each value represents a density value. For instance, for black and white images, we can have densities scaled from 0 (black) to 255 (white).

6.5.1 Mapping Images to the Latent Space

After training a GAN model, the Generator can generate images close to the real distribution, as learns $G(z) \rightarrow x$. On the other hand, the **inverse mapping** from $G(x) \rightarrow z$ is not given initially by the trained GAN model. This is important as the inverse mapping allows, given an unseen image, to identify a point in the latent space z which match most with that image and lies on the learned manifold $\mathcal{X}$.

For instance, as is mention in DCGAN [24] sampling between two close points in the latent space generates similar images. In other words, sampling can be though, as an example of having images which represent numbers such as $[0, 1, \ldots, 9]$, similar to the MNIST dataset. In this case, sampling in the z space can yield different variations of images representing numbers such as number one or any another number. If we continue sampling in a different direction in the z space, the generated images will begin having a smooth change of representing numbers (different shapes). In the case of being precisely at number one in the z space, going farther, the generated images will start representing different numbers. In the assumption that we are sampling towards number seven, we will start observing images that look similar to both one and seven until finally reach the point where number seven is located.

Similarly, like the example described above, AnoGAN approach aims to find the optimal z^* that recovers the best-learned representation based on objective images. The technique is to find the optimal z^* is by sampling z_i randomly from the latent distribution and generates samples through the generator $G(z_i)$, referred as mapping to the latent space. Moreover, through back-propagation and after n steps, the position of z^* is optimised by defining a loss function based on the generated image, where the optimal $G(z^*)$ can be compared with an objective image x.

6.5.2 Combined Loss Function to Mapping Unseen Images to the Latent Space

The mapping consists of a combination of two parts which contribute during the training for mapping new images to the latent space. The first component called **residual loss**, which aims to impose the resemblance between z^* and objective image x. The residual loss is calculated **pixel-wise** to recognise anomalous regions of an image. The residual loss is given by

$$\mathcal{L}_R(z^*) = \sum |x - G(z^*)| \tag{6.1}$$

The second part consists of the Discriminator loss, which aims to impose the generated sample to be located on the learned manifold $\mathcal{X}$. The Discriminator loss is based on **feature matching** [30], as was suggested for improving the stability issues of original GANs. Therefore, rather than optimising generator via equation 4.18, which aims to optimise generator via the Discriminator's output, is imposed to generate samples with similar statistics as the input data, used during training.

$$\mathcal{L}_D(z^*) = \sum |f(x) - f(G(z^*))|, \tag{6.2}$$

where a layer of the Discriminator gives the statistics over input images. In this way, the intuition is the adjustment of latent point does not depend only on discriminator's decision, but instead, it uses the feature representation. Furthermore, as is mentioned in [2], the discriminator acts more likely as a **feature extractor** than a binary classifier. Thereby, the mapping to the latent space by combining both residual loss and discriminator loss is given by

$$\mathcal{L}(z^*) = \underbrace{(1-\lambda)\mathcal{L}_R(z^*)}_{\substack{\text{Residual} \\ \text{loss}}} + \underbrace{\lambda \mathcal{L}_D(z^*)}_{\substack{\text{Discriminator} \\ \text{loss}}}, \tag{6.3}$$

where λ is a factor that controls the influence of each of the two losses, for instance, when λ equals to zero, only the generator is used as the discriminator loss is equal to zero. On the

other hand, when λ is equalled to one only the discriminator loss is used. Overall, during the mapping, the weights for both Generator and Discriminator retained locked, whereas only the z factors are modified.

6.5.3 AnoGAN Detection of Anomalies

The aim of a constant anomaly detection is to input new images and identify if that image is normal or abnormal. As it is described in 6.4.1, having a decision score can be more valuable than just a classify an input as normal or abnormal. Furthermore, by having an anomaly score, a threshold (decision score) can be defined in order to classify normal from abnormal. AnoGAN calculates an anomaly score, which is similarly defined as equation 6.3 is given by

$$\mathcal{A}(x) = \underbrace{(1-\lambda)R(x)}_{\substack{\text{Residual} \\ \text{loss}}} + \underbrace{\lambda D(x)}_{\substack{\text{Discriminator} \\ \text{loss}}} , \tag{6.4}$$

where a higher $A(x)$ anomaly score denotes an abnormal image and a smaller anomaly score implies a normal image which follows the normal learned distribution.

6.6 Fast AnoGAN (f-AnoGAN)

Fast Unsupervised anomaly detection with GANs (f-AnoGAN) [3] is proposed to tackle the difficulty of annotating medical imaging and detection abnormalities on images. Furthermore, handling unknown annotations which might not exist or are not described well is essential as unknown abnormalities might exist. Similar to network anomaly detection where unknown anomalies might occur, the network might become unstable. Fast-AnoGAN (f-AnoGAN) aims to identify abnormal images, where normal behaviour can be used as scanning **biomarkers**[2]. F-AnoGAN consists of three parts:

- The first part consists of a Generative Model for training the normal images based on WGAN with Gradient Penalty 4.9. The approach allows the model to capture the variability inherent of normal data during training. On other words, the GAN model captures the underlying distribution of the "healthy" data.
- The second part consists of an Encoder model which enables a fast mapping to GAN's latent space and subsequently a faster anomaly detection approach.
- The last part consists of an anomaly detection of that trained model on both image-level and image segmentation, which aims to identify if the input image fits on the learned manifold. The anomaly detector makes a use of a combination of the **residual feature error** based on the discriminator and the **image reconstruction error**.

Fast AnoGAN (f-AnoGAN) allows both **anomaly detection** and **novel anomaly detection**. Furthermore, f-AnoGAN permit the anomaly detection on an abnormal image on pixel localisation level.

6.6.1 Training f-AnoGAN for Anomaly Detection

The first training step consists of the GAN model, which make a usage of WGAN with Gradient Penalty as described in 4.9. The input for both WGAN-GP and encoder consist of unlabeled data used similarly for AnoGAN. In this way, WGAN-GP learns a mapping function (non-linear) from the z-space (latent/hidden space) to the manifold **X** trained on the non-labelled images. Fast AnoGAN is training both generator and critic (discriminator) synchronously, which based on [26], as was discussed in 4.9. In this way, the trained model can generate images close to the real distribution $\mathbb{R}_r$ given the input noise z.

[2] **biomarker** refers as a biological state indicator. (source: Wikipedia)

Similarly, with AnoGAN (in 6.5.1) the mapping from $x \rightarrow z$ is required for the anomaly detection. The second training step requires a trained GAN model to train the encoder. The trained encoder allows **mapping** images to the latent space in a single step throughout inference compared with the AnoGAN, which need multiple steps to map a new image to the latent space. The proposed method compares three different approaches for training an encoder:

- A **ziz encoder** training, which is a z-to-z mapping analogous to [46], where this approach aims to minimise the $\mathcal{L}_{ziz}$ loss based on a random selection residual error and reconstructed position in the hidden-latent space.
- A **izi encoder training** is the new proposed approach for an image to image mapping, where the goal of this approach is to minimise the $\mathcal{L}_{izi}$ loss based on the residual error of the true input images and reconstructed images.
- The third one is a combination of Discriminator's features residual loss of $\mathcal{L}_{\mathcal{D}}$ and the $\mathcal{L}_{izi}$ loss, referred to as $\mathcal{L}_{izi_f}$.

Both $\mathcal{L}_{ziz}$ and $\mathcal{L}_{izi}$ use a Convolution Autoencoder 4.3, which $E(x) = x \rightarrow z$. Moreover, the decoder role is assigned to the trained WGAN-GP, which maps from $z \rightarrow x$. The mapping via the encoder is accomplished by using the fixed Generator parameters and only adjust the parameters of the encoder.

6.6.2 Encoder Architecture - ziz

The **ziz encoder architecture** uses the opposite structure of Autoencoders. In this way, the trained generator is used first to generate an image input by randomly sampling z from the latent space Z. Then, the encoder aims to map the generated image back to the latent space. Throughout the training process, the goal is the minimisation of **MSE** between the randomly selected z and the reconstructed $\hat{z}$:

$$\mathcal{L}_{ziz} = \frac{1}{d}||z - E(G(z))||^2, \tag{6.5}$$

where **d** refers to the dimensionality of the latent space. Additionally, as is stated in [3], one limitation of this approach is that only generated images go through the network. Therefore, the encoder is limited in that way, without being able to receive real input images.

6.6.3 Image to Image Encoder Architecture - izi

On the other hand, **izi encoder architecture** uses normal Autoencoders architecture. Throughout the training process, the encoder maps the real image input to z, whereas the generator maps z backwards to the image space. The goal is to minimise the residual loss using MSE between the normal and reconstructed images.

$$\mathcal{L}_{izi} = \frac{1}{n}||x - G(E(x))||^2, \tag{6.6}$$

where **n** refers to the number of the images and $||\cdot||^2$ denotes the residual over greyscale images, squared pixel-wise. Furthermore, izi approach has the limitation that can only use indirect measurement from the accuracy of the model, as the correct location on the latent space is unknown, which means that the Generator will yield images which sporadically was sampled over the latent space. Moreover, these generated images might have a small residual, but can potentially not represent the trained input. Therefore, izi approach is not stable by its own to identify abnormalities.

6.6.4 Image to Image Encoder Architecture with D residual in the feature space - izi$_f$

In this direction, Schlegl et al.[3], implemented the residual over the critic (discriminator) to the feature space to improve the encoder. In this way, the objective of $\mathcal{L}_{izi_f}$ is given by

$$\mathcal{L}_{izi_f} = \frac{1}{n}||\boldsymbol{x} - G(E(\boldsymbol{x}))||^2 + \frac{\kappa}{n_d} \cdot ||f(\boldsymbol{x}) - f(G(E(\boldsymbol{x})||^2, \tag{6.7}$$

where $f(\cdot)$ denotes features from a Critic (Discriminator) layer. Moreover, k refers to a weighting coefficient, and n_d denotes features dimensionality, which is based on feature matching technique [30], as it was discussed in 6.5.1.

6.6.5 Fast AnoGAN Detection of Anomalies

During anomaly detection among the query and generated image, the $\mathcal{L}_{izi_f}$ loss is used, as is formulated in equation 6.7. Therefore, the anomaly score $\mathcal{A}(\boldsymbol{x})$ is defined by

$$\mathcal{A}(\boldsymbol{x}) = \mathcal{A}_R(\boldsymbol{x}) + \kappa \cdot \mathcal{A}_D(\boldsymbol{x}) \tag{6.8}$$

$$\mathcal{A}(\boldsymbol{x}) = \underbrace{\frac{1}{n}||\boldsymbol{x} - G(E(\boldsymbol{x}))||^2}_{\mathrm{A}_R(\boldsymbol{x})} + \kappa \cdot \underbrace{\frac{1}{n_d} \cdot ||f(\boldsymbol{x}) - f(G(E(\boldsymbol{x})||^2}_{\mathrm{A}_D(\boldsymbol{x})} \tag{6.9}$$

Furthermore, both equation 6.5 and equation 6.6 can be used for anomaly score. In this way, for normal images which follows the normal distribution, the anomaly score will yield a low value compared with anomalous images which will yield a higher score. Importantly, localisation of the abnormalities can be found by using the residual for the localisation of the abnormalities pixel-wise given by

$$\mathcal{A}_R^{\cdot}(\boldsymbol{x}) = |\boldsymbol{x} - G(E(\boldsymbol{x}))| \tag{6.10}$$

Chapter 7

Experiment Setup and Results

For the training purposes, Ubuntu 18.04.3 operation system with graphics card GeForce RTX 2080 8GB and python version was 2.7.15+ for both training part and the input generation via the converter 5.2. Keras over Tensorflow [47] has been used for the experiment, enabling fast training and excitement. Tables 7.1 refers to some of the most important packages used in this project. Note that ST-ResNet is ruining with Keras version equal to 2.1.6, whereas all the anomaly detection models running with version equals to 2.2.4.

TABLE 7.1: Experiment - Python Packages

Package	Version
Tensorflow	1.14
Tensorflow-gpu	1.13.1
Scikit-learn	0.20.4
Keras	2.1.6 - 2.2.4
Numpy	1.16.5
Matplotlib	2.2.4
Pandas	0.24.2
Scipy	1.2.2
Sklearn	Initial Version

The chapter presents the experiments for both prediction and anomaly detection model, with mainly focusing in the anomaly detection and root cause analysis of the abnormality based on GANs 4.6. The prediction model which aims to predict the next timestamp of the network is based on ST-ResNet, as is described in 3.2.

Additionally, even if DMVST-Net and STDN outperforms ST-ResNet, replicating both models were not feasible due to the time limitation of this project, where DMVST-Net source code [1] provides only the model part and not the whole implementation[2], whereas STDN source code [3] implementation loads the whole dataset in once, as 128GB of memory was available, which was not in our case. Therefore, we let the implementation for both DMVST-Net and STDN for future work.

TABLE 7.2: Spatial-Temporal Prediction Models

	Year	LSTM	Outperform	GitHub
DeepST [17]	2016	No	ARIMA - SARIMA - CNN	Yes
ST-ResNet [18]	2017	No	DeepST	Yes
DMVST-Net [19]	2018	Yes	ST-ResNet - DeepST	Limited
STDN [20]	2018	Yes	DMVST - ST-ResNet - DeepST	Limited

This table contains recent Deep Learning prediction models for Spatial-Temporal data.

