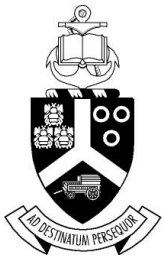


UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Faculty of Engineering, Built Environment and Information Technology

The Department of Mechanical and Aeronautical Engineering

Rail surface anomaly detection: a deep learning approach for computer vision

A dissertation submitted in partial fulfilment for the degree Master of Engineering

Richard Deetlefs

under the supervision of
P.S. Heyns

November 2018

Rail surface anomaly detection: a deep learning approach for computer vision

by
Richard Deetlefs

Abstract

Rail surface defects have become more of an issue in recent years due to new manufacturing techniques which produce head-hardened rails and as industry demands higher speeds, heavier loads and increased traffic. These defects can cause catastrophic accidents, which have consequences such as death, injury, huge cost implications and loss of public confidence. Computer vision systems have become popular, as cameras are non-contact full-field sensors which are low in cost, have high sampling rates and provide appealing performance. However, accurate inspection remains challenging due to dynamic non-linear environmental and rail surface conditions in which images are captured, which result in a heterogeneous image dataset. It is also difficult to select useful features which satisfy the variations due to different failure modes. In addition, there is a class imbalance issue, as most captured images do not contain any defects.

In this dissertation, we develop deep generative models that are trained exclusively using healthy images of a rail surface so that we learn useful features to capture the complex nature of the images which are acquired. We propose multiple models which operate with images at different resolutions. We present a new dataset which will be made publicly available. Experimental results demonstrate that our proposed models can perform accurate detection using our dataset. The proposed algorithms are highly parallel and computationally efficient, which enables real-time inspection at speeds that exceed the world's fastest railway trains: Fuxing Hao CR400AF/BF that has a continuous operation speed of approximately 400 km/h.

Keywords: Deep learning, computer vision, rail surface anomaly detection, unsupervised segmentation, real-time inspection.

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Rail Condition Monitoring Techniques . . . 1
- 1.3 Rail Surface Defects . . . 2
  - 1.3.1 Challenge of Detecting Discrete Surface Defects with Visual Systems . . . 2
  - 1.3.2 Maintenance for Rail Surface Defects . . . 3
- 1.4 Computer Vision in Railway Condition Monitoring . . . 3
  - 1.4.1 Defect Localisation using Hand-Crafted Features . . . 4
  - 1.4.2 Defect Classification using Hand-Crafted Features . . . 4
  - 1.4.3 Defect Classification using Learned Features . . . 5
  - 1.4.4 Discussion of Prior Work . . . 5
- 1.5 Proposed Method . . . 6
  - 1.5.1 Contributions . . . 6
- 1.6 Document Overview . . . 6

2 Data Acquisition **7**
- 2.1 Experimental Rig Design . . . 7
- 2.2 Experiment Set-up and Results . . . 7
- 2.3 Data Preparation . . . 8

3 Deep Learning **11**
- 3.1 Variational Auto-Encoders . . . 12
  - 3.1.1 VAE Training . . . 14
  - 3.1.2 VAE Discussion . . . 15
  - 3.1.3 VAEs in Anomaly Detection . . . 16
- 3.2 Generative Adversarial Networks . . . 16
  - 3.2.1 GAN Training . . . 17
  - 3.2.2 GAN Discussion . . . 17
  - 3.2.3 GANs in Anomaly Detection . . . 18

4 Proposed Approach **20**
- 4.1 Improving VAE and GAN Models . . . 20
  - 4.1.1 Reconstruction Error . . . 20
  - 4.1.2 VAE-GAN Models . . . 21
  - 4.1.3 Relativistic Discriminator . . . 21
- 4.2 Proposed Model . . . 22
  - 4.2.1 NET Training . . . 22
- 4.3 Architecture Development . . . 23
  - 4.3.1 Architecture Motivation . . . 23
  - 4.3.2 Model Architecture . . . 24
- 4.4 Training Procedure . . . 24

4.4.1 Progressively Growing Networks . . . 25
4.4.2 Training Set-up and Observations . . . 26

5 Inference Algorithm **28**
5.1 Pre-processing . . . 28
5.2 Processing . . . 30
5.3 Post-processing . . . 31
5.3.1 Image Comparison . . . 31
5.3.2 Thresholding . . . 32
5.3.3 Morphological Operations . . . 33
5.3.4 Anomaly Score . . . 35
5.4 Discussion . . . 35
5.4.1 Threshold Constants . . . 35
5.4.2 Chosen Parameters . . . 36

6 Experimental Investigation **37**
6.1 Receiver Operating Characteristic . . . 37
6.2 Dimensionality of Encoded Latent Space . . . 39
6.3 Number of Monte Carlo Samples . . . 40
6.4 Inference Speed . . . 41
6.5 Discussion . . . 44

7 Conclusion and Recommendations **45**
7.1 Recommendations . . . 46

Appendices **47**

A Additional VAE Equations **48**
A.1 KL Loss . . . 48

B Additional Figures **49**
B.1 ProGAN and BigGAN Generated Images . . . 49
B.2 Coordinate Convolution . . . 50
B.3 Image Smoothing . . . 51
B.4 Morphological Operations . . . 51
B.5 Defect Localisation for Different Models . . . 52
B.6 Reconstructed Results . . . 54

C Additional Network Architectures **56**
C.1 Model Architecture at Resolution 256×128 . . . 56
C.2 Model Architecture at Resolution 512×256 . . . 57

D Deep Learning Networks **58**
D.1 Neural Networks . . . 58
D.1.1 Network Architecture . . . 58
D.1.2 Activation Functions . . . 59
D.1.3 Forward Propagation . . . 60
D.1.4 Backward Propagation . . . 60
D.1.5 Updating Weights . . . 61
D.2 Convolutional Neural Networks . . . 62
D.2.1 Convolutional Stage . . . 62
D.2.2 Pooling Stage . . . 62
D.2.3 Fully Connected Layer . . . 63

List of Figures

1.1 Rail details. (a) Labelled rail and wheel section. (b) Labelled track components. 3

2.1 Experimental rig. (a) Schematic of camera placement. (b) Close-up of the cameras. (c) Experiment rig set-up. 8
2.2 Example of captured images. 8
2.3 Cropped example images of the three classes. (a) Healthy. (b) Unhealthy. (c) Artefacts. 9
2.4 Region-of-interest image. (a) Defect. (b) Corresponding ground truth of the defects. 10

3.1 Forward pass of (a) Auto-Encoder. (b) Variational Auto-Encoder. 13
3.2 Forward pass of Generative Adversarial Network 17

4.1 Forward pass of proposed model (NET). 22
4.2 Progressively growing the decoder P and discriminator D. (a) Partial NET initial: 128×64. (b) Transition in new layers. (c) Partial NET: 256×128. Note this figure does not describe the full network but only the last (P) and first (D) two layers. 26

5.1 Pre-processing procedure. (a) The sequence of acquired images. (b) The top half of the images. (c) Grey-scale histogram. (d) First derivative. (e) Average derivative. (f) Located edges. (g) Cropped ROI image. Best viewed by zooming in on the electronic copy. 29
5.2 Processing procedure. 30
5.3 Variance within reconstructed samples, $L = 16$. 30
5.4 Original and reconstructed image. (a) Before smoothing. (b) After smoothing. . 32
5.5 Average error, $L = 16$. 32
5.6 Threshold procedure. (a) Original image. (b) Average error, $L = 16$. (c) Dark threshold. (d) Bright threshold. (e) Final (combined) threshold. 33
5.7 Morphological procedure. (a) Original image. (b) Final threshold. (c) Closing: dilation. (d) Closing: erosion. (e) Opening: erosion. (f) Opening: dilation. 34
5.8 Ground truth segmented defect. 35
5.9 Anomaly score on the validation dataset using a mesh grid of threshold constants. 35

6.1 ROC results on the test dataset. (a) ROC scatter plot and the best ROC curve with optimal threshold constants. (b) Final ROC curve with selected characteristics. 38
6.2 Images at resolution 1024×512. (a) Original image. (b-c) Reconstructed images with: (b) $M_c = 512$, (c) $M_c = 1024$, (d) $M_c = 128$. 39
6.3 ROC curve.(a) Varying dimensionality M_c at 256×128 resolution. (b) Zoomed-in. 39
6.4 ROC curve. (a) Best M_c values for all resolutions. (b) Zoomed-in. 40
6.5 Relationship between fps and the velocity a train travels at. 41

B.1 ProGAN generated celebrity faces at a resolution of 1024×1024 [41]. Reproduced with the authors' permission. 49

B.2 BigGAN generated ImageNet samples at a resolution of 512 × 512 [11]. Reproduced with the authors' permission. 49
B.3 Convolution input. (a) Usual convolution layer. (b) CoordConv layer [55]. 50
B.4 Single white pixel image. (a) Before smoothing. (b) After smoothing. 51
B.5 Morphological operations. (a) Original image. (b) Erosion applied to original image. (c) Dilation applied to original image. 51
B.6 Morphological operations. (a) Original image. (b) Erosion applied to original image. (c) Dilation applied to original image. 52
B.7 Large size defect. (a) Original image. (b) Ground truth. (c-e) Anomaly score for model resolution: (c) 256 × 128, (d) 512 × 256, (e) 1024 × 512. 52
B.8 Medium size defect. (a) Original image. (b) Ground truth. (c-e) Anomaly score for model resolution: (c) 256 × 128, (d) 512 × 256, (e) 1024 × 512. 53
B.9 Small size defect. (a) Original image. (b) Ground truth. (c-e) Anomaly score for model resolution: (c) 256 × 128, (d) 512 × 256, (e) 1024 × 512. 53
B.10 Original (left column) and reconstructed (right column) images from the healthy test dataset at a resolution of 1024 × 512, with $M_c = 512$. 54
B.11 Original (left column) and reconstructed (right column) images from the unhealthy test dataset at a resolution of 1024 × 512, with $M_c = 512$. 55

D.1 Shallow neural network. 59
D.2 Two layer (deep) neural network. 60

List of Tables

2.1 Division of the captured image dataset. 9

4.1 VAE architecture for images at 1024×512 resolution. 25
4.2 Discriminator architecture for images at 1024×512 resolution. 25
4.3 Training duration. 27

6.1 Confusion matrix. 37
6.2 Results from experiments on dimensionality M_c. 40
6.3 Results of experiments based on the number of samples used. 41
6.4 Inference algorithm speed for all models. 42
6.5 Inference algorithm speed for all models using multiple GPUs. 43

C.1 VAE architecture for images at 256×128 resolution. 56
C.2 Discriminator architecture for images at 256×128 resolution. 56
C.3 VAE architecture for images at 512×256 resolution. 57
C.4 Discriminator architecture for images at 512×256 resolution. 57

Nomenclature

Abbreviations

AUC	Area under the curve
AE	Auto-encoder
CNN	Convolution neural network
CTFM	Coarse-to-fine model
DCGAN	Deep convolution generative adversarial network
DCNN	Deep convolution neural network
DNN	Deep neural network
ELBO	Evidence lower bound
FN	False negative
FP	False positive
FPR	False positive rate
GAN	Generative adversarial network
GPU	Graphical processing unit
KL	Kullback-Leibler
LReLU	Leaky rectified linear unit
LSGAN	Least-squares generative adversarial network
MSE	Mean squared error
NN	Neural network
PSNR	Peak signal to noise ratio
RCF	Rolling contact fatigue
ReLU	Rectified linear unit
RGAN	Relativistic generative adversarial network
ROC	Receiver operating characteristics
ROI	Region of interest
SSIM	Structural similarity metric
TN	True negative
TP	True positive
TPR	True positive rate
VAE	Variational auto-encoder

English letters and symbols

D	Discriminator
E	Error map
F	Dimensionality of hidden representation of layers for perceptual loss
G	Generator
H	Histogram
I	Light intensity
K	Dimensionality of images
L	Number of Monte Carlo samples

$\mathcal{L}$	Loss function
M	Dimensionality of latent space
M_c	Dimensionality of latent space channels
M_d	Morphological dilation
M_e	Morphological erosion
N	Number of images in a batch
P	Decoder
Q	Encoder
T_b	Bright threshold
T_d	Dark threshold
c_d	Dark threshold constant
c_b	Bright threshold constant
f	Hidden representation of layers for perceptual loss
fps	Frames per second
p	Probability distribution
p_{data}	True data distribution
p_g	Generated distribution
pi	Row of pixels
ppi	Pixels per inch
x	Horizontal image coordinates
y	Vertical image coordinates
I	Identity matrix
$\boldsymbol{s}$	Structuring element
$\boldsymbol{w}$	Circular-symmetric Gaussian weighting function
$\boldsymbol{x}$	Image data
$\boldsymbol{x'}$	Reconstructed image data
$\boldsymbol{z}$	Latent variable

Greek Symbols

α	Transition constant
β	Weighting function
ϵ	Auxiliary variable
θ	Deep network parameters (weights and biases)
θ^*	Trained deep network parameters
λ	Weighting function
μ	Mean
$\sum$	Diagonal covariance
σ	Standard deviation
σ^2	Variance
ϕ	Deep network parameters

Chapter 1

Introduction

Rail transport is used throughout the world, for public transport as well as the transportation of cargo (freight trains). Safe transit of these valuable goods whether it be humans, coal, iron ore, wood or other assets is of utmost importance. Economies rely on the operation of the railway infrastructure. The backbone of the infrastructure is the rails or tracks on which these trains run on. To ensure safe operation, the rails must be monitored and maintained frequently.