[1] DMVST-Net source code: https://github.com/huaxiuyao/DMVST-Net
[2] The implementation does not have the necessary stuff for running.
[3] STDN source code: https://github.com/tangxianfeng/STDN

However, the implementation code for the ST-ResNet was given, and the results of ST-ResNet was relatively satisfied compared with DMVST-Net and STDN [20], table 7.2 contains an overview of the Spatial-Temporal prediction models. On the other hand, the Anomaly detection model is based on Generative Adversarial Networks (GANs), as was described in section 4.6, and more specific our implementation consist of both modified AnoGAN and f-AnoGAN by using as input the generated images based on the converter framework (5.2).

Additionally, the prediction model makes use of the **first channel**, where the colours channel represented as the first dimension of a 3D array, whereas the anomaly model is running with **channel last**, where the colour channel represented as the last dimension of a 3D array. Therefore, the channel is defined at the beginning of the training, and this is important when we want to run the prediction model and the anomaly model in real-time.

7.1 Original Prediction Model Implementation

For ST-ResNet prediction approach, the original model was first trained before modified for our purposes. In this way, the original ST-ResNet model was retrained by having some modification in order to be able to compare with our converted dataset, as in our implementation flows number is equal to one compared with the initial implementation which has flows equal to two. We evaluate the model by both comparing the result of root mean square error and plotting the predicted value for different settings.

The original implementation of ST-ResNet works with two datasets, TaxiBJ and BikeNYC[4], which include both trajectories and weather conditions. In our experiments, TaxiBJ, which provides GPS trajectories referred to the taxicab data, was replicated with the ST-ResNet model. Furthermore, ST-ResNet was trained with three different approaches for closeness, period and trend. Three different training settings was replicated using different parameters for closeness, period and trend (table 3.1). Moreover, the original code uses $\mathbf{l}_c = 3$, $\mathbf{l}_p = 1$ and $\mathbf{l}_q = 1$. Overall, figure 7.1 shows the results of RMSE in equation 3.5, for both training and validation.

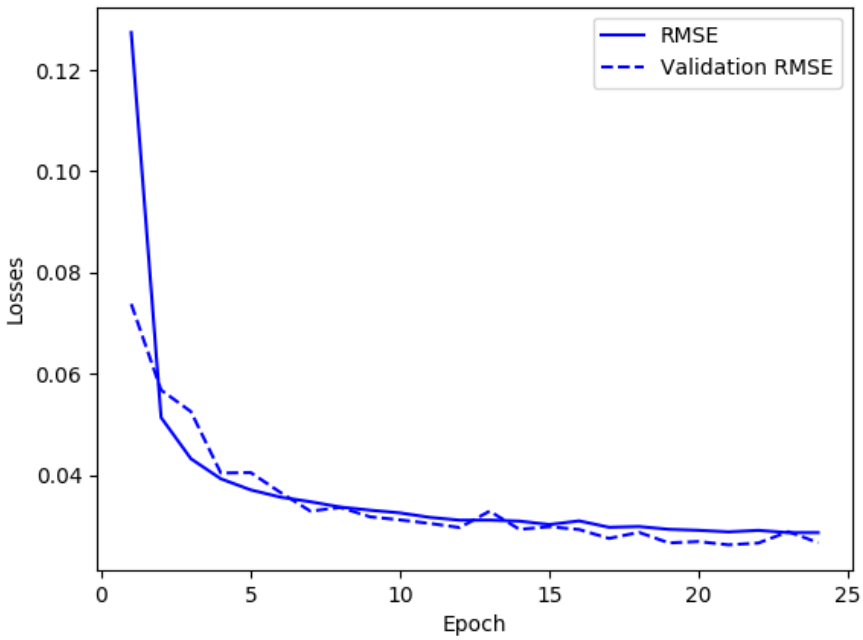


FIGURE 7.1: Original ST-ResNet RMSE - Closeness: 3 Period: 1 Trend: 1

[4]NYC Bike system dataset, which contains trajectories for bike rents in New York

Furthermore, increasing the number of $\mathbf{l}_p = 3$ and $\mathbf{l}_q = 3$, as is illustrated in figure 7.2, returned slightly different results, but without any significant difference.

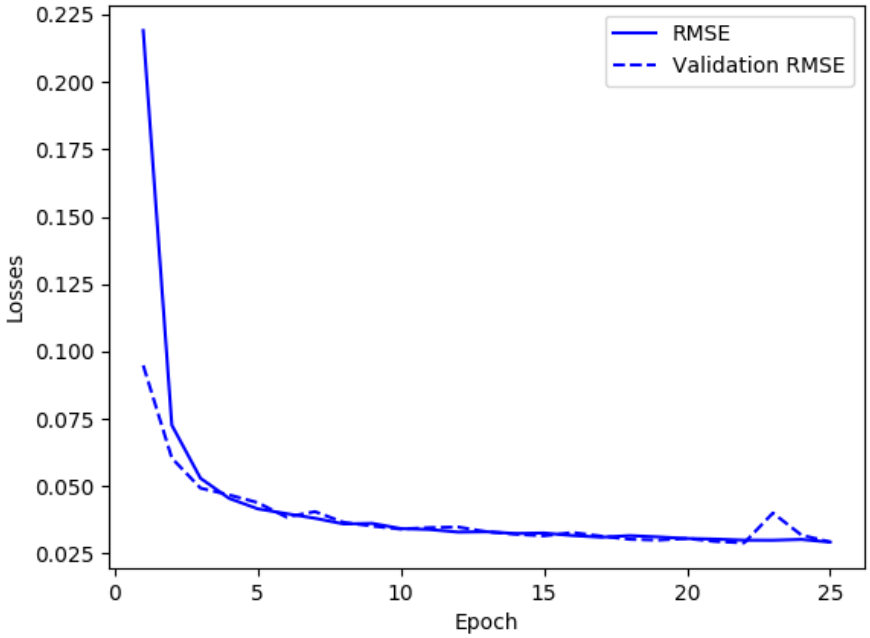


FIGURE 7.2: Original ST-ResNet RMSE - Closeness: 3 Period: 3 Trend: 3

Moreover, figure 7.3 shows the training results where coefficients $\mathbf{l}_c$,$\mathbf{l}_p$ and $\mathbf{l}_q$ are equal to one, where similar RMSE results was observed.

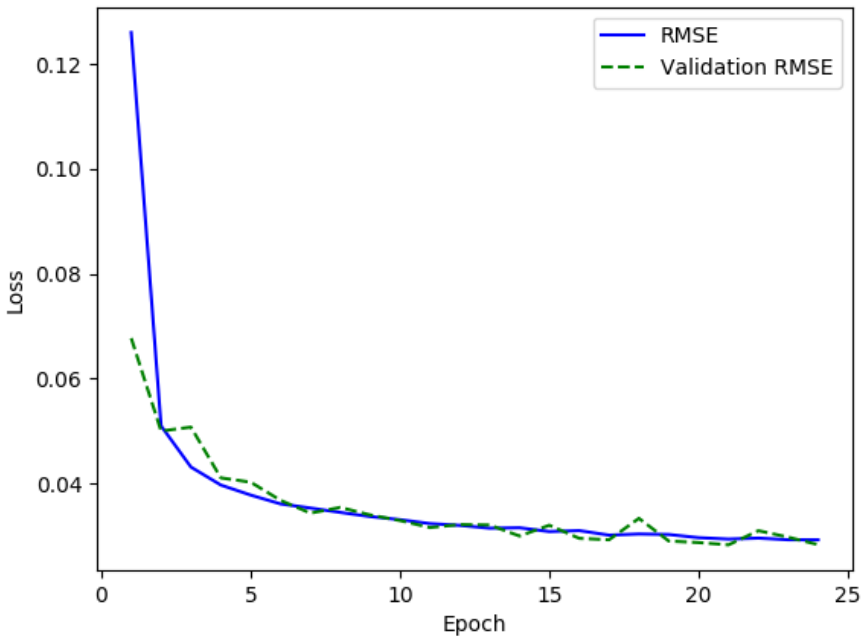


FIGURE 7.3: Original ST-ResNet RMSE - Closeness: 1 Period: 1 Trend: 1

7.2 ST-ResNet - Spatial-Temporal Prediction Implementation

The original ST-ResNet model was modified to process the experiments with the network input dataset. The main changes include modification of the parameters presented in 3.1, added functions for importing the dataset, adding functions for visual representation by creating an additional python file for exclusive prediction and evaluation. The original ST-ResNet model was trained to ensure that the additional function for importing our dataset follows the same principals as the original.

TABLE 7.3: Experiment - Modified ST-ResNet - Training Parameters

ST-ResNet	
Parameters	**Values**
Batch Size	32
Intervals	96
Learning Rate	0.00001
$\mathbf{l}_c$: *closeness*	{1,3}
$\mathbf{l}_p$: *period*	{1,1}
$\mathbf{l}_q$: *trend*	{1,1}
Map height	32, 64
Map width	32, 64
flows	1
L: Residual Units	12

Furthermore, the original ST-ResNet has intervals equals to 48 per day. However, in our case, the number of intervals changed from 48 to 96. Moreover, the flows are equals to one now, as the original ST-ResNet model had two parts of a city in two separated images. In our case, we have one image representing the area of interest.

Moreover, the size of the image has been increased from 32x32 to 64x64, the intuition behind the size is that both anomaly detection models work with images sized 64x64 and higher resolution images can potentially increase the accuracy of identifying the root cause anomaly. Therefore, images size for both 32x32 and 64x64 pixel were generated, and both were trained with the modified ST-ResNet model. Last, the learning rate was reduced as it was reduced from 0.0002 to 0.00001. The next section presents the results for $\mathbf{l}_c, \mathbf{l}_p, \mathbf{l}_q = [1, 1, 1]$ and $\mathbf{l}_c, \mathbf{l}_p, \mathbf{l}_q = [3, 1, 1]$.

The intuition about the chosen parameter for closeness, period and trend is based on the original implementation and the reproducing results of the original implementation, as was described in 7.1. Last but not least, for our residual units remained equals to 12, as this approach was the best of the ST-ResNet implementation. Final, the validation test consists of 10% of the training test for all of our experiments, where approximately 20000 images used for training both prediction and anomaly detection model.

7.2.1 Results of ST-ResNet

We first trained the 32x32 generated input with closeness, period and trends equal to one. The learning rate was set to 0.0007, and the RMSE of the training is illustrated in figure 7.4. Furthermore, meteorology and holiday data were not used in this project. Figure 7.5, shows predicted timestamps on the top of the figure and at the bottom, the tested timestamps. Overall the trained model can predict timestamps with pixel size 32x32.

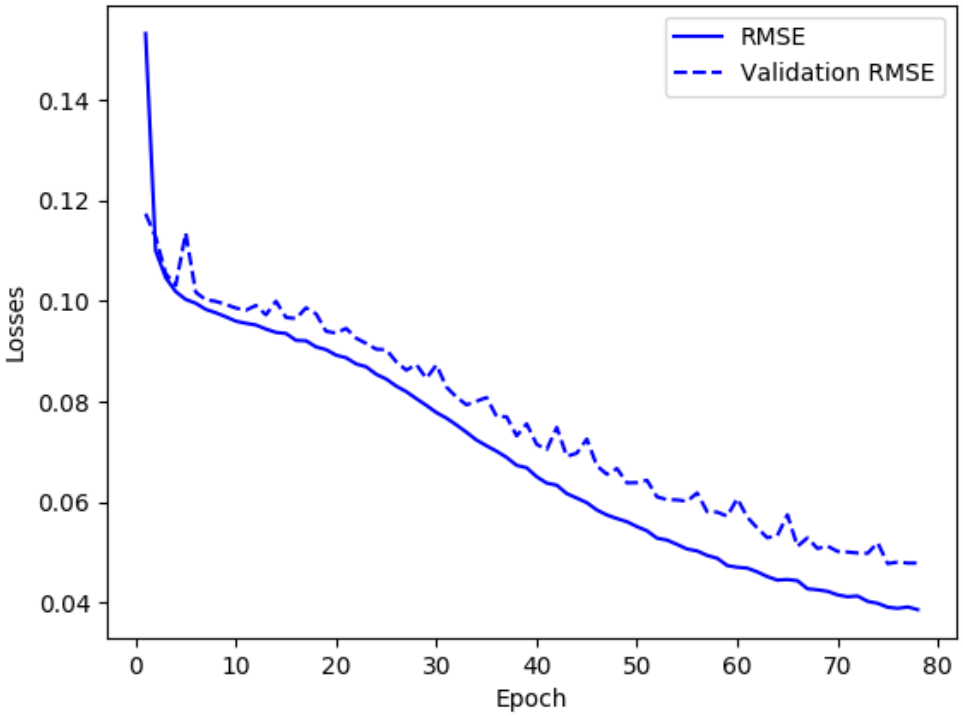


FIGURE 7.4: ST-ResNet RMSE - Predicted 32x32 - Closeness: 1 Period: 1 Trend: 1

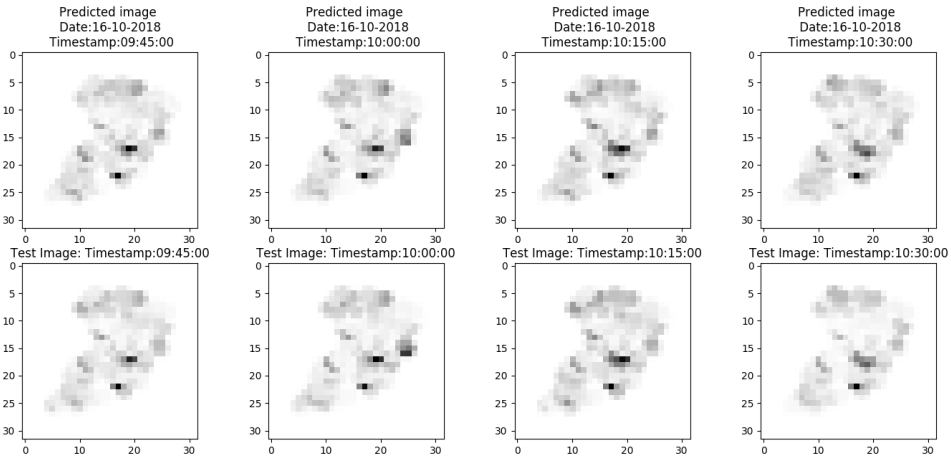


FIGURE 7.5: Prediction Results 32x32 - Closeness: 1 Period: 1 Trend: 1

Next, the model was trained for $\mathbf{l}_c, \mathbf{l}_p$ and $\mathbf{l}_q$ with 64x64 input size images. The learning rate was set to 0.00001, and the early stopping[5] of the original model removed. The training data consists of consist of 205 days. Figure 7.7 present as the predicted timestamps for an image size 64x64. Thus, figure 7.6 illustrates the overall training RMSE over the validation RMSE.

[5]**Early stopping** is a function of Keras python package, which stops the training when a quantity of measurement does not anymore improve.

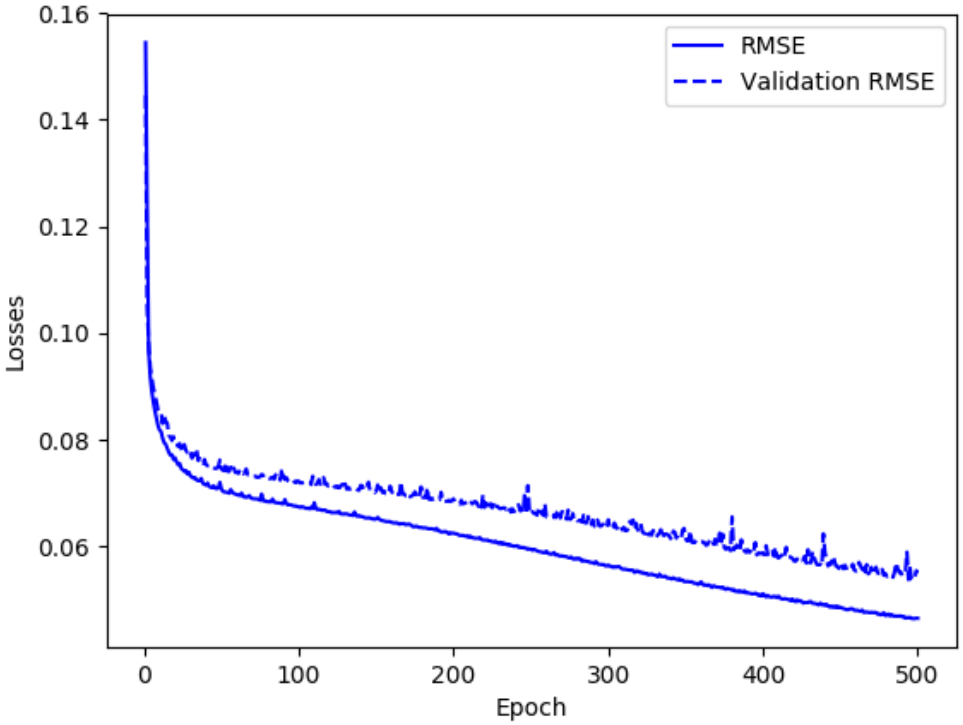


FIGURE 7.6: ST-ResNet RMSE - Predicted 64x64 - Closeness: 1 Period: 1 Trend: 1

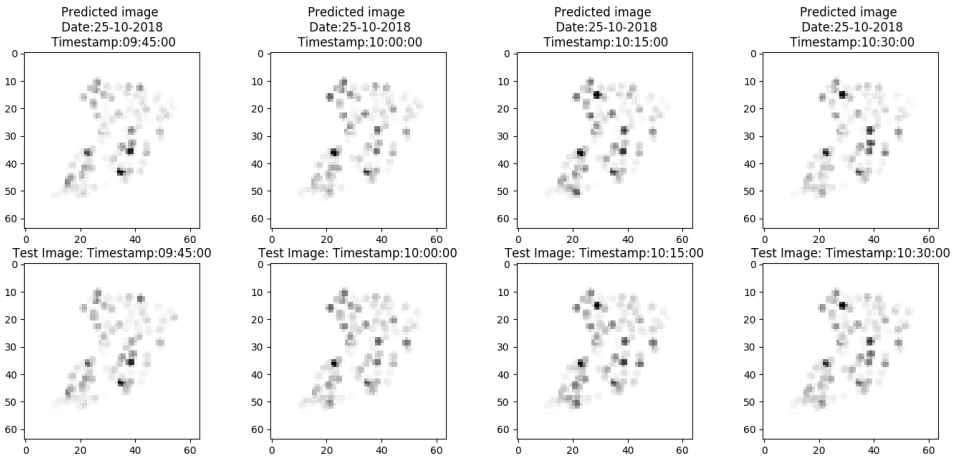


FIGURE 7.7: Prediction Results 64x64 - Closeness: 1 Period: 1 Trend: 1

Furthermore, the model was trained for $\mathbf{l}_c = 3$, $\mathbf{l}_p = 1$ and $\mathbf{l}_q = 1$. The same settings, as described prior were applied in this implementation except for the closeness, which was $\mathbf{l}_c = 3$. Same, as was described in the two previews implementations for 32x32 and 64x64, the RMSE is given in figure 7.8 and the predicted timestamps in figure 7.9.

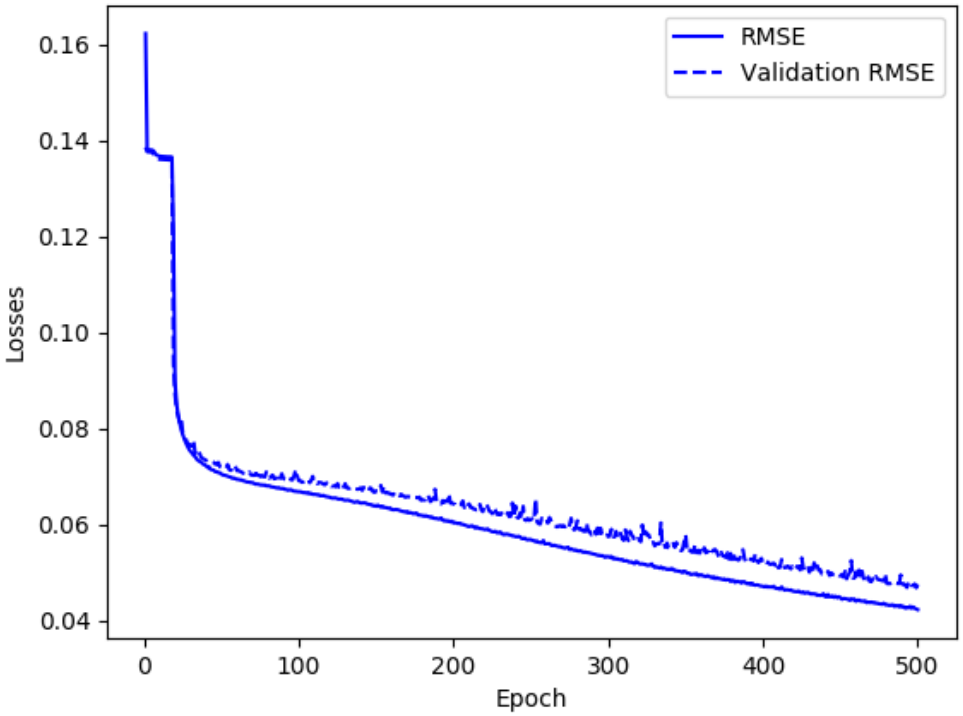


FIGURE 7.8: ST-ResNet RMSE - Predicted 64x64 - Closeness: 3 Period: 1 Trend: 1

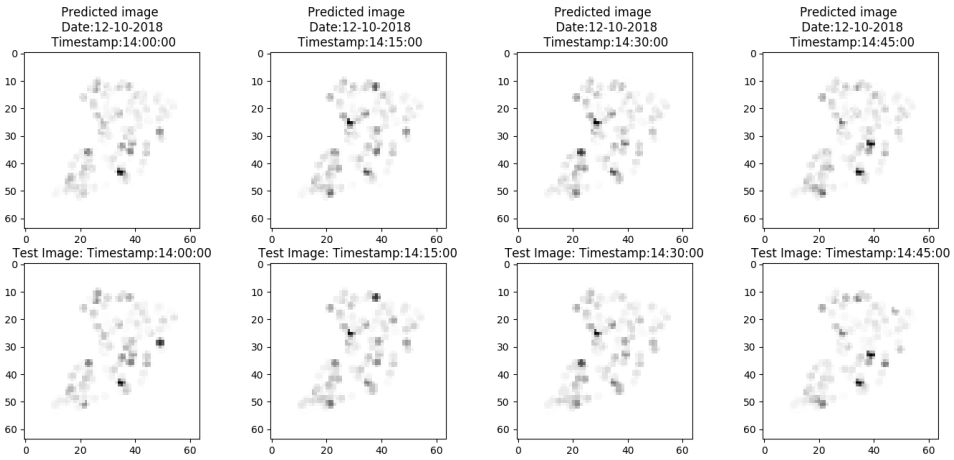


FIGURE 7.9: Prediction Results 64x64 - Closeness: 3 Period: 1 Trend: 1

Metadata applied to the training input. For each day an array of length equals to 8 is created. The first seven values represent week and weekend days, respectively, where the true day is denoted as one and all the other days as zero. Furthermore, the last value of the array is set to zero when it is a weekday and one when it is a weekend. Moreover, table 7.4 indicates the results of the experiment, where the best results observed with 500 epochs, closeness equals to 3, period and trend equal to one. Note, that higher numbers for $\mathbf{l}_p = 3$ and $\mathbf{l}_q = 3$ trained without observing any significant improvements.

TABLE 7.4: Experiment - Final Results - Spatial-Temporal Prediction odel

Epochs	l_c, l_p, l_q	Train Score	Train RMSE	Test Score	Test RMSE	metadata
25	[1,1,1]	8.362002	0.065584	8.535706	0.066947	Yes
500	[1,1,1]	6.077814	0.047669	6.817189	0.053468	Yes
500	[3,1,1]	**4.774285**	**0.037445**	**5.133429**	**0.040262**	Yes

7.3 AnoGan - Network Anomaly Detection Implementation

The implementation of AnoGAN in this work is based on [48] implementation. First, the model was trained with MNIST dataset for testing; figure 7.10 represents the reproduced results. Then AnoGAn was trained for both label and unlabelled data with our dataset, which was generated from the converter framework, as it was discussed in 5.2. Furthermore, a class (Appendix B) was created to import the input and prepared for the training, similarly used for the ST-ResNet implementation. More, our dataset consists of images with 96 intervals per day, were each interval introduced as one hot by Keras package to label the images with the corresponding timestamp.

FIGURE 7.10: MNIST - Labeled Training with AnoGAN

The training is based on AnoGAN implementation 4.7, where Discriminator consists of four Convolution layers and Generator of four transposed Convolution layers. More specifically, table 7.5 indicates some of the Generators' and Discriminators' settings. For more information about the implementation referred to [48].

TABLE 7.5: Generator - AnoGAN Implementation

	Tensorflow (tf): Package	Channels	Filter	Stride
Generator	**tf.nn.conv2d_transposes**	512-256-128-64	5x5	NHWC=[1,2,2,1]
Discriminator	**tf.nn.conv2d**	64-128-256-512	5x5	NHWC=[1,2,2,1]

The data format is NHWC, which refers to the batch size, height, weight and channel accordingly, wherein our implementation, the stride was set to 2x2. The transpose Convolution operation is often called "deconvolution", but this is not a correct name, as is mentioned in the TensorFlow documentation [6]. Table 7.6 contains the training parameter for GAN model; this

[6]More information about the specific function can be found in https://www.tensorflow.org/

approach is closely related to the DCGAN implementation [7], as is mentioned in AnoGAN [2].

TABLE 7.6: Experiment - AnoGAN - Training Parameters

Parameters	Values
Epochs	20
Batch Size	64
Learning Rate	0.0002
Input size	64x64
Optimiser	Adam
beta1	0.5
z dimension	100

We first train AnoGAN with labelled data; figure 7.11 shows an example of the generated data. The trained model captures the distribution of the normal data, but there are some unrelated points which the model has generated. Different hyperparameters were used, such as learning rate from 0.0002 to 0.00001 and beta1 for the optimiser equal to 0.9, but no semantic improvement was observed for training with labeled dataset. Additionally, by increasing the epoch, overfitting was observed, in this way we set epoch size equals to 20 similar to the AnoGAN implementation.