In this chapter, we give a background on railway faults and techniques used in railway condition monitoring. We consider prior work in computer vision applied to railways and briefly discuss our proposed approach along with our contributions.

1.1 Background

In the past, the clear majority of rail failures was due to internal defects. Nowadays rails are manufactured to reduce internal defects by using head-hardened rails and higher carbon steel rails, which are highly resistant to wear [17, 18]. Due to this, rail failures caused by surface defects are now more common [62]. According to the U.S. Department of Transportation, Federal Railroad Administration from 1995-2002, surface defects were strongly implicated in 122 derailments and may have contributed to 160 more [61]. One of the major incidents caused by surface defects is that of the Hatfield (U.K.) derailment in the year 2000, in which 4 lives were lost and 107 people injured [61]. In the late 1990s surface defects caused approximately 60% of defects found by East Japan Railways, while in France and the U.K. the figures were about 25% and 15%, respectively [38]. Besides safety implications, surface defects result in huge economic costs. A European study reported that the cost of surface defects including inspection, train delay, rail replacements and weld repair, rail grinding and derailments, is about €300 million per year [10], the cost implications in the U.S. is expected to be much greater [62].

1.2 Rail Condition Monitoring Techniques

Rails are inspected for defects using non-destructive tests during the manufacturing process, as well as when the rails are in use. Rail defects occur either internally or externally. Internal defects are commonly identified using ultrasonic testing equipment. This technique has been shown to be very successful for internal defects and can detect near surface defects [75].

Surface defects are traditionally inspected manually by experts [88]. This visual inspection process is crucial, but very labour intensive, costly and has limitations in speed, quality, objectivity and scope [70]. This approach is popular as inspection vehicles which focus on internal defects (ultrasonic testing) travel rather slowly allowing time for visual inspection to take place simultaneously.

Eddy current techniques have also been used to detect defects [7, 53]. Eddy current allows one to determine small deviations from the surface of the rail at a high sampling rate, however, it is not capable of picking up larger deviations such as a defect which penetrates a few millimetres into the rail. This approach is also difficult to implement in practice, as the system is very sensitive to variations in the distance between the probe and the track. Typically, the distance is $1 - 2\ mm$, which is difficult to maintain at high speeds with rough and irregularly shaped rails [61]. False negatives are an issue as material properties of the railway change, due to rail replacements during maintenance. Eddy current measurements are taken only at a single point, thus multiple probes are needed to cover the surface of the rail.

Vibration measurements [42, 71] and acoustic emission [5, 95, 96] are also used, but these systems are only capable of detecting severe defects [61]. Another more recent approach uses line laser sensors for rail profile measurements [78, 81]. These systems simply check for wear on the rails, rather than defects.

Computer vision is a very active area of research which is relevant to several industries ranging from medical care to self-driving cars. The progress in computer vision has assisted in the development of novel visual inspection techniques relevant to railways. Rail inspection using visual cameras has emerged as a popular technique with the advantage of high-speed application, low cost and appealing performance [50]. According to Papaelias *et al.* [75], computer vision is regarded as the most attractive technique for rail surface defect detection. The focus of this research is based on anomaly detection using computer vision. Chapter 1.4 is dedicated to a review of computer vision techniques used in railway condition monitoring.

1.3 Rail Surface Defects

Railways work consistently under harsh environments and as part of the rail structure, it has little redundancy. Rail surface defects can cause catastrophic accidents through derailment, which have consequences such as death, injury, cost and loss of public confidence [12]. Surface defects are likely to become more of a concern as industry demands for higher speed, heavier loads and more traffic [12].

Rail surface defects are mainly due to the interaction between the wheel and rail [61]. Most defects take place on the head surface and the gauge corner, refer to Figure 1.1(a). Rolling contact fatigue (RCF) is a broad class of defects which includes the following surface defects: head checks, shelling, squats and crushed heads. Additional surface defects not classified under RCF are: corrugation, wheel slips and wheel burns [4]. Railway experts and papers in the literature use different names for defects and classify different defects under different categories, however, for the application of rail inspection, surface defects are commonly split into two categories, namely, corrugation defects and discrete defects [51]. Corrugation defects appear in a repeatable and periodic manner [64], whereas discrete defects appear in a random or arbitrary manner [51]. Throughout this dissertation, we will use the terms discrete surface defects and surface defects interchangeably. The focus of this research is on the inspection of discrete surface defects. For in-depth and comprehensive reviews regarding surface defects and RCF defects refer to [4, 12, 61].

1.3.1 Challenge of Detecting Discrete Surface Defects with Visual Systems

Discrete surface defects are difficult to detect in computer vision. According to Li and Ren [50], this is due to the following factors:

1. Insufficient features for recognition. Due to the many different types of discrete defects that can appear in countless forms, it is difficult to find common features (such as texture and shape features) for discrete defects.
2. Illumination inequality. Acquired rail images are affected by dynamic environmental conditions.
3. Reflection properties. The surface of rail heads differs in reflection properties, with some rails having smooth surfaces where the centre region of the rail is more reflective than elsewhere. Rails with rust reflect less light and different defects reflect light in different ways.
4. Computation speed. Inspection systems are expected to be able to perform in real-time. This is needed as it is infeasible to store the immense amount of image data recorded over long hauls.

Under ideal conditions, discrete defects can be distinguished from the background using local grey-level information, however, this is often not the case due to the above mentioned issues.

1.3.2 Maintenance for Rail Surface Defects

Almost all surface defects can be maintained through rail grinding if the defects are found early enough before they become too severe. Rail grinding has been used for many years in the rail industry to maintain and increase the life of rails [12]. In the United States on heavy-haul railways, it is not unusual for 50% or more of the total track length to be maintained through rail grinding per annum [12]. Rail grinding is a well-established process, reducing the rate at which defects develop, as well as removing damaged portions of the rail [61].

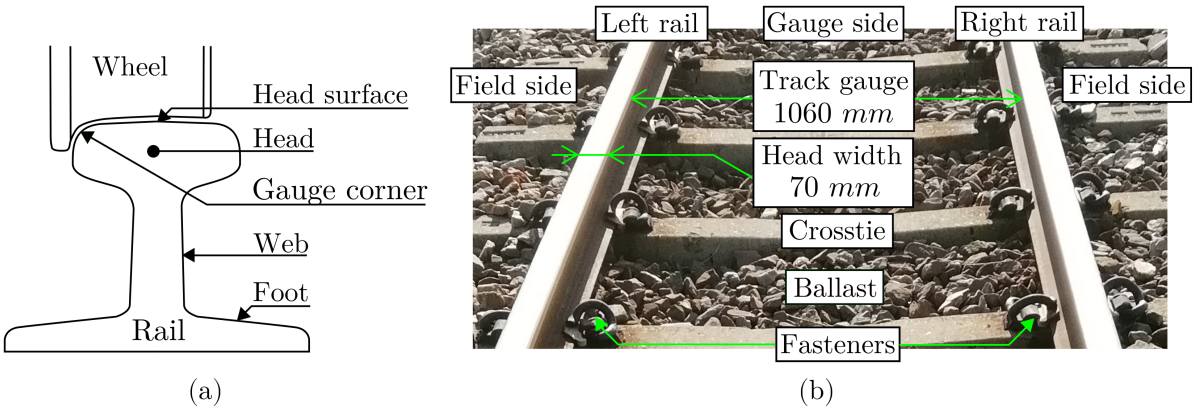


Figure 1.1: Rail details. (a) Labelled rail and wheel section. (b) Labelled track components.

1.4 Computer Vision in Railway Condition Monitoring

Computer vision techniques have been applied to inspect several aspects of the railway infrastructure, such as ballast inspection [81], crosstie inspection [30, 81], inspection of rail fastener elements [25, 29, 30], bolt detection [66], profile measurements [2], track alignment [40], corrugation inspection [52, 64] and discrete surface defects. The application of computer vision for surface defects can be split into two approaches, those which use hand-crafted features and those which use learned features. Methods which use hand-crafted features may be data-driven (machine learning) or not, whereas methods which use learned features are purely data-driven (deep learning, which is part of a broader family of machine learning methods). These methods mainly solve two types of problems, defect classification and/or defect localisation.

1.4.1 Defect Localisation using Hand-Crafted Features

Various methods which use hand-crafted features have been proposed for defect localisation. Li and Ren [50] used local normalization methods to enhance the contrast of rail images to distinguish defects from the background. Defect localisation was then performed based on a projection profile of the mean intensity over each longitudinal or transversal line. In the same year, Li and Ren [51] proposed an alternative approach which used a Michelson-like contrast measure to enhance images locally, to again distinguish defects from the background. In addition, they presented a new automatic thresholding method called proportion emphasized maximum entropy thresholding.

Yu *et al.* [91] presented a coarse-to-fine model (CTFM) to identify defects at three different scales: sub-image, region and pixel. The CTFM model is used to perform filtering, locating and refining on the images before Otsu thresholding is applied. Gan *et al.* [26] also proposed a hierarchical framework with coarse and fine extractors. Their coarse extractor is used to locate outliers based on the mean shift algorithm. The fine extractor determines whether an outlier is a true defect or noise through a saliency function which incorporates background modes and prior information of true defects.

Vijaykumar and Sangamithirai [88] analysed the texture of rails using a Gabor filter bank of various frequencies and orientation to determine an optimal orientation to maximise the energy difference between defects and the background. Jie *et al.* [54] presented a grey-level histogram curve to locate defects on smoothed rail head surfaces also using a statistical texture analysis method based on first-order histogram statistics.

Taştimur *et al.* [87] presented an approach where morphological operations are used as a feature extractor to identify defected regions in an image. Min *et al.* [69] presented a solution where an image is enhanced using an adaptive dual threshold coupled with morphological processes. The characteristics of defects are then obtained using the direction chain code tracking algorithm to extract the contour information of the defect region. Sambo *et al.* [80] utilized adaptive histogram equalisation for local contrast enhancement, followed by an adaptive threshold method to segment the defects. Liu *et al.* [57] developed a spalling defect detection system based on dynamic thresholding.

Gan *et al.* [27] proposed a background-oriented defect inspector which uses a random sampling stage to obtain a representation of the background without prior information. A background model is calculated by means of a fusion strategy to balance the random samples which may contain defects with those which are defect free. The background model is incorporated into a saliency function that labels each pixel as either background or defect. He *et al.* [36] used a new inverse Perona-Malik diffusion model with a distinct nearest-neighbour difference scheme to smooth defects and leave the background unchanged. Defects are then highlighted by comparing the diffused image to the original.

Zhang *et al.* [93] proposed a method to denoise images using a curvature filter. Defects are segmented using an improved Gaussian mixture model based on a Markov random field. Dubey and Jaffery [22] used geometrical features of the defected regions based on maximally stable extremal region technique to segment defects.

1.4.2 Defect Classification using Hand-Crafted Features

Defect classification methods are less common in literature since classification algorithms require large amounts of labelled data.

Wang *et al.* [89] proposed a bilevel super pixel-based framework based on hand-crafted features. The first level applies a simple linear iterative clustering algorithm to generate super pixels from raw images. The second level then uses a bag-of-words feature extractor for each super pixel with DAISY descriptors to develop a classifier. Each super pixel is labelled as either containing a crack or not.

Zhuang *et al.* [97] also proposed a method to classify patches of an image as either containing a crack or not using hand-crafted features. Extended Haar-like features are applied to develop features for identifying sub-regions of the image with cracks. A feature-based linear iterative crack aggregation is used to identify the boundary of cracks.

Hajizadeh *et al.* [34] performed a one class classification approach, in which an image is either classified to contain a squat defect or a normal healthy rail. This approach utilized hand-crafted features such as a Gabor function and the two-dimensional discrete cosine transform.

1.4.3 Defect Classification using Learned Features

Faghih-Roohi *et al.* [24] proposed a learned feature approach with deep convolution neural networks (DCNNs). The networked was trained using a large labelled dataset containing six classes, namely, normal, weld, light squat, moderate squat, severe squat, and joint.

Gibert *et al.* [28] presented a learned feature semantic segmentation method also using DCNNs. This network was trained in a supervised manner, not to detect defects but to label each pixel in an image to belong to one of the ten classes: ballast, wood, rough concrete, medium concrete, smooth concrete, crumbling concrete, chipped concrete, lubricator, rail or fastener.

1.4.4 Discussion of Prior Work

The clear majority of literature uses hand-crafted features, as these techniques have a strong theoretical background in image and signal processing. Hand-crafted features have had success, however, they often fall short in dealing with images captured under dynamic conditions. These approaches are not general enough and only work well under specific conditions.

Deep learning approaches have the ability to perform condition monitoring autonomously, with an end-to-end system that can automatically learn features from raw data. Recently learned features have become popular due to the great advancements in deep learning research and with the development of regularly available parallel computing devices such as graphical processing units (GPUs), which enable algorithms to be computationally efficient. Deep neural networks (DNNs) and DCNNs have been applied in multiple areas of research with great success. The key to this success is based on the large amounts of data which are available to train these algorithms.

The rail industry is in a prime position to apply such methods, with several hundred or even thousands of kilometres of rail track present throughout the world, an immense amount of data can be acquired. This large amount of data could be our best approach to learning the non-linear nature of images captured under dynamic conditions. Labelling this data into multiple different classes, as done in [24], is very time consuming and requires expert knowledge in rail defects. With such a large number of images, often only a portion of the dataset is labelled and used for training algorithms. Another issue is that there is a class imbalance in the dataset, which could create a bias towards healthy images during training. Most captured images will be healthy and very few images will contain defects. In the Netherlands, often less than one percent of inspected rails have some defect [34].

1.5 Proposed Method

In this research we propose a deep learning approach for anomaly detection based on image data captured of a rail's surface. Anomaly detection is the process of identifying data points which differ from the norm. In contrast to diagnostics, anomaly detection is performed without labelled data. We perform reconstruction-based anomaly detection with the goal of segmenting defects in an unsupervised manner. In general machine learning algorithms are considered to be a black box as results are not easily interpretable, however, defect segmentation allows one to interpret the predicted faults within an image.

The class imbalance issue (ratio of healthy to unhealthy data), can be overcome through a novelty detection approach. This is performed by training a deep learning model in a semi-supervised fashion, where we train our model only using healthy data. Anomalies are then detected as deviations from this learned healthy representation. We use deep generative models in order to learn the distribution of healthy data. The key to success in maintaining rails through rail grinding is to identify defects as soon as possible. Once defects become too severe they require rail replacement. For this reason, we use high-dimensional images in order to detect small and recent defects.

Given this description of the proposed method and the brief discussion of deep learning in Chapter 1.4.4, we can see how these approaches can overcome the challenges of detecting discrete surface defects with visual systems, as mentioned in Chapter 1.3.1.

1.5.1 Contributions

The main contributions of this dissertation (listed in the order in which they are presented) are as follows:

1. A new high-resolution rail surface image dataset is presented and will be made publicly available.
2. A new semi-supervised deep generative model based on a novel loss function and training procedure.
3. A novel inspection algorithm based on deep learning which is highly adaptive to dynamic environmental and rail surface conditions. The algorithm can also perform real-time detection at speeds which significantly exceed the world's fastest train.

1.6 Document Overview

In Chapter 2, we discuss the data acquisition set-up and present the rail surface image dataset. Chapter 3 focuses on deep learning where we discuss generative models in detail, indicating their suitability for our application. We also discuss where and how generative models have been used in anomaly detection. In Chapter 4, we propose our novel generative model. We develop the proposed inspection algorithm in Chapter 5. We cover the pre-processing, processing and post-processing techniques used. Experimental investigation are performed in Chapter 6, where we implement the proposed solution and conduct experiments to develop an optimal solution. Lastly, we conclude our investigations and provide recommendations for further research in Chapter 7.

In order to ensure this dissertation is self contained we have included additional information in the Appendix. A full description of deep learning networks has been provided in Appendix D. We describe neural networks, convolutional neural networks, the back-propagation algorithm and popular training algorithms.

Chapter 2

Data Acquisition

Image data of the surface of a rail was needed to develop a data-driven method, however, there are no publicly available datasets. Research institutions have not yet released the data used in published articles. With no datasets freely available, it was decided to design and build an acquisition rig, to capture images of the surface of a rail.

In this chapter, we discuss the experimental rig and present an image dataset which will be made publicly available alongside the journal articles from this work.

2.1 Experimental Rig Design

A car was designed which could be pushed or pulled manually along a railway. The experimental rig can be seen in Figure 2.1. Three area scan cameras (IDT MotionXtra NX8-S2) were synchronised together and used to capture multiple angles of the same portion of the rail surface. Area scan cameras are easy to use and synchronise, unlike line scan cameras which require a wheel encoder. In this research we developed a system which only uses images captured by one of the cameras (centre camera), however, in further research we plan to use the data acquired from the other two cameras. Each camera was fitted with a 1" 25 mm c-mount lens (Kowa).

The camera placement was based on the suggested camera set-up for stereo digital image correlation [86], where the distance between two stereo cameras should be approximately a third of the distance from the cameras to the target object. In our case, the rail surface is our target. The cameras were placed 400 mm above the rail surface ($d = 400$ mm in Figure 2.1(a)). Two LED lights (Veritas Constellation 120E) were positioned on opposite ends of the rail to provide sufficient illumination. Power was supplied to the cameras and lights via a 12V battery and a 12V-220V inverter. A protective box was placed over the cameras and lights to reduce the impact of natural light and to protect the cameras from dust.

2.2 Experiment Set-up and Results

The experimental rig was used to acquire image data on a Transnet rail line situated in Sentrarand, South Africa. The images were captured through a laptop using Motion Studio software, where the cameras were synchronised and triggered with an external trigger. The camera's exposure was set to 250 μs and the lens aperture was fully closed (F16). Using Motion Studio the acquired resolution was set to 1600×640 pixels (unit8 grey-scale images), which was set to sample at 12 fps to ensure slight overlap in consecutive images. A total of 11830 images were captured along a straight portion of railway that was just over 2 km in length. This portion of the rail is not frequently active and for this reason, light rust has formed on the surface.

Two examples of the captured images can be seen in Figure 2.2. The gauge corner is positioned in the top edge of the images. Each image captures approximately 230 *mm* of the rail length. Towards the left and right edge of the images, we can see lens distortion and inconsistent illumination. The light rust on the surface can also be seen in Figure 2.2. This rust adds additional noise to the images, which in return, is expected to make the inspection task more challenging than a smooth rail surface.

Figure 2.1: Experimental rig. (a) Schematic of camera placement. (b) Close-up of the cameras. (c) Experiment rig set-up.

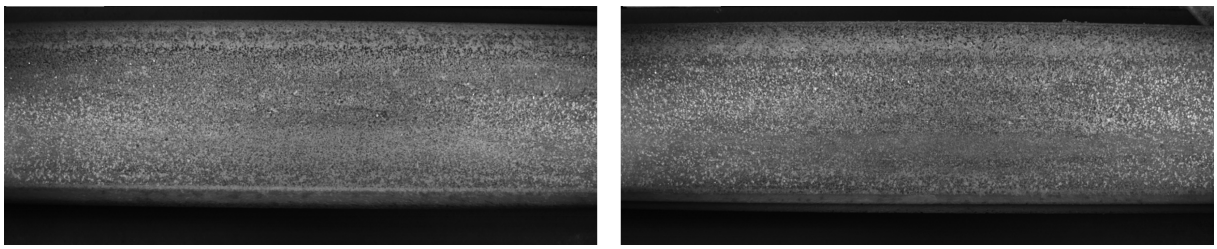
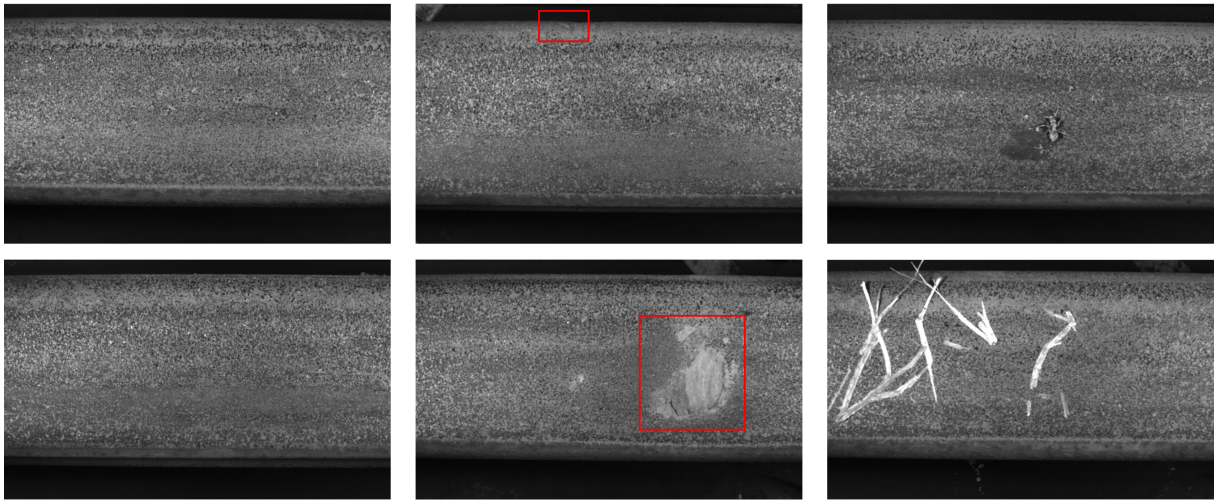


Figure 2.2: Example of captured images.