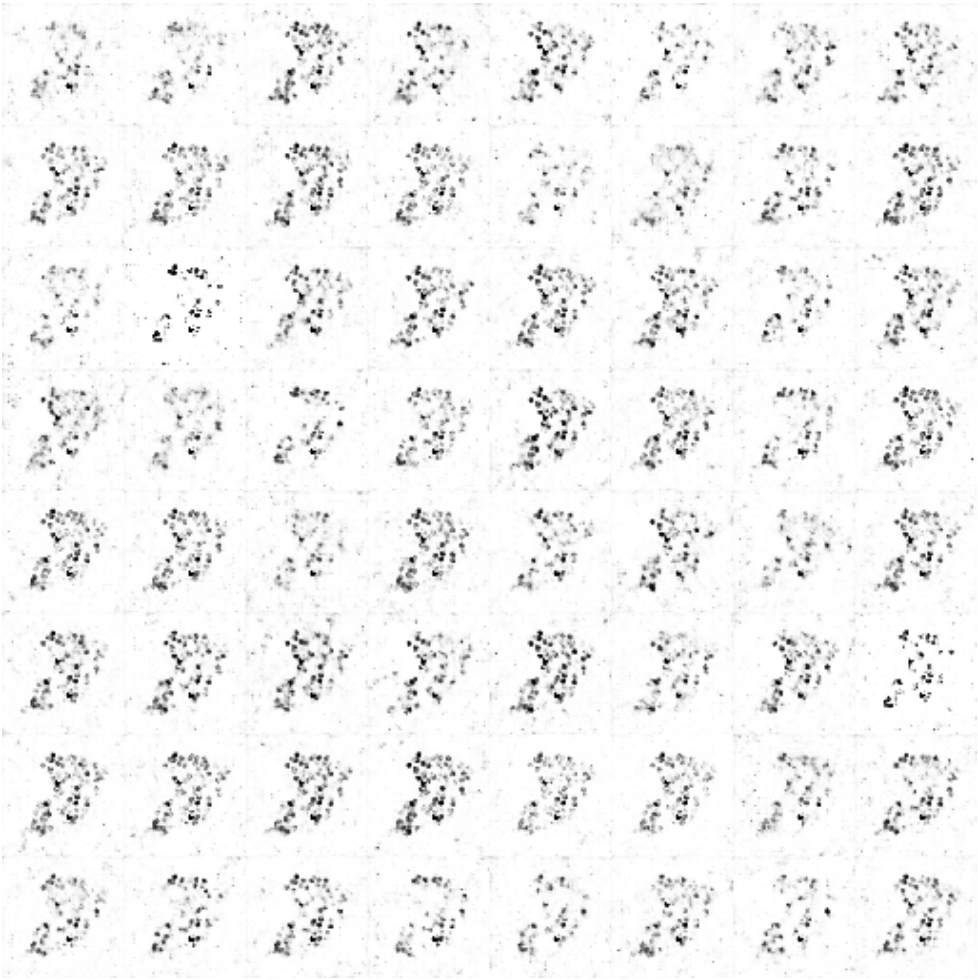


FIGURE 7.11: Experiment - Labeled Training with AnoGAN

[7] AnoGAN is based on the DCGAN GitHub implementation: https://github.com/bamos/dcgan-completion.tensorflow/blob/master/train-dcgan.py

Next, we trained the model with the images only, where figure 7.12 indicates generated images. The results seem good, but this might be due to the intuition of the model to reproduce images that are comfortable with, this is potentially related to the limitations of the original GANs, as it was discussed in Deep Generative Models.

FIGURE 7.12: Experiment - Unlabeled Training with AnoGAN

7.3.1 Results of Mapping New Images to the Latent Space & Anomaly Score

The mapping to the latent space is based on the equation 6.3, where λ is equal to *0.1*. The proposed backpropagation steps for mapping in AnoGAN is 500, but in our implementation, that number was increased to 4000. Next, by increasing the step size, the mapping time is increased, but as is illustrated in figure 7.13, the anomaly score drops. Moreover, the exact learning rate for mapping is not referred by AnoGAN; in this project, the learning rate for the mapping was equal to 0.001. Figure 7.13 indicates three snapshots; the first represents the initial step; the second image represents step 125, the third step 2000 and the final 4000 steps. On the whole, by increasing mapping steps results in closer images to the latent space, as the anomaly score drops from approximately 80 to 57. In order to have a meaningful score, we calculate the mean, the minimum and the maximum score for an image, then we scale it with the minmaxscaler of sklearn preprocessing package (see Appendix B).

$$X = \begin{cases} anomaly, & \text{if } \mathcal{A}(\boldsymbol{x}) \geq threshold \\ normal, & \text{otherwise} \end{cases} \tag{7.1}$$

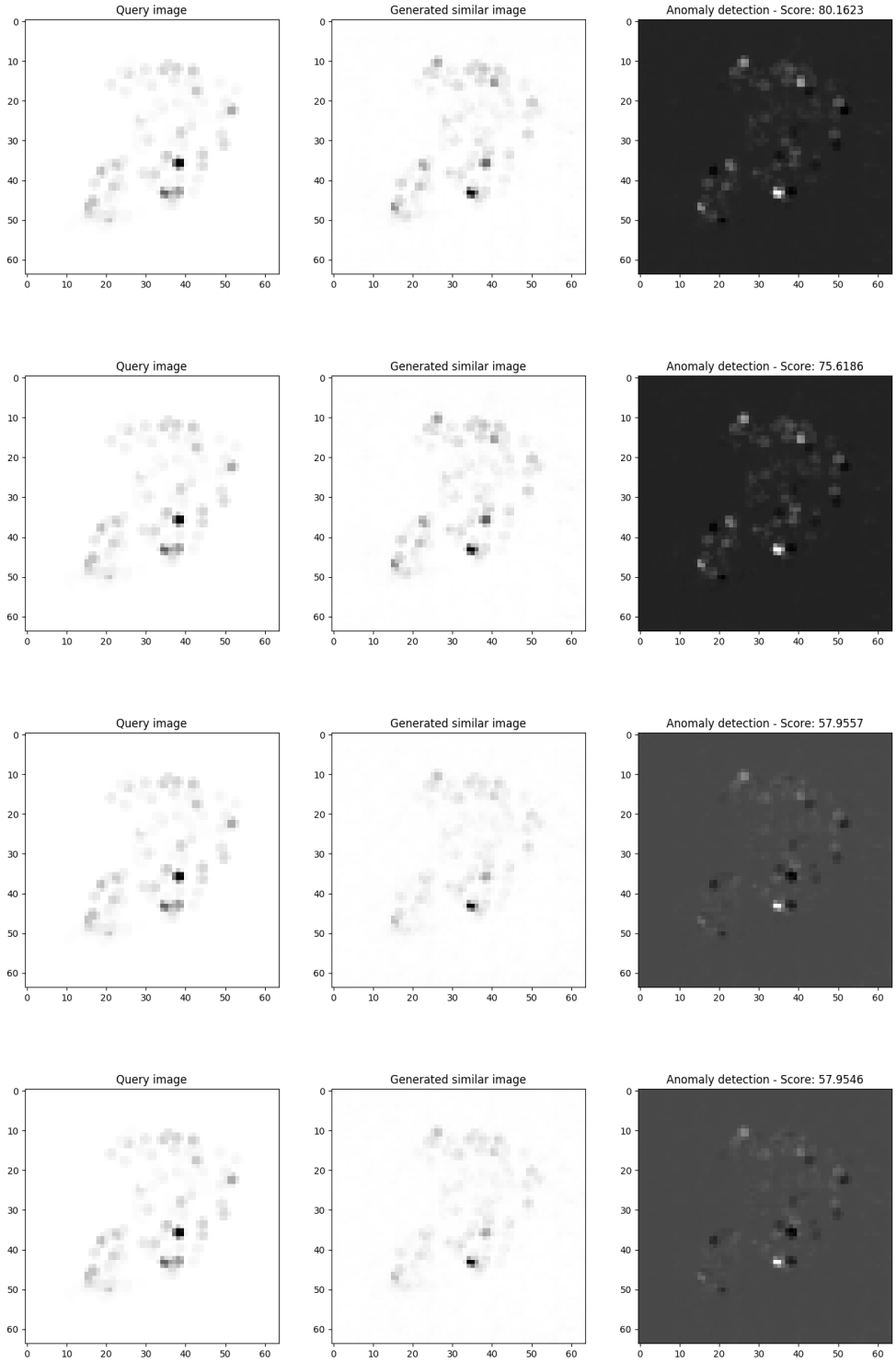


FIGURE 7.13: Experiment - Mapping to the latent space - AnoGAN

Steps accordingly from top to bottom:
Step 0 (AS: 75.6186) - Step 125 (AS: 80.1623) - Step 2000 (AS: 57.9557) - Step 4000 (AS: 57.9546)

Moreover, to obtain a threshold, the mapping was trained on normal images, this results in a value in the range [0,1] which refers to the mean of all mapped images. In this way, a threshold combined with experts can be used for assigning a different status to the anomaly detector, such as normal, alert, anomaly, high anomaly, and so on. The above example represented in equation

equation 7.2.

$$X = \begin{cases} high\ anomaly, & \text{if } \mathcal{A}(\boldsymbol{x}) \geq threshold_{high\_anomaly} \\ anomaly, & \text{else if } \mathcal{A}(\boldsymbol{x}) \geq threshold_{anomaly} \\ alert, & \text{else if } \mathcal{A}(\boldsymbol{x}) \geq threshold_{alert} \\ normal, & \text{otherwise} \end{cases} \tag{7.2}$$

An experiment for normal predicted and anomalous images based on the $\mathcal{A}(x)$ of equation 6.3 illustrated in table 7.7, where the actual $\mathcal{A}(x)$ for each of the experiments are represented. For the predicted and the anomalous images, 15 predicted and 15 abnormal images were used, whereas for the test images, two days of images where used. Note that these are preliminary testing results for this particular example, the mapping will score in slightly different results for each new mapping iteration.

TABLE 7.7: Experiment - Mapping to New Images and Predicted & Anomaly Score Based on $\mathcal{A}(x)$

Images	Min	Mean	Max	Normalised Mean
Test images	*49.656*	*87.833*	*150.492*	**0.378**
Predicted ST-ResNet images	*131.366*	*149.856*	*163.724*	***0.571***
Mean	*90.511*	*118.8445*	*157.108*	***0.425***
Anomaly Images	*347.137*	*470.152*	*537.466*	***0.646***

Overall, AnoGAN enables anomaly detection and the expansion of the anomaly detection vocabulary. However, defining a threshold is challenging, and experts should further assess it.

7.4 f-AnoGAN - Implementation & Results

As is mentioned in 6.6, f-AnoGAN training consists of two steps. The first step consists of the WGAN-GP training following by the mapping training based on the trained WGAN-GP. Final, the trained Autoencoder is used for anomaly detection.

Furthermore, consider the evaluation of GAN, which was discussed in 4.10, original GAN loss, which means that AnoGAN is not referring directly to the quality of the generated image, but instead, it provides information about how well the generator fools the Discriminator. Moreover, the original GANs loss does not provide any information about the quantitative trained model. However, Wasserstein loss directly refers to the quality of the image, as was discussed in 4.8. Thus, by combining multiple evaluation measurements, the effectiveness of the quantitatively GAN evaluation measurement can be increased by satisfying the following desiderata, as defined in [6]:

- **Discriminability**: The ability of the model to preference models with high discriminability, which refers to the strength of the model to distinguish true from fake samples.
- **Overfitting sensitivity**: The ability of the model to learn the whole distribution and not just memorise part of it. In that way, model becomes sensitive to overfitting, mode collapse, and so on.
- **Disentanglement in the latent space**: The ability of the evaluation to the preference model with disentanglement in the latent space, which can denote the semantics of the generated images along the axis in the latent space.
- **Perceptual Judgement**: Human perceptual judgement and ranking of the model

- **Small degree of sample and computational complexity**, where samples refer to the number of samples that are needed to discriminate between real and fake samples.
- **Distortion Sensitivity**, where a small shift of the pixel or rotation does not affect the measurement.
- **Defined boundaries**

In this work, **Wasserstein loss** combined with **Precision, Recall and F1 Score** [8] used for quantitative evaluate the model. Furthermore, the model was assessed qualitatively by investigating and visualising the internals of networks by interpolating in the z-space, as was discussed in 4.10.

7.4.1 Results of WGAN-GP

In the original WGAN-GP implementation, both Generator and Critic (Discriminator) is a **residual network**, where Generator consists of four residual networks with the up-sampling following by **Batch Normalisation**. Next, the residual is followed by a Relu activation, a Convolution layer and final a tanh activation. On the other hand, the critic consists of four residual networks with down-sampling, followed by a Linear Normalisation instead of Batch Normalisation, as was proposed in 4.9. Moreover, table 7.8 shows the training parameters, where λ refers to the equation 4.25 and critic iterations is equals to five, as was discussed in 4.9. For more details about the implementation refer to 5.

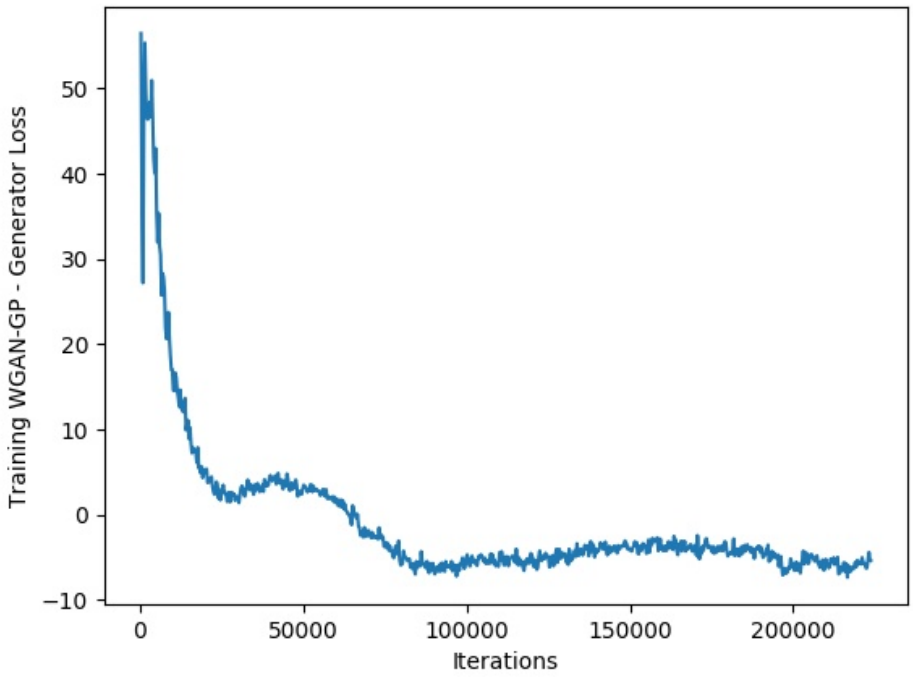


FIGURE 7.14: f-AnoGAN: WGAN-GP - Generator Loss

Additionally, the model was trained for 224000 iterations, where figure 7.14 represents the generator loss, whereas figure 7.14 the critic (discriminator) loss.

[8] Precision, Recall and F1 Score, implementation was inspired from [49] measurement implementation. The GitHub source code: https://github.com/awweide/pub-ffi-gan

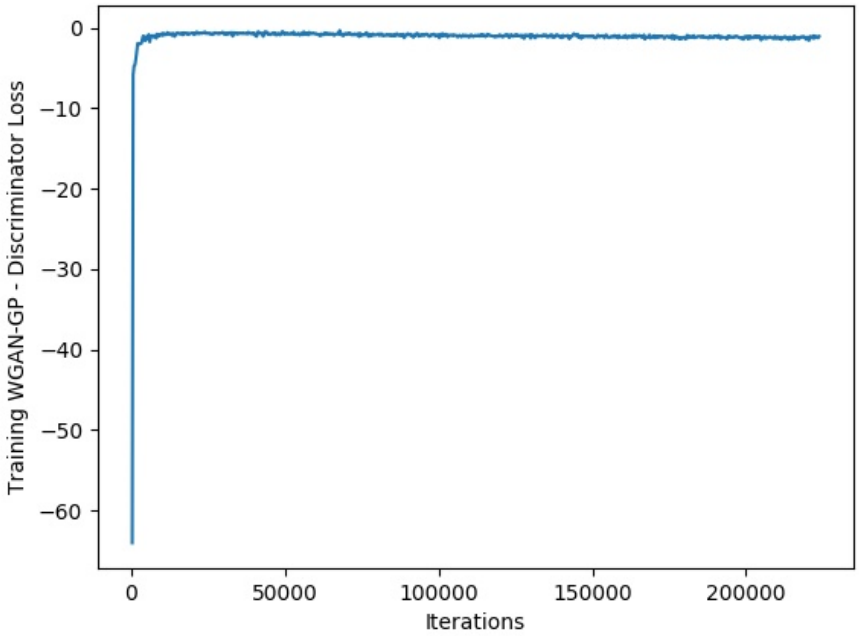
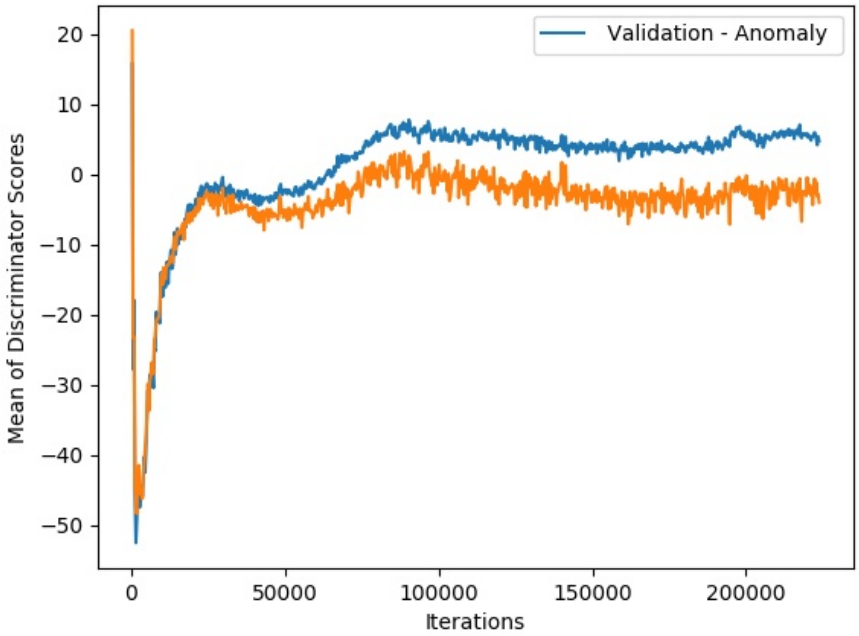


FIGURE 7.15: f-AnoGAN: WGAN-GP - Critic (Discriminator) Loss

TABLE 7.8: Experiment - Training f-AnoGAN - Parameters for WGAN with Gradient Penalty

Parameters	**Values**
Epochs	*7*
Iteration per Epoch	*32000*
Batch Size	*64*
λ	*10*
Critic Iterations	*5*
Input	*64x64*
Optimiser	*Adam*
Learning Rate	*0.0001*
Adam beta1	*0*
Adam beta2	*0.9*
z dimension	*128*

Furthermore, during training, both validation and anomalous data passed to the critic, where the mean of critic cost for both validation and anomalous images were computed for each iteration. In this way, figure 7.16 presents both mean costs for validation and abnormal images, where blue colour represents the validation and the orange shows the anomalous costs. Thus, the difference between abnormal and validation costs starts separating approximately after 40000 iterations, where the model has started learning the true data distribution.



spi

FIGURE 7.16: f-AnoGAN: WGAN-GP - Critic (Discriminator) - Mean Costs between Validation and Anomaly

Additionally, figures 7.17 illustrates the histogram between scores of validation and anomalous data. More specifically, the figure illustrates the Critic (Discriminator) scores for two batches of 64 images, where one batch represents normal images, which have not been used for during training, and 64 images for representing an abnormality.

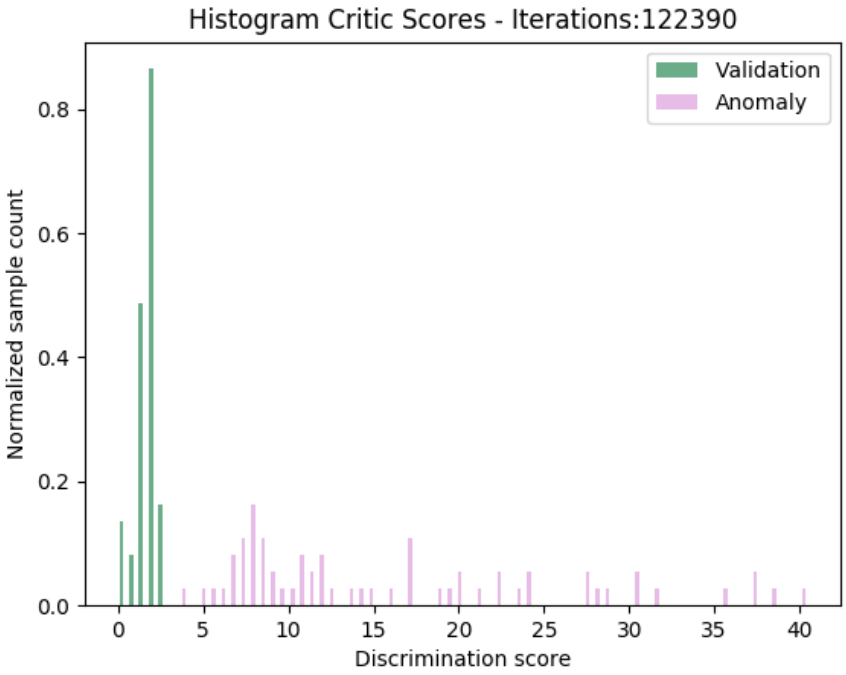


FIGURE 7.17: f-AnoGAN: WGAN-GP: Histogram of Critic Scores for Validation and Anomalous Images

As was mentioned above, during the training, Recall, Precision and F1 score were calculated. The threshold for identifying the confusion matrix was the maximum cost value of the validation images, which was different for each batch and iteration, then true-positive (TP) was equal to the sum of the abnormal costs which were less than the threshold, and false-positives (FP) the rest. On the other hand, the true-negative (TN) was equalled to the sum of validation costs which were greater or equal to the threshold, and false-negative (FN) the rest. Moreover, the F1 score is defined as:

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{7.3}$$

where,

$$Recall = \frac{TP}{TP + FN} \tag{7.4}$$

$$Precision = \frac{TP}{TP + FP} \tag{7.5}$$

For this particular experiment, the F1 score is equal to one approximately after 110000 iterations. The algorithm was trained several times to justify that these results remained the same for each training process approximately. Due to the lack of true anomalous data, in this experiment, the abnormal images were generated. Therefore, it is crucial to test the model with true abnormal images in the future and determine the behaviour of the model by passing authentic anomalous images.

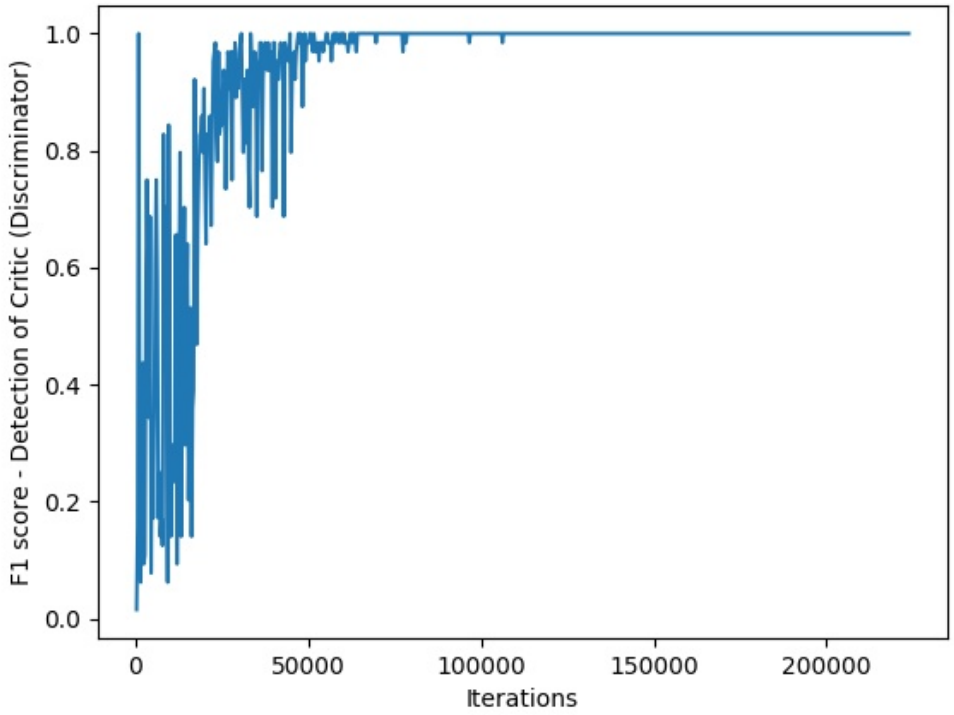


FIGURE 7.18: f-AnoGAN: WGAN-GP - Critic (Discriminator) - F1 score

Furthermore, figure 7.19 shows the ground truth of 64 grey-scale images with 64x64 pixel size. Next, figure 7.20 illustrated generated samples of the trained model. Overall, it seems that the model has learned to generate images to satisfy visually human perception.

FIGURE 7.19: f-AnoGAN: WGAN-GP - Ground Truth - Training Input

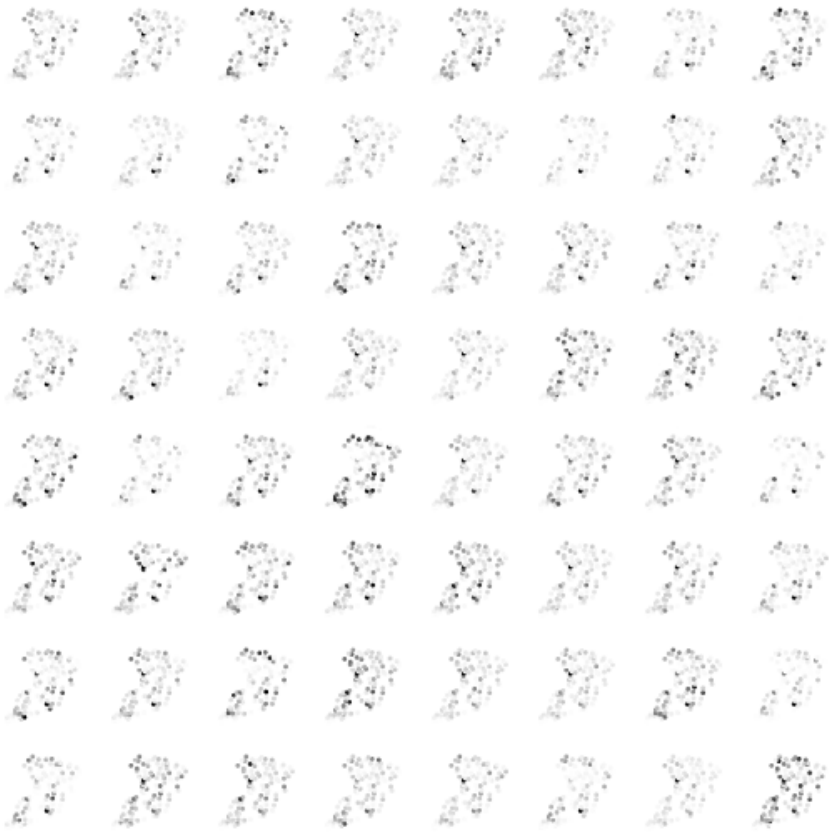


FIGURE 7.20: f-AnoGAN: WGAN-GP - Generated Samples - 64x64 grey-scale - Iteration 224000

Moreover, figure C.1 in Appendix C is an investigation to the space continuity over different vectors seeds by visualising the intervals of the network, as was described in 4.10, for assessing qualitative GAN model.