2.3 Data Preparation

In a true deep learning fashion, minimal preprocessing was applied to the captured images. The only form of pre-processing applied was to crop the images. To compensate for lens distortion and inconsistent illumination, the original length of the images was cropped from 1600 pixels to 1024 pixels, as can be seen in Figure 2.3. This crop would not actually be considered a pre-processing step, as we could have set the desired captured images to be 1024×640 pixels beforehand using Motion Studio software. The cropped images still include other background information in addition to the rail surface. To reduce the inspection area within the image and to avoid redundant operations, we perform a pre-processing step to extract purely the rail surface, which is our region-of-interest (ROI). The specifics of this operation are discussed in detail in Chapter 5.1, but a few ROI images can be seen in Figure 2.4(a). The ROI captures an area of the rail of approximately 150×75 *mm* in a 1024×512 pixels image. Using these dimensions, we have calculated the pixel density of the image to be 300 pixels per inch (*ppi*). Refer to Equation 2.1.

$$ppi = \frac{(\sqrt{width^2 + height^2})_{pixels}}{(\sqrt{width^2 + height^2})_{inches}} \tag{2.1}$$



(a) (b) (c)

Figure 2.3: Cropped example images of the three classes. (a) Healthy. (b) Unhealthy. (c) Artefacts.

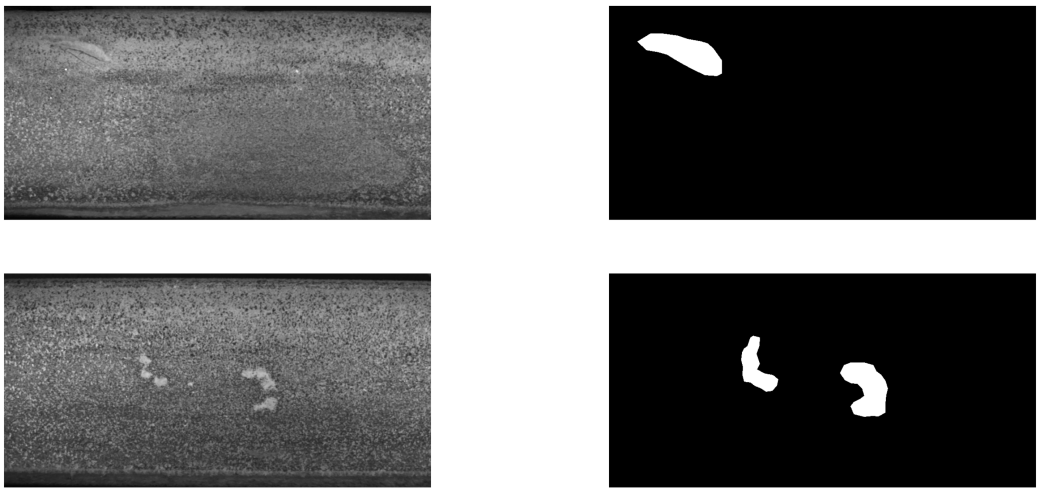
Labelling railway data usually requires expert knowledge on rail defects, however, we feel confident that we can label healthy images sufficiently well, especially if we do so in a conservative manner. All 11830 images were initially split into two categories, either healthy or abnormal. Any images which were perceived not being completely healthy were added to the abnormal category. This strict requirement would ensure no questions about the healthy class, although, we expect some portion of the abnormal dataset to contain images which might be healthy. The abnormal dataset mostly consists of images which contain artefacts such as leaves, twigs, bird droppings, insects, etc., as well as defected portions of the rail. Figure 2.3 depicts examples of healthy images, unhealthy images (defects are shown by red bounding box) and images which contain artefacts.

The healthy and abnormal images were split further into four categories, namely, training images (healthy images), validation images (healthy images), test images (both healthy and defected images) and unused images. The abnormal images were inspected in detail using multiple camera angles (all three cameras) to locate obvious defects which penetrate the rail. These defected images were added to the test dataset and the remainder of abnormal images were unused. The same number of healthy images were chosen at random and added to both the test and validation dataset. Refer to Table 2.1 for a detailed description of the final datasets.

Table 2.1: Division of the captured image dataset.

Healthy	Training images	8570
	Validation images	150
	Healthy test images	150
Abnormal	Unhealthy test images	150
	Unused images	2810
Total number of images		**11830**

Ground truth images which segment out the location of any defects within an image were made. Each of the 150 defected images has a corresponding ground truth image. These ground truth images were made by what we perceived to defective regions. Ideally a domain expert should confirm this. We use these images to ensure that possible faults we find through our algorithm, correspond to a defected region within the image. Examples of ground truth images and their respective ROI defected image can be seen in Figure 2.4.



(a) (b)

Figure 2.4: Region-of-interest image. (a) Defect. (b) Corresponding ground truth of the defects.

Chapter 3

Deep Learning

The core of artificial intelligence tasks (e.g. computer vision, natural language processing, etc.) is to learn rich, hierarchical models that represent probability distributions over observed data. Theoretical results suggest that in order to learn this kind of complicated function, one may need deep architectures [6], which compose of multiple levels of non-linear operations, such as DNNs and/or DCNNs with many hidden layers. Deep learning involves searching the parameter space of these differential deep networks through back-propagation algorithms.

Deep learning techniques have the ability to perform condition monitoring autonomously. In unsupervised and semi-supervised deep learning, a differential network is tasked with learning a distribution which best approximates the true distribution of some observed dataset. The goal of this chapter is to learn the complex distribution of a large amount of healthy high-dimensional images.

Generative models are the ideal solution to model the likelihood of healthy data in an end-to-end deep learning framework. Generative models attempt to match the true data distribution p_{data} of observed data $\boldsymbol{x}$ with a generated distribution p_g. A generative model, once trained on a dataset, has the capabilities of generating new data which resembles that of the training/observed data. The intuition behind generative models follows a famous quote by Richard Feynman, "What I cannot create, I do not understand".

In order for a generative model to approximate p_{data} over the observed data $\boldsymbol{x}$, it is necessary for a differential deep network to capture the relationships and dependencies between different components of the observed dataset. This relationship is defined by an assumption that the data comes from some random process, involving an unobserved continuous random variable $\boldsymbol{z}$, known as a latent variable. Therefore, we imply that the true data distribution $p_{data}(\boldsymbol{x})$ is actually a conditional distribution $p_{data}(\boldsymbol{x}|\boldsymbol{z})$, where $\boldsymbol{z}$ is some vector of random latent variables drawn from an assumed prior distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$.

Generative models have achieved state-of-the-art performance in various unsupervised and semi-supervised learning tasks. The two most popular approaches are Variational Auto-Encoders (VAEs) which explicitly approximate p_{data} and Generative Adversarial Networks (GANs) which implicitly estimate p_{data}. These two approaches are the focal point of this chapter. We discuss both models in detail and include where these models have been used in anomaly detection.

3.1 Variational Auto-Encoders

VAEs were proposed by Kingma and Welling [45], and Rezende *et al.* [77] as a directed statistical model, which consists of two probabilistic networks: an encoder Q and a decoder P. VAEs are a probabilistic spin on Auto-Encoders (AEs) [14]. AEs are commonly used as an unsupervised model for learning a lower-dimensional feature space on training data. This is performed by encoding observed data into a smaller feature space (dimensionality reduction) through a differential encoder network. By having the feature space be smaller than the data space, we force the encoder to capture meaningful factors of variation in the data. The feature space is learned through an additional differential network called a decoder, which is tasked with mapping the feature space back into data space. An AE (both the encoder and decoder network) is trained to learn useful features that can be used to reconstruct the input data. The objective of an AE is a minimal reconstruction loss between the original input data and the reconstruction of this data. Figure 3.1 illustrate the difference between AEs and VAEs models.

The fundamental problem with AEs, for generation, is that their feature space may be discontinuous and sparse which prevents interpolation. Due to the sparseness, a new unseen input may have an encoded feature space distanced far away from any observed data point. The decoder has no idea how to deal with this distanced region in the feature space, therefore the decoder will produce an unrealistic reconstruction. VAEs solve this issue by ensuring the encoded space is continuous and follows some prior distribution which allows one to generate data through the decoder. Note our derivations in this sub-chapter are most similar to the work by Kingma and Welling [45], where the final results are identical the their work.

VAEs attempt to explicitly model the probability or likelihood of observed data. The objective function of a VAE is to maximise the marginal likelihood of observed data, Equation 3.1 [45]. We can assume a prior on $p_\theta(\boldsymbol{z})$ and we can approximate the posterior distribution $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ using a differential decoder network, however, it is intractable to compute the integral of $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ over the entire latent space.

$$p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{z}) p_\theta(\boldsymbol{x}|\boldsymbol{z}) \, d\boldsymbol{z} \tag{3.1}$$

VAEs solve this problem by incorporating knowledge of the posterior density function, Equation 3.2 (Bayes' theorem), where $p_\theta(\boldsymbol{x})$ is intractable. We define a differential encoder network $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ to approximate $p_\theta(\boldsymbol{z}|\boldsymbol{x})$. The probabilistic encoder $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and decoder $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ produce distributions of $\boldsymbol{z}$ and $\boldsymbol{x}$ respectively by sampling from their outputs. The encoder outputs the mean $\mu_{z|x}$ and diagonal covariance $\sum_{z|x}$ of $\boldsymbol{z}$ given $\boldsymbol{x}$, whereas the decoder outputs the mean $\mu_{x|z}$ and diagonal covariance $\sum_{x|z}$ of $\boldsymbol{x}$ given $\boldsymbol{z}$, refer to Figure 3.1(b).

$$p_\theta(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_\theta(\boldsymbol{z}) p_\theta(\boldsymbol{x}|\boldsymbol{z})}{p_\theta(\boldsymbol{x})} \tag{3.2}$$

The marginal likelihood can be approximated by the sum over marginal likelihoods of individual data points (Equation 3.3). Equation 3.4 represents the log-likelihood and the expectation with respect to $\boldsymbol{z}$ samples from $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. We then utilize Equation 3.2 and a constant $\left(\frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right)$ to incorporate the encoder into the likelihood, with logarithm rules to go from Equation 3.4 to 3.7.

$$\log p_\theta(\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(N)}) = \sum_{i=1}^{N} \log p_\theta(\boldsymbol{x}^{(i)}) \tag{3.3}$$

$$\log p_\theta(\boldsymbol{x}^{(i)}) = \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[\log p_\theta(\boldsymbol{x}^{(i)})\right] \tag{3.4}$$

$$\log p_\theta(\boldsymbol{x}^{(i)}) = \mathbb{E}_{\boldsymbol{z}}\left[\log \frac{p_\theta(\boldsymbol{z}) p_\theta(\boldsymbol{x}^{(i)}|\boldsymbol{z})}{p_\theta(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\right] \tag{3.5}$$

$$\log p_\theta(\boldsymbol{x}^{(i)}) = \mathbb{E}_{\boldsymbol{z}}\Big[\log \frac{p_\theta(\boldsymbol{z})p_\theta(\boldsymbol{x}^{(i)}|\boldsymbol{z})}{p_\theta(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\frac{q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})}{q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\Big] \tag{3.6}$$

$$\log p_\theta(\boldsymbol{x}^{(i)}) = \mathbb{E}_{\boldsymbol{z}}\Big[\log p_\theta(\boldsymbol{x}^{(i)}|\boldsymbol{z})\Big] - \mathbb{E}_{\boldsymbol{z}}\Big[\log \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})}{p_\theta(\boldsymbol{z})}\Big] + \mathbb{E}_{\boldsymbol{z}}\Big[\log \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})}{p_\theta(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\Big] \tag{3.7}$$

We can measure the difference between two probability distributions using Kullback-Leibler (KL) divergence to derive Equation 3.8. The issue with this equation is that we know the posterior $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ is intractable, which implies we cannot calculate the second KL term in Equation 3.8. Although we cannot calculate this term, we still know that by definition a KL divergence is always positive. We therefore drop this term and calculate the variational lower bound or evidence lower bound (ELBO), as expressed by Equation 3.9 [45]. We can take gradients and optimize this tractable ELBO.