7.4.2 Results of Implementation Training Mapping - izi_f Architecture

By having the trained WGAN-GP model, the next step is to train the mapping based on the izi_f architecture, as was described in 6.6.4. The encoder consist is a residual network, which consists of a Convolutions layer followed by four residual blocks, a linear operation and final tanh activation. For more information about the implementation refers to the implementation code [9]

TABLE 7.9: Experiment - Training f-AnoGAN - Parameters for izi_f Architecture

Parameters	Values
Iteration per Epoch	*50000*
Batch Size	*64*
Optimiser	*RMSProp*
Learning Rate	*0.00005*
Loss	*MSE*
κ	*1.0*

The izi_f architecture trained for 50000 iterations in approximately four hours, table 7.9 presents the model parameters. The loss is based on equation 6.7, where the $\mathcal{A}_R(\boldsymbol{x})$ is the MSE between real and reconstructed image; and $\mathcal{A}_D(\boldsymbol{x})$ is the MSE between real and reconstructed image features. Additionally, figure 7.21 illustrates the loss over the training input and figure 7.22 over the validation, where the loss converges approximately over 0.01 for the training input and over 0.03 for the validation.

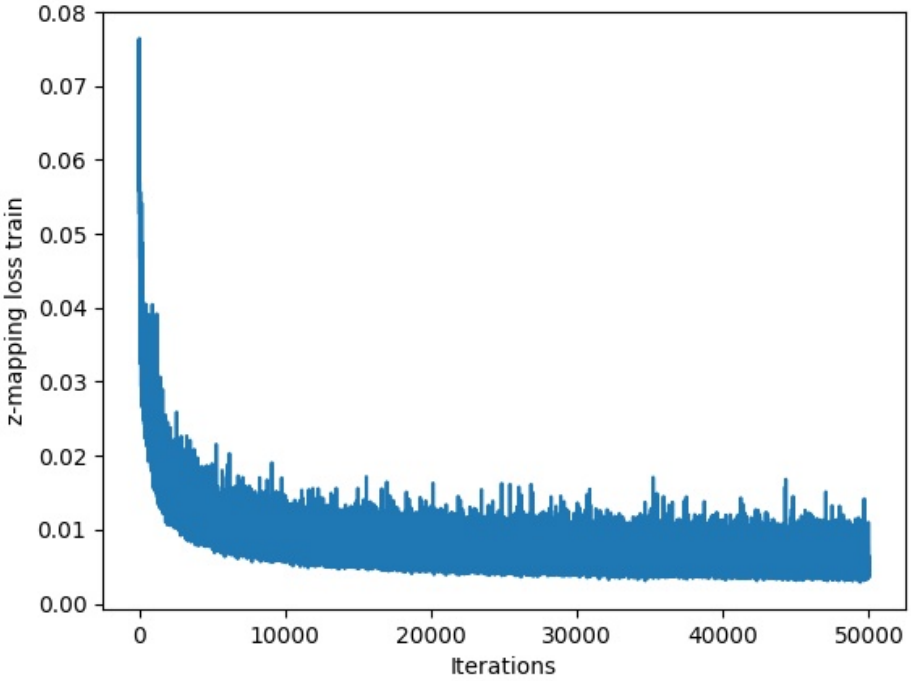


FIGURE 7.21: f-AnoGAN: Training Mapping with izi_f – *Loss*

[9] Code for the f-AnoGAN implementation: https://github.com/tSchlegl/f-AnoGAN/.

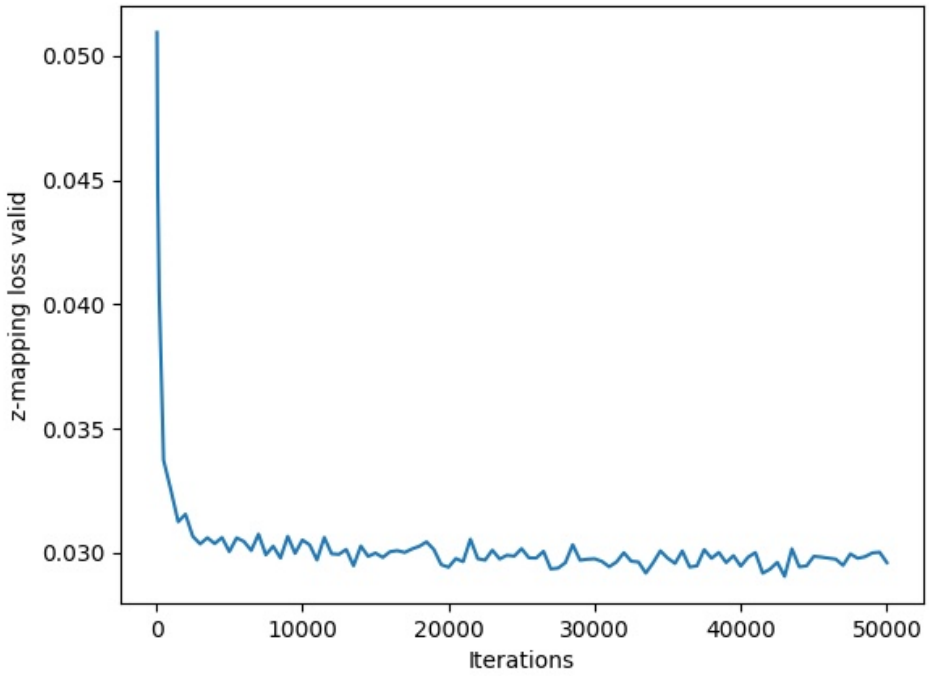


FIGURE 7.22: f-AnoGAN: Validation Loss - Training Mapping with izi_f

Furthermore, figure 7.23 represents the mapping over nine fixed validation images in the initial state of the training, whereas figure 7.24 illustrates the final iteration. Overall, during the final iteration, the model successfully has mapped images. However, the mapping does not recover an identical image but can recover an image quite close to the true distribution.

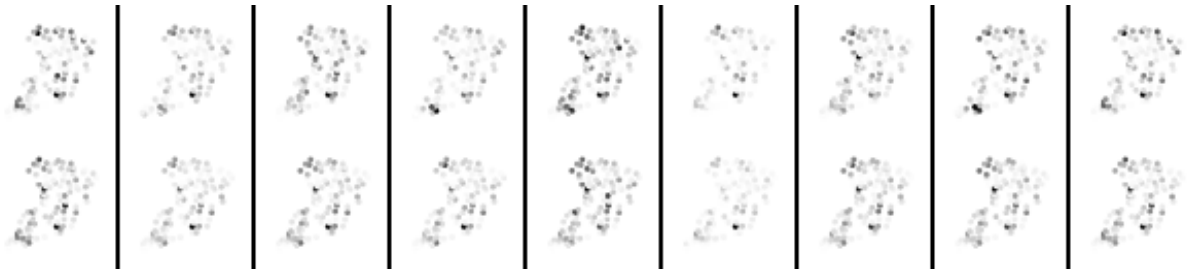


FIGURE 7.23: f-AnoGAN: Real and Reconstructed samples - Iteration 100

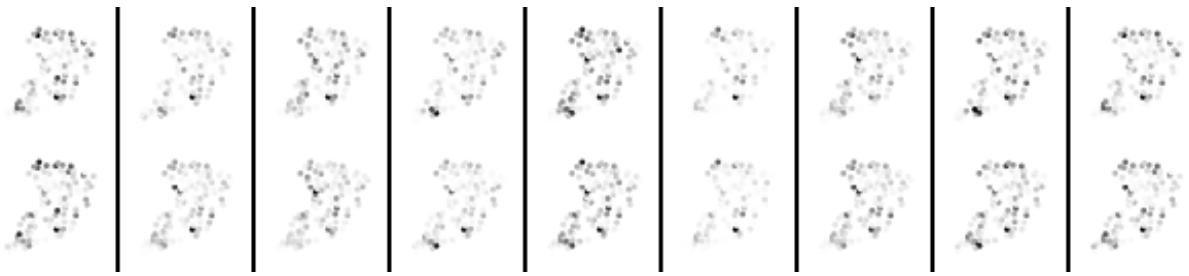


FIGURE 7.24: f-AnoGAN: Real and Reconstructed samples - Iteration 50000

7.5 Results of Anomaly Detection $A(x)$

For the anomaly detection experiment, we tested 576 validation images and 64 anomalous images. Furthermore, the anomaly detector was tested on 64 predicted images, from the ST-ResNet mode, for different timestamps. Table 7.10 shows the maximum, the minimum and the mean A(x) score value that was found for this experiment, based on equation 6.8.

Moreover, for the same validation and anomaly batch size, the same results will occur compared with AnoGAN, where different results will occur for every new mapping. Thus, another interesting fact is that for normal images, the $\mathcal{A}_R(x)$ is higher compared with the final $\mathcal{A}(x)$.

However, when anomalous images were tested, $\mathcal{A}(x)$ scores were higher compared with the $\mathcal{A}_D(x)$. Overall, the results of our implementation can be found on GitHub [10]

TABLE 7.10: Experiment - Anomaly Score A(x) of New Unseen & Predicted Images

Images	Min	Mean	Max
Validation images	*0.0056*	*0.0299*	*0.1337*
Predicted ST-ResNet images	*0.0425*	*0.0845*	*0.2380*
Mean	***0.0241***	***0.0572***	***0.1858***
Anomaly Images	*0.2132*	*0.9227*	*2.0997*

Last, Figure 7.26 shows the anomaly detection visualisation for a healthy unseen sample, whereas figure 7.26 illustrates the anomaly score over an anomalous sample.

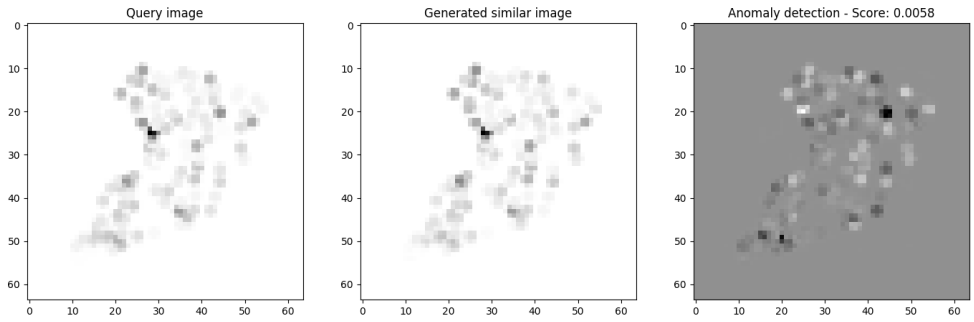


FIGURE 7.25: f-AnoGAN: Anomaly Score - Healthy Sample

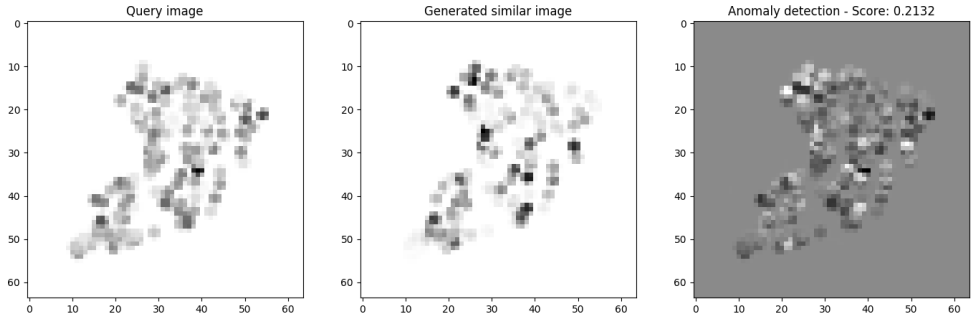


FIGURE 7.26: f-AnoGAN: Anomaly Score - Anomaly Sample

[10] GitHub repository:https://github.com/alexofficial/ADaRCA

7.6 Discussion

Overall, we successfully trained both AnoGAN and f-AnoGAN anomaly detection models. Specifically, f-AnoGAN results (table 7.10) are more accurated, compared with AnoGAN (table 7.7). More specifically, the difference of AnoGAN between mean (118.8445) and anomalous images (470.152) is approximately four times higher, if we divide 470.152 with 87.833. In contrast, the difference of f-AnoGAN between mean (0.0572) and anomalous images mean (0.9227) is approximately 16 times higher. Therefore, we can say that f-AnoGAN provides more accurate results as the difference between the mean is higher, but still, both models can detect anomalies.

However, f-AnoGAN use WGAN with Gradient Penalty loss which reflects the distance between data, whereas AnoGAN architecture based on the DCGAN architecture makes use of vanilla GANs, which does not reflect directly to the distribution. Therefore, f-AnoGAN seems more suitable for the anomaly detection task compared with AnoGAN, as it is more accurated, faster and has a learned mapping compared with AnoGAN, which needs to map for every new image, as is presented in 7.11.

TABLE 7.11: Experiment - Overview between AnoGAN vs f-AnoGAN Implementation

	AnoGAN	**f-AnoGAN**
GAN model	DCGAN	Improved Wasserstein GAN
Require Mapping to Latent space	Yes	Yes
Mapping to latent space	Iterative procedure	Learned mapping
Real time anomaly detection	slow	fast

On the other hand, we cannot compare both AnoGAN and f-AnoGAN directly, as the loss between the models are different, as it was described in 4. However, we have evaluated the models by testing the same query images and evaluate them visually.