$$\log p_\theta(\boldsymbol{x}^{(i)}) = \mathbb{E}_{\boldsymbol{z}}\Big[\log p_\theta(\boldsymbol{x}^{(i)}|\boldsymbol{z})\Big] - D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)}) \,||\, p_\theta(\boldsymbol{z})) + D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)}) \,||\, p_\theta(\boldsymbol{z}|\boldsymbol{x}^{(i)})) \tag{3.8}$$

$$\log p_\theta(\boldsymbol{x}^{(i)}) \geq \mathbb{E}_{\boldsymbol{z}}\Big[\log p_\theta(\boldsymbol{x}^{(i)}|\boldsymbol{z})\Big] - D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)}) \,||\, p_\theta(\boldsymbol{z})) \tag{3.9}$$

The new objective of a VAE is to maximise the ELBO. In essence the first term encourages reconstructed samples to be similar to the original input and the second term encourages the encoded latent space (approximate posterior) to follow the assumed prior distribution. We can solve the KL term in closed-form by assuming our prior $p_\theta(\boldsymbol{z})$ to be some distribution, such as a normal (Gaussian) distribution. To solve an approximation of the first term we use a reparameterization trick, to reparameterize the random variable $\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ using a differentiable transformation $\boldsymbol{z} = g_\phi(\boldsymbol{\epsilon}, \boldsymbol{x})$, where $\boldsymbol{\epsilon}$ is an auxiliary variable with independent marginal $p(\boldsymbol{\epsilon})$. We can then calculate a Monte Carlo estimate of expectations of some function $f(\boldsymbol{z})$ with respect to $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, using Equation 3.10

$$\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})}[f(\boldsymbol{z})] = \mathbb{E}_{p(\boldsymbol{\epsilon})}[f(g_\phi(\boldsymbol{\epsilon}, \boldsymbol{x}^{(i)}))] \simeq \frac{1}{L}\sum_{l=1}^{L} f(g_\phi(\boldsymbol{\epsilon}^{(l)}, \boldsymbol{x}^{(i)})) \tag{3.10}$$

where $\boldsymbol{\epsilon}^{(l)} \sim p(\boldsymbol{\epsilon})$ and we choose the distribution of $p(\boldsymbol{\epsilon})$. The reparameterization trick allows this function to be differentiable. Note that from here onwards, unless stated otherwise, we drop (i) from $\boldsymbol{x}^{(i)}$ and (l) from $\boldsymbol{\epsilon}^{(l)}$ by assuming we are only interested in a single input and one Monte Carlo sample ($N = 1,\ L = 1$).

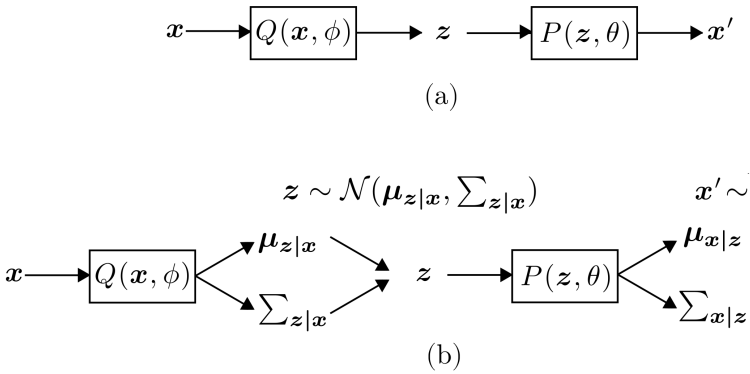


Figure 3.1: Forward pass of (a) Auto-Encoder. (b) Variational Auto-Encoder.

3.1.1 VAE Training

In order to train a VAE we need to specify its set-up. Firstly, we choose the prior over the latent variables to be a multivariate isotropic Gaussian: $p_\theta(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \mathrm{I})$. We then assume the intractable true posterior $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ takes on an approximate multivariate Gaussian with a diagonal covariance (Equation 3.11). Finally, we can also assume $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ follows some distribution such as a multivariate Gaussian with a diagonal covariance (Equation 3.12) for continuous variables, or a multivariate Bernoulli distribution (Equation 3.13, where $\boldsymbol{x}'$ indicates a reconstruction of $\boldsymbol{x}$ and K is the dimensionality of $\boldsymbol{x}$) for binary data. In practice we however, generally implement this mapping to be deterministic, as our loss function does not require $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ to be close to a prior distribution, unlike the approximate posterior $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. We estimate $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ by comparing $\boldsymbol{x}'$ to $\boldsymbol{x}$.

$$q_\phi(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathrm{I}) \tag{3.11}$$

$$p_\theta(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathrm{I}) \tag{3.12}$$

$$\log\, p_\theta(\boldsymbol{x}|\boldsymbol{z}) = \sum_{k=1}^{K} \Big(x_k \,\log\, x'_k + (1 - x_k) \;\; - \log(1 - x'_k) \Big) \tag{3.13}$$

The forward pass of this VAE model (similar to Figure 3.1(a) but the decoder is deterministic), given a random mini-batch of data points $\boldsymbol{x}_b$ is explained by Equation 3.14 to 3.16, where the encoder Q and decoder P are differential functions represented by their respective parameters, ϕ and θ (weights and bias of a network), $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and $\odot$ implies an element-wise product.

$$\boldsymbol{\mu},\;\; \boldsymbol{\sigma}^2 = Q(\boldsymbol{x}_b, \phi) \tag{3.14}$$

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \tag{3.15}$$

$$\boldsymbol{x}'_b = P(\boldsymbol{z}, \theta) \tag{3.16}$$

When training a VAE we maximise a lower bound on the log likelihood of the data (Equation 3.9). Given that both $p_\theta(\boldsymbol{z})$ and $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ are Gaussian, the KL divergence between these distributions is expressed by Equation 3.17, which we can analytically integrate (hence no estimation by sampling) to the form of Equation 3.18, where M is the dimensionality of $\boldsymbol{z}$. The full derivation can be found in Appendix A.

$$-D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \;||\; p_\theta(\boldsymbol{z})) = \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \,\log\, p_\theta(\boldsymbol{z})\; d\boldsymbol{z} \;\; - \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \,\log\, q_\phi(\boldsymbol{z}|\boldsymbol{x})\; d\boldsymbol{z} \tag{3.17}$$

$$-D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \;||\; p_\theta(\boldsymbol{z})) = \frac{1}{2} \sum_{m=1}^{M} (1 + \log\, \sigma_m^2 - \mu_m^2 - \sigma_m^2) \tag{3.18}$$

This KL penalty is differentiable. We can make the reconstruction term of the ELBO to also be differentiable using the reparameterization trick shown in Equation 3.10. In our case the differentiable transformation function is given by Equation 3.15. The derived reconstruction probability is represented by Equation 3.19.

$$\mathbb{E}_{\boldsymbol{z}}\Big[\log\, p_\theta(\boldsymbol{x}|\boldsymbol{z})\Big] = \frac{1}{L} \sum_{l=1}^{L} \log\, p_\theta(\boldsymbol{x}|\boldsymbol{z}^{(l)}) \tag{3.19}$$

With this term we maximise the log-likelihood of the original input being reconstructed. The reconstruction loss ($\log\, p_\theta(\boldsymbol{x}|\boldsymbol{z})$) has many possible forms, but the most common is a mean squared error (MSE) between $\boldsymbol{x}$ and $\boldsymbol{x}'$. Note the MSE is equivalent to the average squared

L_2-norm. In practice and as done in the original implementation [45], we generally only use one Monte Carlo sample during training ($L = 1$), although, once the VAE model has been trained we might use multiple samples for inference queries. Equation 3.20 indicates the MSE for a single Monte Carlo sample. We investigate alternative loss functions in Chapter 4.1.1.

$$\mathbb{E}_{\boldsymbol{z}}\Big[\log\, p_\theta(\boldsymbol{x}|\boldsymbol{z})\Big] = \log\, p_\theta(\boldsymbol{x}|\boldsymbol{z}) = \frac{-1}{K}\sum_{k=1}^{K}(x_k - x'_k)^2 = \frac{-1}{K}||\boldsymbol{x} - \boldsymbol{x}'||_2^2 \tag{3.20}$$

The resulting loss function for this VAE model given N data points is represented by Equation 3.21, which produces a scalar loss that we minimise through optimisation. Note we take the average KL-loss over M, as we averaged the reconstruction error.

$$\min_{\theta,\phi}\ \mathcal{L}_{VAE} = \frac{1}{N}\sum_{i=1}^{N}\Big[\frac{-1}{2M}\sum_{m=1}^{M}(1 + \log[(\sigma_m^{(i)})^2] - (\mu_m^{(i)})^2 - (\sigma_m^{(i)})^2) + \frac{1}{K}||\boldsymbol{x}^{(i)} - \boldsymbol{x}'^{(i)}||_2^2\Big] \tag{3.21}$$

We simultaneously train the encoder and decoder using mini-batch stochastic gradient descent.

3.1.2 VAE Discussion

VAEs are theoretically elegant and easy to train, however, their reconstructed images tend to be blurry and lack details. The continuous latent variable allows for random generation of data through the decoder by sampling from the prior distribution. We can couple this generation process with an encoded latent variable of an input, to produce samples through the decoder that are similar to that input. VAEs can learn independent factors from the training data, it can generalise new combinations of factors it has not seen before, making it ideal for missing data problems [43]. This implies new unseen inputs which may differ from the training data, will still be represented sufficiently well in the latent space, so that high quality reconstruction is still possible. The chosen dimensionality of the latent variables can impact the reconstructed/generated results. We discuss the significance of this dimensionality in Chapter 6.

The learned latent space of a VAE captures insightful information of the observed data. This latent space can be interpreted as a manifold. The manifold hypothesis states that real world data is embedded onto a low dimensional manifold which exists in the high dimensional observed space. The manifold representation for any input variable can be inferred via the encoder $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. This allows manifold learning to take place, where we can identify features of inputs, as well as the difference between features of multiple inputs. Interpretation of this non-linear manifold is challenging, but this may be a favourable approach if our observed data is complicated and incomprehensible, such as non-stationary vibration signals or multiple inputs from heterogeneous observations.

In this research we are interested in analysing a single image which is naturally interpretable to us. Therefore we focus our interest on the reconstruction of an image rather than its latent representation. The manifold is still very useful, as we can encode a single input to a mean and variance of the latent space and then take multiple Monte Carlo samples to produce multiple reconstruction images (Equation 3.19, with $L > 1$). This allows a single input to be compared to several reconstructions of that input. VAEs produce blurry samples due to the chosen reconstruction loss (Equation 3.19). As previously mentioned there are numerous loss functions to choose from. We further analyse the implications of this loss function in Chapter 4.1.1.

3.1.3 VAEs in Anomaly Detection

VAEs are commonly applied to anomaly detection using two approaches, namely, reconstruction error and/or manifold learning in the latent space. As we are interested in images, we focus on the reconstruction error. According to Khan and Yairi [43], VAEs are ideal for health management applications for anomaly detection and failure prediction problems. Faults can be detected in an unsupervised manner under the assumption that most of the observations are normal and deviations from this could indicate anomalies.

An and Cho [3] proposed an anomaly detection algorithm based on the reconstruction probability of a VAE (Equation 3.19). During testing, a number of samples ($L > 1$) are drawn from a probabilistic encoder and for each sample a probabilistic decoder (Equation 3.12) outputs the mean and variance parameter. Using these parameters, the probability of the data being generated from a given latent variable drawn from the approximate posterior distribution is calculated. The average probability is used as an anomaly score and is called the reconstruction probability. Note that the anomaly score is not based on comparing the original data to the reconstructed data. Presumably anomalous data will have greater variance and show lower reconstruction probability. Anomalies are detected when the reconstruction probability is below a chosen threshold. This approach had success in simple low dimensional problems such as the MNIST dataset [49] of handwritten digits, but higher-resolution images are more troublesome.

Bergmann *et al.* [8] employed both VAEs and AEs applied for anomaly detection on images of nanofibre materials. They attempt to segment defective regions (128×128 pixels) after having trained on exclusively non-defective samples. Their anomaly score is based on a reconstruction error where they compare the input and reconstructed images using the structural similarity (SSIM) metric [90], a distance measure designed to capture perceptual similarity. The SSIM was shown to be better than the MSE loss, although, the SSIM metric is computationally expensive, due the statistics which need to be calculated. This approach is also not well suited for real-time inference.

Matsubara *et al.* [68] proposed a new anomaly score for the reconstruction probability of a VAE. Similar to [3] they also used a probabilistic decoder, however, they only sample the latent variable once ($L = 1$). The proposed anomaly score is similar to Mahalanobis distance or normalized Euclidean distance. A VAE was trained only using healthy image patches (96×96 pixels) of manufactured screw holes and gears. This anomaly score was shown to outperform traditional VAE scores, however, their reconstructed images are blurry (worse than [3]).

3.2 Generative Adversarial Networks

Goodfellow *et al.* [33] proposed GANs as an unsupervised deep learning algorithm that consists of two networks competing against one another. One network denoted the generator G has the objective of producing samples which resemble that of the observed data. The generator is pitted against an adversary denoted as the discriminator D that learns to distinguish real samples from generated samples. The generator can be thought of as a team of counterfeiters who attempt to produce fake currency that will be perceived as real currency, while the discriminator can be thought of like the police who try to detect counterfeited currency.

A GAN does not explicitly model a density function for the observed data, as the goal is to only sample data. GANs take a game-theory approach to learn to generate samples through a two-player game. Figure 3.2 describes the structure of a GAN.

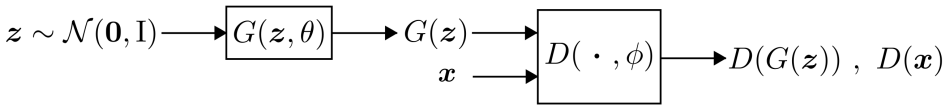


Figure 3.2: Forward pass of Generative Adversarial Network

3.2.1 GAN Training

The training framework corresponds to a minimax two-player non-cooperative game where each player has trainable parameters. The goal of the game is for each player to simultaneously adjust its parameters to minimise its own loss function. The competitive nature of this game drives both players to become progressively better with the end goal of reaching Nash equilibrium [33]. Nash equilibrium is defined as a game state in a non-cooperative game where it is impossible for any player to improve its score by only changing its strategy [73]. In the context of a GAN, Nash equilibrium is achieved once the generator matches the true distribution ($p_g = p_{data}$) so that the discriminator is unable to distinguish between real and generated samples.

To learn the generator's distribution p_g over training data $\boldsymbol{x}$, we define a prior on the input noise variable $p_{\boldsymbol{z}}(\boldsymbol{z})$ and then represent a mapping to data space as $G(\boldsymbol{z}, \theta)$. The noise $\boldsymbol{z}$, is a vector sampled from a latent variable which generally takes the form of a normal or uniform distribution. The discriminator $D(\cdot\,, \phi)$, takes either real or generated samples as an input and outputs a scalar estimate of the probability that $\boldsymbol{x}$ or $G(\boldsymbol{z})$ came from p_{data} rather than p_g. Both G and D are differential functions represented by their respective parameters, θ and ϕ. We train D to maximize the probability of assigning the correct labels to both training and generated samples. We simultaneously train G to minimize the probability that D assigns the correct labels for generated samples. In essence, D strives for $D(\boldsymbol{x}) \approx 1$ and $D(G(\boldsymbol{z})) \approx 0$, whereas G ideally wants $D(G(\boldsymbol{z})) \approx 1$, and Nash equilibrium would result in $D(\boldsymbol{x}) = \frac{1}{2} = D(G(\boldsymbol{z}))$. This minimax optimization problem follows the value function of Equation 3.22.

$$\min_{\theta} \max_{\phi}\ V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{3.22}$$

When D is optimal, this loss function is approximately equal to the Jensen-Shannon divergence (JSD). The loss functions for the generator and discriminator are expressed by Equation 3.23 and Equation 3.24 respectively, given a mini-batch of N data points.

$$\min_{\theta}\ \mathcal{L}_G = \frac{1}{N} \sum_{j=1}^{N} \log(1 - D(G(\boldsymbol{z}^{(j)}))) \tag{3.23}$$

$$\min_{\phi}\ \mathcal{L}_D = \frac{-1}{N} \Big[\sum_{i=1}^{N} \log\ D(\boldsymbol{x}^{(i)}) + \sum_{j=1}^{N} \log\ (1 - D(G(\boldsymbol{z}^{(j)}))) \Big] \tag{3.24}$$

Each model is trained using mini-batch stochastic gradient descent. Note that in the literature many variations of loss functions exist, however, the original objective of matching p_g with p_{data} using an adversarial loss, remains unchanged.

3.2.2 GAN Discussion

GANs have the ability to generate detailed, sharp and realistic looking images, however, as they are trained to only generate samples we cannot solve inference queries, such as $p(\boldsymbol{z}|\boldsymbol{x})$. GANs are also known to be notoriously difficult to train, due to dynamic training and sensitivity in nearly every aspect of its set-up. In fact, GANs were hardly used in research until, Radford *et*

al. [76] introduced deep convolution generative adversarial networks (DCGANs) with certain stabilising guidelines. The presented DCGAN model was capable of generating realistic looking natural images at a resolution of 64×64 pixels.

It did not take long for GANs to be successfully applied to high-resolution image datasets. Some of the most impressive results are those by Karras *et al.* [41], who presented ProGAN and Brock *et al.* [11], who presented BigGANs. ProGAN was trained on a higher quality version of the CelebA dataset [56] to generate images at a resolution of 1024×1024 pixels and BigGANs was trained on ImageNet [19], a more diverse dataset, to produce images at 512×512 pixels. As a showcase for GANs, Appendix B.1 contains generated images reproduced from ProGAN and BigGAN with the authors' permission. These results show that we can learn the distribution of complex datasets and that GANs can generate high-resolution samples with excellent details in images.

Although great strides have been made with GAN models learned directly from devise high-resolution images, there are still several issues. GAN training commonly fails due to a phenomenon known as mode collapse. Generative models aim to match the true distribution of real world data which is inherently multi-modal. Mode collapse takes place when the generator converges to just a few of the modes representing a dataset [79]. Once mode collapse has taken place the generator no longer produces diverse samples, but rather the same sample is repeatedly produced regardless of the value of the random latent variable $\boldsymbol{z}$. Another issue with GAN's is that the latent space is underspecified [13]. The latent space is highly entangled and not easily interpretable due to the lack of structure in this space.

3.2.3 GANs in Anomaly Detection

GANs have been applied for anomaly detection in images. Most of the research attempts to learn an inverse mapping of the generator. That being a mapping from image space to latent space. The idea is to train a GAN only on healthy images, so that we can perform inference (map an image to a latent space and then map back into image space using the generator) to get a healthy reconstruction of an input image. The reconstruction and original image are then compared to one another to point out anomalies. The issue with this approach is that inverting the generator is not a simple operation and for this reason, it is still a very active area of research.

Schlegl *et al.* [82] presented a GAN model for the task of anomaly detection in medical images. Their proposed model called AnoGAN, was trained only on healthy image patches (64×64 pixels), using DCGAN guidelines. Through an iterative optimisation scheme they search for a latent variable which produces a generated image most similar to the image in question. This method is not well suited for real-time inference, due to the computationally expensive search in the latent space, given a new query image. Other efficient inference models attempt to train an encoder E to learn the mapping from image space to the GANs latent space. The encoder is learned either simultaneously whilst training a GAN or on an already trained GAN.

Zenati *et al.* [92] developed an anomaly detection method to simultaneously learn an encoder that maps input samples to a latent representation, along with a generator and discriminator during training. This method is based on bidirectional GANs presented by Donahue *et al.* [20] and Dumoulin *et al.* [23]. This efficient GAN-based anomaly detection method had success in simple problems such as the MNIST dataset, however, bidirectional GANs are troublesome to train especially on more complex datasets.

The generator from a trained GAN can be inverted using a reconstruction error either in the image or latent space. Image-based reconstruction (comparing $\boldsymbol{x}$ to $G(E(\boldsymbol{x}))$) use real images,

whereas latent space reconstruction (comparing $\boldsymbol{z}$ to $E(G(\boldsymbol{z}))$) is performed using generated images. Both approaches have been proposed under multiple variations, such as the work by Luo *et al.* [59] and Creswell *et al.* [15], but with minimal success due to the lack of structure in the latent space which might not even hold all the modes of the data distribution.

Chapter 4

Proposed Approach

The two most common generative models, VAEs and GANs have drawbacks that limit their applications. VAEs are easy to train and have nice latent representations but produce very blurry images that lack detail, whereas GANs are difficult to train and have poor latent representations but produce much sharper, detailed and realistic images. VAEs learn a bidirectional mapping between the data and latent space, allowing for both generation and inference. On the contrary, GANs learn a unidirectional mapping that only generates samples.

We propose a generative model which incorporates the advantages of VAEs and GANs into a single model. By doing so we mitigate the disadvantages of VAEs and GANs. We use a VAE model with a novel loss function based on a discriminator loss, and other improvements in VAEs and GANs. The objective is to have a bidirectional deep learning model that is capable of producing high quality images.

In this chapter we discuss techniques to improve both VAEs and GANs. We then propose a generative model which combines VAEs and GANs in a novel approach. We discuss the architecture of our proposed network, along with a training procedure. We finally train the model and discuss observations from the results.

4.1 Improving VAE and GAN Models

VAEs and GANs complement one another and for this reason, research has gone into combining the two networks. In this sub-chapter we discuss methods to improve VAEs and GANs, we also mention approaches which combine these two generative models.

4.1.1 Reconstruction Error

GANs have been trained to generate images with resolutions much greater than that of VAEs. The issue with VAEs is that the reconstruction error is calculated element-wise. Similarity metrics like the MSE directly compare pixel values. Element-wise metrics do not represent the properties of human visual perception, for example a small image translation might result in a large pixel-wise error that a human would barely notice. GANs avoid element-wise similarity measures by construction of an adversarial loss function.

In deep learning, the most commonly used reconstruction loss is the MSE which produces blurry/smooth images with a high peak signal to noise ratio (PSNR). These images often lack high frequency details [16]. Recently, deep perceptual similarity metrics have become popular. Dosovitskiy and Brox [21] proposed that instead of computing errors in the image space, we rather compute errors between image features extracted by DCNNs. The proposed loss is based

on feeding both the original and reconstructed images into a pre-trained model and then using the outputs of layers to compare the images. Pre-trained deep networks such as AlexNet [46] and VGG [84] are commonly used. These networks have achieved state-of-the-art results in the ImageNet Challenge and are freely available. Pre-trained DCNNs provide feature representations that are invariant to small translations and capture sharp textual details, although, it is known that optimising purely in the feature space leads to high frequency artefacts [63].

Adversarial loss functions based on GANs have the effect of matching the distribution of reconstructed images to that of the training data [83]. This loss is capable of producing realistic, high frequency details in images. Evidently, perceptual and adversarial losses complement one another and are consequently used together, however, optimisation on this combined loss is known to be troublesome as adversarial training is unstable and sensitive to parameters. Resulting images typically also have a low PSNR. Stable training and a higher PSNR is possible by additionally considering the MSE loss in the image space [21]. Thus good results can be obtained by combining the MSE loss with the deep perceptual loss and an adversarial loss. Note from here onwards we use the terms deep perceptual loss and perceptual loss interchangeably.