Therefore, both models have been tested for four different timestamps. Figure 7.33 illustrates the anomaly detection over the first timestamp, where the reconstructed image does not capture some parts of the input distribution. More specifically, if we compare the query image with the generated, some pixels are not following the query image distribution. On the other hand, figure 7.34 shows the results of f-AnoGAN, where we can see that the model captures the underlined distribution, as there are no semantic differences between the query and the generated image.

FIGURE 7.27: Test AnoGAN - Query Image - Timestamp: 00:00:00

FIGURE 7.28: Test f-AnoGAN - Query Image - Timestamp: 00:00:00

Moreover, the next figures illustrate three more different timestamps, where the same behaviour occurs. Additionally, based the reconstruction data and actual data of AnoGAN and f-AnoGAN, abnormalities can be seen visually. However, due to the limitation of 64x64 pixel size images information are lost, as higher resolution images will serve better of root cause anomaly.

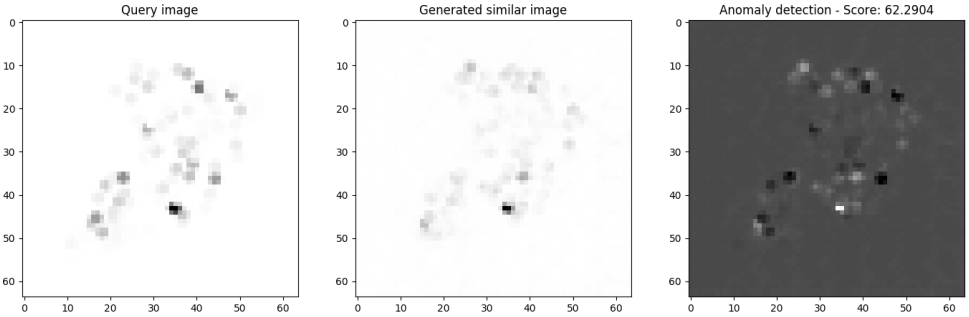


FIGURE 7.29: Test AnoGAN - Query Image - Timestamp: 06:00:00

FIGURE 7.30: Test f-AnoGAN - Query Image - Timestamp: 06:00:00

Overall, f-AnoGAN seems to capture the underlined distribution of the input, as in all four cases it has visually better-reconstructed images.

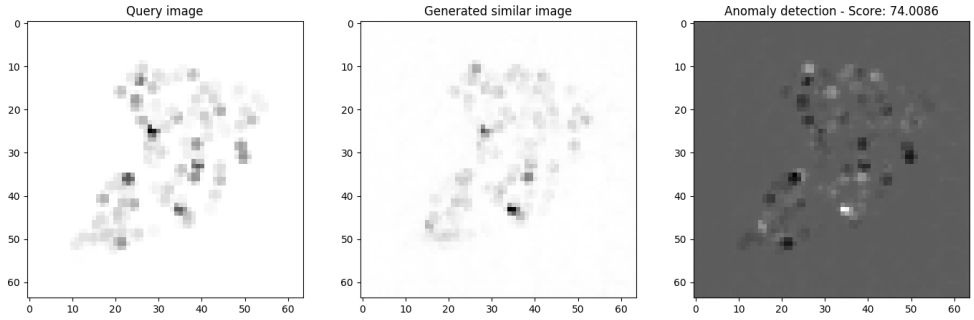


FIGURE 7.31: Test AnoGAN - Query Image - Timestamp: 12:00:00

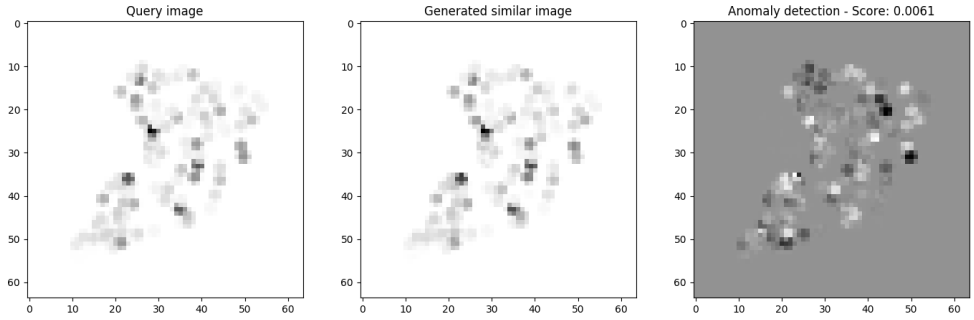


FIGURE 7.32: Test f-AnoGAN - Query Image - Timestamp: 12:00:00

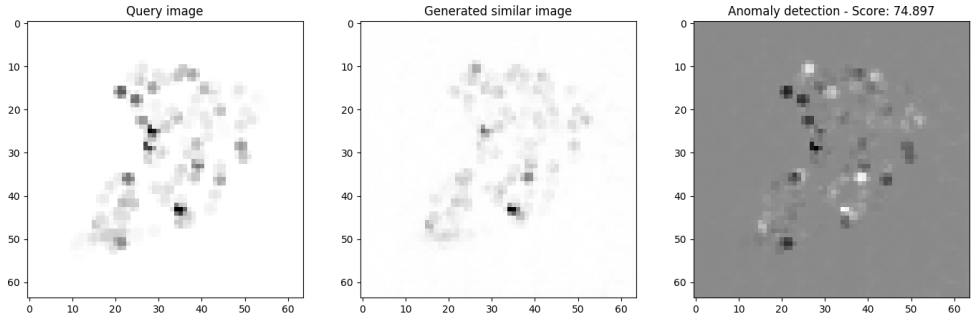


FIGURE 7.33: Test AnoGAN - Query Image - Timestamp: 18:00:00

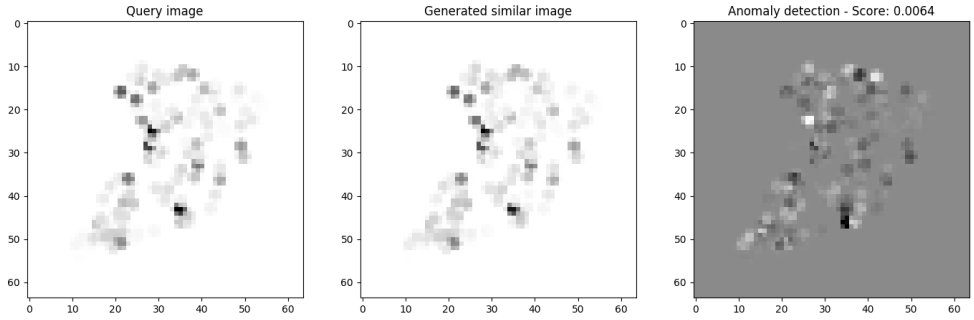


FIGURE 7.34: Test f-AnoGAN - Query Image - Timestamp: 18:00:00

Chapter 8

Conclusion and Future work

8.1 Conclusion

The objective of this thesis was to apply Deep Generative Models to detect anomalies in a network over PM counters for different locations in an area and root cause the location of the anomaly, where anomalies potentially occur after a big update in the network. Therefore, a converter framework was created, which converts PM counters for every timestamp and all the different locations in the area of interesting to images. Additionally, the framework is flexible for experiments, as was described in 5.2, and it allows the generation of different types of images, sizes, PM counters, and so on.

Consider the anomaly detection and root cause analysis problem and based on the future work suggestions of a closely related work [50], where

- GANs suggested as an alternative for anomaly detection and root cause analysis, and forecasting prediction models for predicting the next timestamp of a network considering temporal dependencies.
- Replacement of manually decision rules to define an anomaly with NN, which will learn the relationship between prediction and ground truth.

In this work, Deep Generative Models were applied for anomaly detection successfully, and root cause analysis based not only in the initial implementing of GAN used in AnoGAN work. Moreover, a state-of-the-art implementation of GANs, called WGAN-GP and used in f-AnoGAN, applied successfully for anomaly detection and root cause analysis.

Furthermore, extensive literature for both prediction models, which consider spatial-temporal dependencies and Generative Adversarial Networks is done, were table 8.1 contains the GANs models that have been used in this project.

TABLE 8.1: Experiment - Generative Adversarial Network (GANs) - Papers

Papers	Year
Vanilla GANs [5]	2014
DCGAN [24]	2015
AnoGan [2]	2017
Wasserstein GAN [25]	2017
Improved Wasserstein GAN [26]	2017
f-AnoGan [3]	2019

Moreover, for the Spatial-Temporal prediction models (table 7.2), ST-ResNet was chosen, due to the time limitation of this thesis to implement both DMVST-Net and STDN, as described in 7. In this way, ST-ResNet prediction model was successfully trained with our dataset, as it was shown in 7.2. Also, the prediction used as an input to both AnoGAN and f-AnoGAN to define a threshold which can determine an anomaly automatically, considering the replacement of

the manual decision rules. Nevertheless, this is just an attempt for dealing with the replacement of the manually defined decision, which in general is a challenging task.

8.2 Future Work

As future work, an objective could test data, which are actual anomalies, as in our case, anomalies where generated. Furthermore, the models could be tested with different anomalies types. For instance, anomalies that were occurred after an update to the whole network or anomalies that were occurred due to software or hardware issues. In this way, the network can be tested in different scenarios, and even more to classify the type of the anomaly, which would be usefully for maintaining and protecting the network's stability.

Moreover, the implementation of this work was considered abnormalities in an area of interesting, such as a city. However, it could be possible in the case of alerting an anomaly city-wise to use a similar model trained with individual PM counters. Specifically, PM counters can be converted to images based on [51], encoding time series as images. There are two types of images, which have been proposed, Gramian Angular Field (GAF) and Markov Transition Fields (MTF).

However, in this work, it was not possible to implement sufficient both GAF and MTF, due to the limitation of the network having a 15-minute interval, which is an average of PM counters. Therefore, 96 timestamps per day do not allow to generate enough data for the training. Hence, this could be a potential use of GANs with encoded PM counter images for future work, where data can be kept in second intervals.

Furthermore, combining different models would be possible to create a framework, which starts by checking anomalies at a higher level, like a city, as was implemented in this work. Next, if an abnormality occurs, a loop can be activated over all the PM counters in that specific area to identify the anomalous counters. Furthermore, the model could potentially use the proposed GAF or MTF encoding. Moreover, another interesting approach could be to convert multiple PM counters to images, as it would be helpful to analyse more than one PM counter, as the Deep Learning model could find the potential correlation between different PM counters.

Thus, in this work, ST-ResNet was implemented for the next timestamp prediction, whereas DMVST-Net and STDN were left for future studies. Also, another approach for prediction, which is based [52], aims to predict by learning information from multiple cities and this can be beneficial for prediction as external factors for one specific area are sometimes limited or even not available.

Additionally, GANs has been used for video frame prediction with Deep Learning [53], but video frame prediction has also been used in [54]. Overall, it could be interesting to treat each of the timestamps as a video frame and then predict the next frame or the next timestamp by using the prior implementations. In this work, this was consigned, but as the points are stationary and only the density of the pixels is changing, we leave this for future work.

Moreover, evaluation GANs was discussed in 4.10, where more investigation can be a potential future implementation work, where evaluation methods such as Fréchet Inception Distance (FID), Geometry Score, and so on; can be tested. Additionally, evaluation with a one-dimensional score can fail to distinguish cases such as having two models with a similar score, trained on the same datasets, where different generated characteristics observed, as shown in [55]. In this way, the authors proposed a novel definition for Recall and Precision, which untangle the divergence into two divided dimension.

Last but not least, considering WGAN with Gradient Penalty implementation, the authors implemented a one-side penalty for penalised gradients over one, but also the others considered implementing a two side penalty for penalised gradient smaller than one, but as a complete comparison was needed, they left the compression for future work. Therefore, it is expected a thoroughly study between one-side and two-side penalise, which can potentially improve the

initial implementation of WGAN with Gradient Penalty. Last, it would be possible to generate higher size images. For instance, SN-GANs [56] generates images with 128x128 pixel size. In this way, both AnoGAN and f-AnoGAN model can be tested with higher size images, which can improve both detection and root cause analysis.

Bibliography

[1] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly Detection for Discrete Sequences: A Survey". In: *IEEE Trans. on Knowl. and Data Eng.* 24.5 (May 2012).

[2] Thomas Schlegl et al. "Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery". In: *CoRR* abs/1703.05921 (2017).

[3] Thomas Schlegl et al. "f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks". In: *Medical image analysis* 54 (2019), pp. 30–44.

[4] Ian J. Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks". In: *CoRR* abs/1701.00160 (2017).

[5] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[6] Ali Borji. "Pros and cons of gan evaluation measures". In: *Computer Vision and Image Understanding* 179 (2019), pp. 41–65.

[7] Andries P. Engelbrecht. *Computational Intelligence: An Introduction*. 2007.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press.

[9] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[10] Balint Gersey. "Master Thesis title: Generative Adversarial Networks". May 2018.

[11] T. Teofili. *Deep Learning for Search*.

[12] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *arXiv:1603.07285* (2016).

[13] Shakir Mohamed and Balaji Lakshminarayanan. "Learning in implicit generative models". In: *arXiv:1610.03483* (2016).

[14] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014).

[15] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR, abs/1512.03385* (2015).

[16] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv:1409.1556* (2014).

[17] Junbo Zhang et al. "DNN-based prediction model for spatio-temporal data". In: *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM. 2016, p. 92.

[18] Junbo Zhang, Yu Zheng, and Dekang Qi. "Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*. 2017, pp. 1655–1661.

[19] Huaxiu Yao et al. "Deep Multi-View Spatial-Temporal Network for Taxi Demand Prediction". In: *CoRR* abs/1802.08714 (2018).

[20] Huaxiu Yao et al. "Revisiting Spatial-Temporal Similarity: A Deep Learning Framework for Traffic Prediction". In: *CoRR* abs/1803.01254 (2018).

[21] Huaxiu Yao et al. "Modeling Spatial-Temporal Dynamics for Traffic Prediction". In: *CoRR* abs/1803.01254 (2018).

[22] "Deep Generative Models". 2017. URL: http://introtodeeplearning.com/2017/schedule.html.

[23] Serena Yeung et al. Fei-Fei Li Justin Johnson. *Lecture notes CS231n: Convolutional Neural Networks for Visual Recognition*. 2017.

[24] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: *CoRR* abs/1511.06434 (2015).

[25] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein gan". In: *arXiv:1701.07875* (2017).

[26] Ishaan Gulrajani et al. "Improved training of wasserstein gans". In: *Advances in neural information processing systems*. 2017, pp. 5767–5777.

[27] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "Unsupervised learning". In: *The elements of statistical learning*. Springer, 2009, pp. 485–585.

[28] *Deep Generative Models*. https://towardsdatascience.com/deep-generative-models-25ab2821afd3 Accessed: 2019-01-9.

[29] Carl Doersch. "Tutorial on variational autoencoders". In: *arXiv:1606.05908* (2016).

[30] Tim Salimans et al. "Improved techniques for training gans". In: *Advances in neural information processing systems*. 2016, pp. 2234–2242.

[31] Martin Heusel et al. "Gans trained by a two time-scale update rule converge to a local nash equilibrium". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6626–6637.

[32] Yichuan Tang. "Deep learning using linear support vector machines". In: *arXiv:1306.0239* (2013).

[33] Ishan Durugkar, Ian Gemp, and Sridhar Mahadevan. "Generative multi-adversarial networks". In: *arXiv:1611.01673* (2016).

[34] Zhifei Zhang, Yang Song, and Hairong Qi. "Decoupled Learning for Conditional Adversarial Networks". In: *CoRR* abs/1801.06790 (2018).

[35] Valentin Khrulkov and Ivan V. Oseledets. "Geometry Score: A Method For Comparing Generative Adversarial Networks". In: *CoRR* abs/1802.02664 (2018).

[36] Tero Karras et al. "Progressive Growing of GANs for Improved Quality, Stability, and Variation". In: *CoRR* abs/1710.10196 (2017).

[37] Mario Lucic et al. "Are gans created equal? a large-scale study". In: *Advances in neural information processing systems*. 2018, pp. 700–709.

[38] Jake Snell et al. "Learning to generate images with perceptual similarity metrics". In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 4277–4281.

[39] Tom White. "Sampling Generative Networks: Notes on a Few Effective Techniques". In: *CoRR* abs/1609.04468 (2016).

[40] Jost Tobias Springenberg et al. "Striving for simplicity: The all convolutional net". In: *arXiv:1412.6806* (2014).

[41] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[42] Edouard Duchesnay and Tommy Löfstedt. "Statistics and Machine Learning in Python, Release 0.2". In: (Jun 22, 2018). URL: ftp://ftp.cea.fr/pub/unati/people/educhesnay/pystatml/StatisticsMachineLearningPythonDraft.pdf.

[43] Gilberto Fernandes et al. "A comprehensive survey on network anomaly detection". In: *Telecommunication Systems* 70.3 (2019), pp. 447–489.

[44] Raghavendra Chalapathy and Sanjay Chawla. "Deep Learning for Anomaly Detection: A Survey". In: *CoRR* abs/1901.03407 (2019). arXiv: 1901.03407. URL: http://arxiv.org/abs/1901.03407.

[45] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3 (2009), p. 15.

[46] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. "Adversarial feature learning". In: *arXiv:1605.09782* (2016).

[47] Martín Abadi et al. "Tensorflow: a system for large-scale machine learning." In: *OSDI*. Vol. 16. 2016, pp. 265–283.

[48] Doyup Lee. *AnoGAN implementation*. https://github.com/LeeDoYup/AnoGAN.

[49] Aksel Wilhelm Wold Eide. "Applying generative adversarial networks for anomaly detection in hyperspectral remote sensing imagery". NTNU - Norwegian University of Science and Technology, 2018.

[50] SERGIO LÓPEZ ÁLVAREZ. "Anomaly Detection in Evolved Node B's Resource Consumption". KTH Royal Institute of Technology School of Electrical Engineering and Computer Science, 2018.

[51] Zhiguang Wang and Tim Oates. "Encoding time series as images for visual inspection and classification using tiled convolutional neural networks". In: *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.

[52] Huaxiu Yao et al. "Learning from Multiple Cities: A Meta-Learning Approach for Spatial-Temporal Prediction". In: *CoRR* abs/1901.08518 (2019).

[53] Junhyuk Oh et al. "Action-Conditional Video Prediction using Deep Networks in Atari Games". In: *CoRR* abs/1507.08750 (2015).

[54] Michael Mathieu, Camille Couprie, and Yann LeCun. "Deep multi-scale video prediction beyond mean square error". In: *arXiv preprint arXiv:1511.05440* (2015).

[55] Mehdi SM Sajjadi et al. "Assessing generative models via precision and recall". In: *Advances in Neural Information Processing Systems*. 2018, pp. 5228–5237.

[56] Takeru Miyato et al. "Spectral normalization for generative adversarial networks". In: *arXiv:1802.05957* (2018).

Appendix A

Appendix: Importing Data Class

```
import glob2
import os
import pdb
from utils import *
import keras
import numpy as np

class DataC(object):
    def __init__(self, input_height,input_width,resize_height,resize_width,
    boolCrop, boolgrayscale,y_dim, images_path):
      self.input_height=input_height
      self.input_width=input_width
      self.resize_height=resize_height
      self.resize_width=resize_width
      self.crop = boolCrop
      self.grayscale = boolgrayscale
      self.y_dim = y_dim
      self.data_y = None
      self.one_hot = True
      self.images_path = images_path

    def load_trains(self):
      print(self.images_path)
      data_X = glob2.glob(os.path.join(self.images_path))
      sample_files = data_X[0:]
      sample = [get_image(sample_file,self.input_height,self.input_width,self.
    resize_height,self.resize_width,self.crop, self.grayscale) for sample_file in
     sample_files]

      if (self.grayscale):
       sample_inputs = np.array(sample).astype(np.float)[:, :, :, None]
      else:
       sample_inputs = np.array(sample).astype(np.float)

      self.data_y = []
      rangeA = len(sample_inputs)/self.y_dim

      for i in range(int(rangeA)):
       count=0
       while count<=self.y_dim-1:
        self. data_y.append(count)
        count=count+1

      X = np.asarray(sample_inputs)
      y = np.asarray(self.data_y)

      if self.one_hot:
        y = keras.utils.to_categorical(y, 96)
      return X,y
```

Appendix B

Appendix: AnoGAN - Threshold for Anomaly Detection

```
if FLAGS.anomaly_test:
        dcgan.anomaly_detector()
        mean_anomaly_score = np.NZERO
        min_anomly_score = np.PINF
        max_anomaly_score = np.NINF
        assert len(dcgan.test_data_names) > np.NZERO

        for idx in range(len(dcgan.test_data_names)):
            test_input = np.expand_dims(dcgan.test_data[idx], axis=0)
            test_name = dcgan.test_data_names[idx]
            anomaly_score= dcgan.train_anomaly_detector(FLAGS,
                test_input, test_name)

            if min_anomly_score >= anomaly_score:
                min_anomly_score = anomaly_score
            if max_anomaly_score <= anomaly_score:
                max_anomaly_score = anomaly_score

            print(anomaly_score)
            mean_anomaly_score = mean_anomaly_score+anomaly_score

        final_mean_anomaly_threshold = mean_anomaly_score/len(dcgan.
    test_data_names)
        print('Threshold for Anomaly Score is:'+ str(
    final_mean_anomaly_threshold))
        print('Min score:'+ str(min_anomly_score))
        print('Max score:'+ str(max_anomaly_score))

        numpy_array_value = np.array([final_mean_anomaly_threshold,
    min_anomly_score,max_anomaly_score])
        numpy_array_value = numpy_array_value.reshape(-1,1)
        min_max_scaler = MinMaxScaler()
        normalized_array = min_max_scaler.fit_transform(numpy_array_value)

        print('The normalized Threshold for Anomaly Score is:'+ str(
    normalized_array[0]))
        print('The normalized Min score:'+ str(normalized_array[1]))
        print('The normalized Max score:'+ str(normalized_array[2])
```

Appendix C

Appendix: f-AnoGAN -Linear Z-space Interpolation

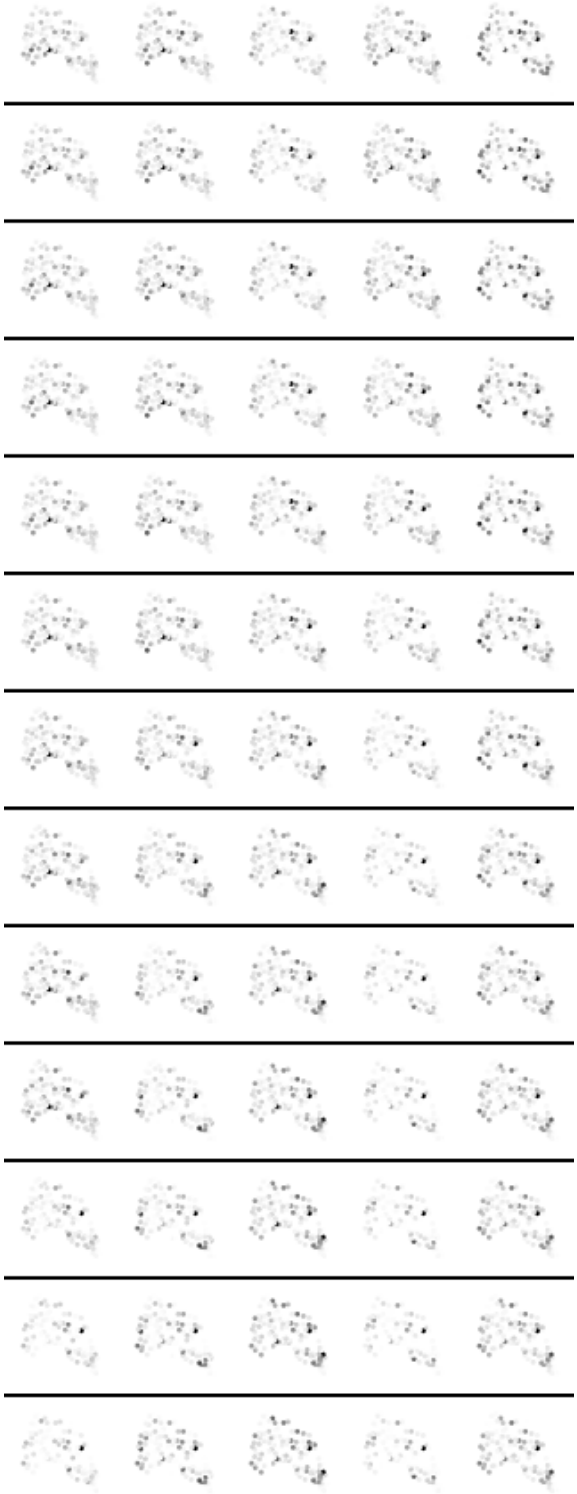


FIGURE C.1: f-AnoGAN: WGAN-GP - Linear z-space Interpolation for Random Endpoints - Iteration 220000

Appendix D

Appendix: According the Real Time Anomaly Detection

> *An observer, casually watching the patrons at a neighboring table in a fashionable restaurant, notices that the first guest to taste the soup winces, as if in pain. The normality of a multitude of events will be altered by this incident. It is now unsurprising for the guest who first tasted the soup to startle violently when touched by a waiter; it is also unsurprising for another guest to stifle a cry when tasting soup from the same tureen. These events and many others appear more normal than they would have otherwise, but now necessarily because they confirm advance expectations. Rather, they appear normal because they recruit the original episode, retrieve it from memory, and are interpreted in conjunction with it."*
>
> —Daniel Kahnenman,
> *Thinking Fast & Slow*

In this Appendix, the intuition is to rephrase the real-time anomaly detection problem with an example of Daniel Kahnenman and Dale Miller example in their book Thinking Fast and Slow.

In this example, the anomaly detector is the observer which observing the patrons. Moreover, when the first quest tastes the soup and winces, an abnormality has occurred. Therefore, the violent startle of the first guest which has now tasted the soup is expected. In the same way, when an abnormality has first observed, it is expected to be continued.

Moreover, now the observer is aware of the abnormality situation, as an anomaly detector model when first detect an abnormality. Therefore, it is unsurprising for another guest to be in pain when he or she tastes the same soup. In the same way, if we consider the same intuition for a network, it is unsurprising to observe more abnormalities, considering the initial anomaly.

Furthermore, all these events appear reasonable when the first anomaly has occurred, and the expected behaviour is confirmed, as it appeared reasonable to consider the recruit of the first event.

In this way, it is crucial when dealing with real-world anomalies to be aware of different scenarios, even if unknown anomalies might occur, they might have relevant outcomes. Therefore, as the observer would not be surprised when another anomaly occurred, an anomaly detector can learn to deal with different anomaly scenarios or even be aware by learning as different anomalies occur. Last, as anomalies are not often observed, it would be possible to generate different scenarios where the model can be trained similarly as a robot trained in a simulated environment before is tested on the real environment.