4.1.2 VAE-GAN Models

VAEs and GANs have been combined together to improve the reconstructed images of VAEs. As briefly discussed in Chapter 4.1.1, Dosovitskiy and Brox combined multiple reconstruction errors to improve the quality of images. In their work a VAE is pitted against a discriminator in a minimax optimisation problem similar to that of a GAN. The discriminator is tasked with distinguishing between real and reconstructed images, whereas the VAE is trained to fool the discriminator into classifying reconstructed images as real. The typical VAE loss (Equation 3.21) is complimented with this adversarial loss and a perceptual loss using pre-trained AlexNet. This model was successfully trained on ImageNet at a resolution of 256×256 pixels. The produced reconstructed images were better than a usual VAE, but their results still fall short of the standards of GANs.

Larsen *et al.* [47] presented a similar solution, but intermediate layers from the discriminator were used to calculate the perceptual loss, rather than a pre-trained model. In addition they require the decoder to not only fool the discriminator with reconstructed samples but also purely generated samples. This model called VAE/GANs was trained on a 64×64 pixels version of the CelebA dataset. Reconstructed images are similar in quality to GANs, however, training this model is troublesome due to the additional requirement for generation. This approach has not yet been applied to higher-resolution images.

4.1.3 Relativistic Discriminator

In a standard GAN the generator is trained to increase the probability that generated data is real. Jolicoeur-Martineau [39] argues that the generator should also simultaneously decrease the probability that real data is real. In order to reach Nash-equilibrium, we require an optimal discriminator, $D(\boldsymbol{x}) = \frac{1}{2} = D(G(\boldsymbol{z}))$. When directly optimising the GAN loss function (Equation 3.22) we would expect $D(\boldsymbol{x})$ to smoothly decrease from 1 to 0.5 and $D(G(\boldsymbol{z}))$ to smoothly increase from 0 to 0.5, however, when minimizing the loss we are only increasing $D(G(\boldsymbol{z}))$, we are not decreasing $D(\boldsymbol{x})$. Furthermore, we are bringing $D(G(\boldsymbol{z}))$ closer to 1 rather than 0.5.

To bring GANs closer to divergence minimization, the generator should be trained to not only increase $D(G(\boldsymbol{z}))$ but also decrease $D(\boldsymbol{x})$. This is achieved using a relativistic discriminator which estimates the probability that real data is more realistic than randomly sampled fake data. In essence, the summation of N data points expressed by the loss function for the dis-

criminator in Equation 3.24, is only done after we directly compare real and generated samples. The generator loss function is also changed to directly compare real and generated samples. This approach is referred to as relativistic GANs (RGANs). Empirical evidence suggests that RGANs are more stable and produce data of higher quality than standard GANs [39]. RGANs are also shown to be more effective when combined with least-squares GANs (LSGANs) [65], this combination is referred to as RLSGANs.

4.2 Proposed Model

We propose a combination of VAEs and RLSGANs inspired by the loss functions and models discussed in Chapter 4.1.1 and 4.1.2. Our approach is most similar to VAE/GANs except the decoder is not tasked with fooling the discriminator with purely generated samples but only reconstructed samples. The discriminator's intermediate layers are used to calculate the perceptual loss. We also incorporate RGANs, so that the adversarial loss for the VAE compares reconstructed images to the original images, with the goal of reconstructed images being more realistic than the original images. The discriminator's objective remains the same as a usual GAN, where randomly sampled real data is compared to randomly sampled fake data. We use a least-squares adversarial loss.

We expect better results than [47], as our model is purely bidirectional, and we also incorporate RLSGANs loss functions which improves training stability and results in higher quality images. Additionally, we use the state-of-the-art network architectures for deep generative models, semantic segmentation models, and encoding-decoding models (Chapter 4.3). We also train our network in a progressive growing manger, which further improves the training and the image results (Chapter 4.4).

To simplify further discussions of our proposed model, we refer to it as NET. This is not an acronym, but simply a name to allow for easy reference of our model. This name should not be used outside the context of this dissertation.

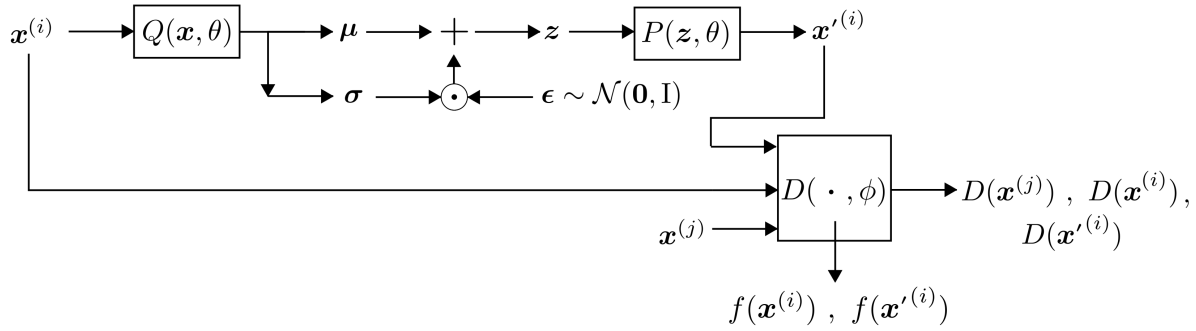


Figure 4.1: Forward pass of proposed model (NET).

4.2.1 NET Training

The proposed model is described by Figure 4.1. Given two mini-batches of N data points, $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$, the least-squares loss function for the discriminator $D(\cdot\,, \phi)$ is expressed by Equation 4.1.

$$\min_{\phi} \; \mathcal{L}_D = \frac{1}{N}\Big[\sum_{i=1}^{N} D(\boldsymbol{x}'^{(i)})^2 + \sum_{j=1}^{N}(D(\boldsymbol{x}^{(j)}) - 1)^2\Big] \tag{4.1}$$

In a VAE both the encoder and decoder strive for the same objective (with the exception that the decoder has no impact on the KL-loss), therefore we represent both networks by the same parameter θ. The multi-objective VAE($\boldsymbol{x}, \theta$) loss function is expressed by Equation 4.2. The KL, MSE, perceptual and relative adversarial loss functions are shown by Equations 4.3, 4.4, 4.5 and 4.6 respectively.

$$\min_{\theta} \mathcal{L}_{VAE} = \frac{1}{N} \sum_{i=1}^{N} \Big[\mathcal{L}_{KL} + \beta \mathcal{L}_{MSE} + \lambda \mathcal{L}_{P} + \mathcal{L}_{RA} \Big] \tag{4.2}$$

with,

$$\mathcal{L}_{KL} = \frac{-1}{2M} \sum_{m=1}^{M} (1 + \log[(\sigma_m^{(i)})^2] - (\mu_m^{(i)})^2 - (\sigma_m^{(i)})^2) \tag{4.3}$$

$$\mathcal{L}_{MSE} = \frac{1}{K} ||\boldsymbol{x}^{(i)} - \boldsymbol{x}'^{(i)}||_2^2 \tag{4.4}$$

$$\mathcal{L}_{P} = \frac{1}{F} ||f(\boldsymbol{x}^{(i)}) - f(\boldsymbol{x}'^{(i)})||_2^2 \tag{4.5}$$

$$\mathcal{L}_{RA} = D(\boldsymbol{x}^{(i)})^2 + (D(\boldsymbol{x}'^{(i)}) - 1)^2 \tag{4.6}$$

where f indicates the hidden representation of layers in the discriminator and F is the dimensionality of these layers. The reconstruction losses (MSE and perceptual) are weighted using β and λ. We simultaneously train the VAE and discriminator using mini-batch stochastic gradient descent.

4.3 Architecture Development

Deep networks can be formulated in an infinite amount of variations, however, much research has been conducted to suggest guidelines on network architectures for generative models and networks used in computer vision. We follow many of the guidelines from recent research.

In this sub-chapter we discuss the relevant literature to motivate our chosen architecture. We then give details of the architecture for our NET.

4.3.1 Architecture Motivation

The implementation of generative models nowadays is predominately based on the DCGAN implementation. DCGANs follow the trend towards eliminating fully-connected layers, by mainly utilizing convolutional layers. Examples of such architectures, is the research performed by Long *et al.* [58], who proposed fully convolutional networks, trained end-to-end, pixels-to-pixels for semantic segmentation. Fully convolutional networks are currently the state-of-the-art deep learning method for segmentation.

In CNNs pooling layers are commonly used to down-sample the spatial dimension in consecutive layers. Springenberg *et al.* [85] proposed an all convolutional network, which replaces deterministic spatial pooling functions with strided convolutions (for an image, a stride of $[2, 2]$), allowing the network to learn its own spatial down-sampling. Strided convolutions have been implemented without loss in accuracy on several image recognition benchmarks. DCGAN utilized strided convolutions for down-sampling and fractionally-strided convolutions (also known as transposed convolutions, but in some recent papers, these are wrongly called deconvolutions) for up-sampling.

Generative models are known to suffer from vanishing and exploding gradients, especially deep networks. Two activation functions are commonly used to circumvent these issues, namely, rectified linear unit (ReLU) proposed by Nair and Hinton [72], and leaky rectified linear unit (LReLU) proposed by Maas *et al.* [60]. DCGAN utilized both of these activation functions with the exception of the output layer of the generator which uses a tanh function, and the output layer of the discriminator which uses a sigmoid function.

CNNs are invariant to translation, for example in an image classification task between dogs and cats, a CNN will make accurate predictions regardless of the location of the animal in the image. Liu *et al.* [55] demonstrated that CNNs perform poorly in simple coordinate transform problems, such as predicting the Cartesian location of a single white pixel in a black image. The authors fix this problem with a solution called CoordConv. This approach is implemented by giving convolution layers access to its own input coordinates through the use of extra coordinate channels [55]. A typical convolution layer can be turned into a CoordConv layer by concatenating hard-coded coordinates into the channels of the convolution layer (refer to Appendix B.2 for an illustrated example). These layers were shown to improve the latent representation of GANs and reduce mode collapse during training. Such an implementation comes without sacrificing the computational efficiency of ordinary convolutions. CoordConv allows networks to learn either perfect translation invariance or varying degrees of translation dependence. The network will decide whether the additional coordinates provide useful features or not.

4.3.2 Model Architecture

The proposed NET consists of three networks, namely, an encoder, a decoder and a discriminator. We designed the architecture for these three networks by considering the discussions from Chapter 4.3.1. A detailed description of the VAE (encoder and decoder) network and the discriminator network can be seen in Table 4.1 and Table 4.2 respectively.

All convolution layers utilize CoordConv, with a kernel size of 3×3 (same padding). CoordConv layers provide an encoder-decoder model with valuable spacial information to reconstruct samples. The encoder and discriminator networks down-samples with strided convolutions, whereas the decoder network up-samples with fractionally-strided convolutions.

The VAE network consists of only convolutional layers. Fully convolutional networks are highly parallel and computational efficiency on GPUs. Such an architecture is ideal for real-time image segmentation. Note the discriminator network has an additional fully-connected layer between the last convolutional layer and the output. In Table 4.2 we have also labelled the output of each layer which we will refer to for the perceptual loss.

We sample $\boldsymbol{z}$ with $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. In Chapter 6 we perform experiments to determine the dimensionality of the channels M_c in the VAEs latent space. Note $Mc = M/8$.

4.4 Training Procedure

We train the network described in Chapter 4.3.2 with the goal of using the VAE to reproduce high-resolution images of the rail surface. We simultaneously optimise the loss functions for the discriminator and VAE expressed by Equation 4.1 and Equation 4.2 respectively, using mini-batch stochastic gradient descent. Training such a network on high-resolution images is difficult because higher resolution images make it easier for the discriminator to tell the generated images apart from training images [74]. High-resolution images also consume memory in a computer, which constrains training to take place using small batches and therefore greatly increasing the

time needed to train a model. Small batch sizes are also known to cause training instabilities.

In this sub-chapter we describe an approach to train a network on high-resolution images and explain our training set-up. We implement the training procedure and discuss general observations of the results.

Table 4.1: VAE architecture for images at 1024×512 resolution.

Encoder θ	Activation Function	Output Shape w × h × c
Input: $\boldsymbol{x}$	-	$1024 \times 512 \times 1$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	LReLU	$512 \times 256 \times 8$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	LReLU	$256 \times 128 \times 16$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	LReLU	$128 \times 64 \times 32$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	LReLU	$64 \times 32 \times 64$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	LReLU	$32 \times 16 \times 128$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	LReLU	$16 \times 8 \times 256$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	LReLU	$8 \times 4 \times 512$
Output: $\boldsymbol{\mu}$, $\boldsymbol{\sigma}^2$		
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv,	linear,	$4 \times 2 \times M_c$,
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	linear	$4 \times 2 \times M_c$

Decoder θ	Activation Function	Output Shape w × h × c
Input: $\boldsymbol{z}$	-	$4 \times 2 \times M_c$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv^T	LReLU	$8 \times 4 \times 512$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv^T	LReLU	$16 \times 8 \times 256$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv^T	LReLU	$32 \times 16 \times 128$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv^T	LReLU	$64 \times 32 \times 64$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv^T	LReLU	$128 \times 64 \times 32$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv^T	LReLU	$256 \times 128 \times 16$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv^T	LReLU	$512 \times 256 \times 8$
Output: $\boldsymbol{x}'$		
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv^T	tanh	$1024 \times 512 \times 1$

Table 4.2: Discriminator architecture for images at 1024×512 resolution.

Discriminator ϕ	Activation Function	Output Shape w × h × c	f-layer
Input: $\boldsymbol{x}$ / $\boldsymbol{x}'$	-	$1024 \times 512 \times 1$	-
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	ReLU	$512 \times 256 \times 4$	f_8
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	ReLU	$256 \times 128 \times 8$	f_7
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	ReLU	$128 \times 64 \times 16$	f_6
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	ReLU	$64 \times 32 \times 32$	f_5
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	ReLU	$32 \times 16 \times 64$	f_4
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	ReLU	$16 \times 8 \times 128$	f_3
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	ReLU	$8 \times 4 \times 256$	f_2
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	ReLU	$4 \times 2 \times 512$	f_1
Fully-connected	sigmoid	$1 \times 1 \times 256$	f_0
Output: $D(\boldsymbol{x})$ / $D(\boldsymbol{x}')$			
Fully-connected	sigmoid	$1 \times 1 \times 1$	-

4.4.1 Progressively Growing Networks

As discussed in Chapter 3.2.2, Karras *et al.* [41] trained a GAN to produce images at a resolution of 1024×1024 pixels. Training at this resolution was made possible through their proposed training methodology, to progressively grow a network. The idea is to initially train a small network using low-resolution images and then to progressively add new layers to your network

and increase the image resolution until you have grown your network to the desired resolution. The network initially learns large-scale details from the image distribution, but then as we progressively grow, the network learns finer scale detail, instead of having to learn all details at once [41]. This approach both speeds up and stabilises training, as we can use a much larger batch size when training with low-resolution images.

To implement this approach, we start off with low-resolution images and randomly initialise all parameters in the network. We train at this resolution for a number amount of steps and then stop. We then add the new layers to the network, and increase the resolution of the training images. We randomly initialise the parameters of the new layers, but we do not do this with the old layers. We instead initialise the old layers with their already pre-trained parameters from the previous resolution. We then train at this higher resolution and continue the process until we reach the desired resolution. With each step that we grow, we smoothly fade the new layers into the network. Figure 4.2 illustrates this procedure for our NET as we grow the resolution from $128 \times 64 \rightarrow 256 \times 128$. We transition in the new layer in a specified number of steps where α is increased linearly from 0 to 1. We double $\times 2$ and half $\times 0.5$ the previous output and input using nearest neighbour filtering and average pooling, respectively.

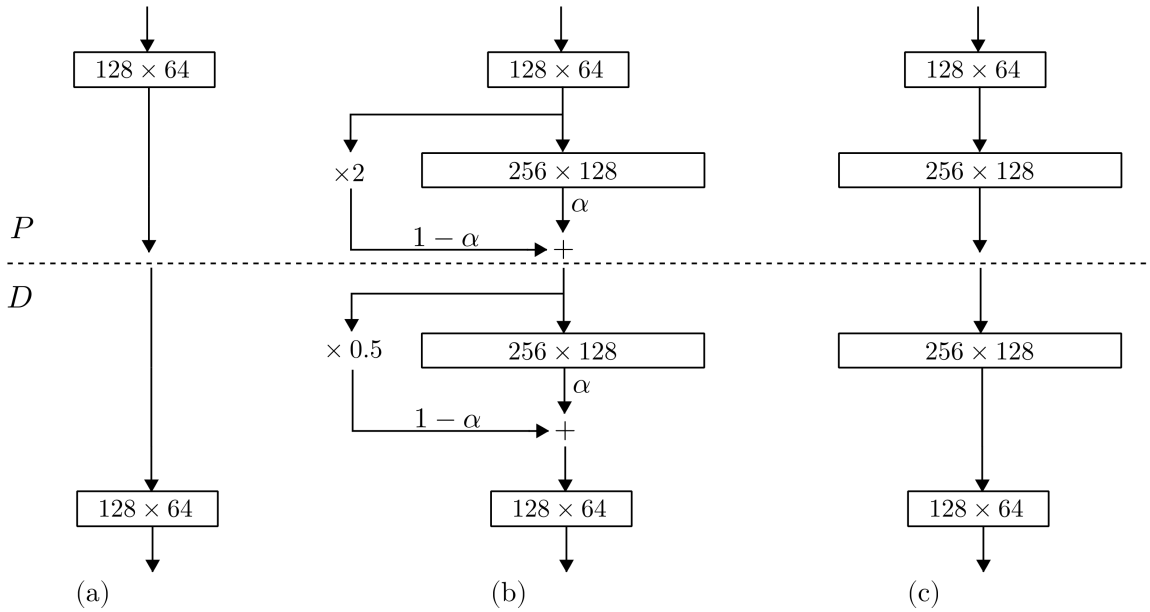


Figure 4.2: Progressively growing the decoder P and discriminator D. (a) Partial NET initial: 128×64. (b) Transition in new layers. (c) Partial NET: 256×128. Note this figure does not describe the full network but only the last (P) and first (D) two layers.

4.4.2 Training Set-up and Observations

Our details of training are again based on the DCGAN implementation. We used the Adam optimiser which was presented by Kingma and Ba [44], with the following tuned hyperparameters from the original algorithm: learning rate = 0.0001 and the momentum term $\beta_1 = 0.5$. The learning rate was set to decay exponentially every 10,000 steps with base 0.95. All layers in the network were initialised using Xavier initialisation. This initialiser was proposed by Glorot and Bengio [31] and is designed to keep the scale of the gradients roughly the same in all layers.

We progressively grew our network in four stages: $128 \times 64 \rightarrow 256 \times 128 \rightarrow 512 \times 256 \rightarrow 1024 \times 512$ pixels. Note we only progressively grew the decoder and discriminator as we found this to be

more stable, due to consistency in the encoded latent space.

For the perceptual loss $\mathcal{L}_P$ we include all available convolution layers from the discriminator. With each step we grow the network we add the new layer: $f_1 : f_5 \rightarrow f_1 : f_6 \rightarrow f_1 : f_7 \rightarrow f_1 : f_8$ refer to Table 4.2. For the MSE loss $\mathcal{L}_{MSE}$ we used $\beta = 10$ and $\lambda = 100$ for the perceptual loss. The values of these weights impact the reconstructed results. If β is too large, the reconstructed images are unduly smoothed (typical VAE issue), but if β is too small, training is less stable and the reconstructed images contain high frequency artefacts. Values chosen for λ have the opposite effect to β. We also noticed that the larger the value of λ, the more likely defects will appear in the reconstructed images. In a GAN stable training takes place when the adversarial loss is near Nash-equilibrium ($\mathcal{L}_D \approx 0.5$), we noticed that our model is also stable at this point ($\mathcal{L}_D \approx 0.5 \approx \mathcal{L}_{RA}$). We determined that one should choose β and λ values so that their loss values are near 0.5 ($\beta\mathcal{L}_{MSE} \approx 0.5$ and $\lambda\mathcal{L}_P \approx 0.5$).

Recent analysis has shown that there is a direct link between how fast models learn and their generalisation performance [35]. For this reason, we trained our model with as few steps as possible. When growing the network, we transitioned in each new layer for 1500 steps. From Figure 4.2, α increases linearly from 0 to 1 in 1500 steps. Table 4.3 describes the duration of training at each resolution. Note the training steps shown include the transition steps.

The model was implemented in Tensorflow [1] and trained on a single Nvidia GeForce GTX Titan X GPU. Training takes less than a day.

Table 4.3: Training duration.

Resolution	Batch size	Training Steps	Epoch
128×64	128	10000	150
256×128	64	10000	75
512×256	32	10000	37.5
1024×512	32	10000	37.5
Total	-	**40000**	**300**

Chapter 5

Inference Algorithm

With our trained NET, we only use the VAE which has learned the distribution of healthy rail images. We use this VAE model to develop a real-time inference algorithm to segment surface defects located in high-resolution images.

In this chapter, we propose and describe an inference algorithm that is used to make predictions given new unseen query images $\boldsymbol{x}$. Firstly, given an image we extract the ROI in a pre-processing step. We then process the ROI image using our trained VAE model to reconstruct ($\boldsymbol{x'}$) a healthy representation of the query images $\boldsymbol{x}$. Finally, we compare $\boldsymbol{x}$ to $\boldsymbol{x'}$, in order to segment defects in a post-processing step (unsupervised).

5.1 Pre-processing

We pre-process the acquired images to extract the ROI. Most of the prior work discussed in Chapter 1.4 perform this step along with additional pre-processing steps to enhance the images. These additional steps attempt to modify all images to be somewhat similar, as the processing steps performed after pre-processing use hand-crafted features which are designed for homogeneous images. Pre-processing steps commonly used, strive to reduce noise, enhance contrast, image smoothing and image sharpening. These traditional pre-processing techniques are necessary when using hand-crafted features, but ideally one would skip these steps as they are computationally expensive and they alter the images to appear unnatural. For these reasons, we apply no additional pre-processing steps.

The ROI extraction algorithm is based on edge detection. The full pre-processing procedure can be seen in Figure 5.1. One could detect both the gauge corner and the field corner from the acquired image, however, it would be computationally faster to simply focus on one edge. We locate the edge by estimating the derivative of the horizontal grey-scale histogram. Note that we do not perform the derivative on the image but rather on the grey-scale histogram. A derivative calculated using simply a one-dimensional histogram is drastically more efficient than on a two-dimensional image.

We utilize only the top half of the acquired image (Figure 5.1(b)) to calculate the horizontal grey-scale histogram based on the light intensity $I(x, y)$ of an image $\boldsymbol{x}$. The light intensity values vary between black (-1) and white (1) pixels, $-1 \leq I \leq 1$. The results of this operation can be seen in Figure 5.1(c), with the horizontal grey-scale histogram expressed by Equation 5.1

$$H(y) = \sum_{x=0}^{1023} I(x, y) \tag{5.1}$$

where $0 \leq x \leq 1023$, $0 \leq y \leq 319$, and the top left corner is the origin ($x = 0$, $y = 0$).

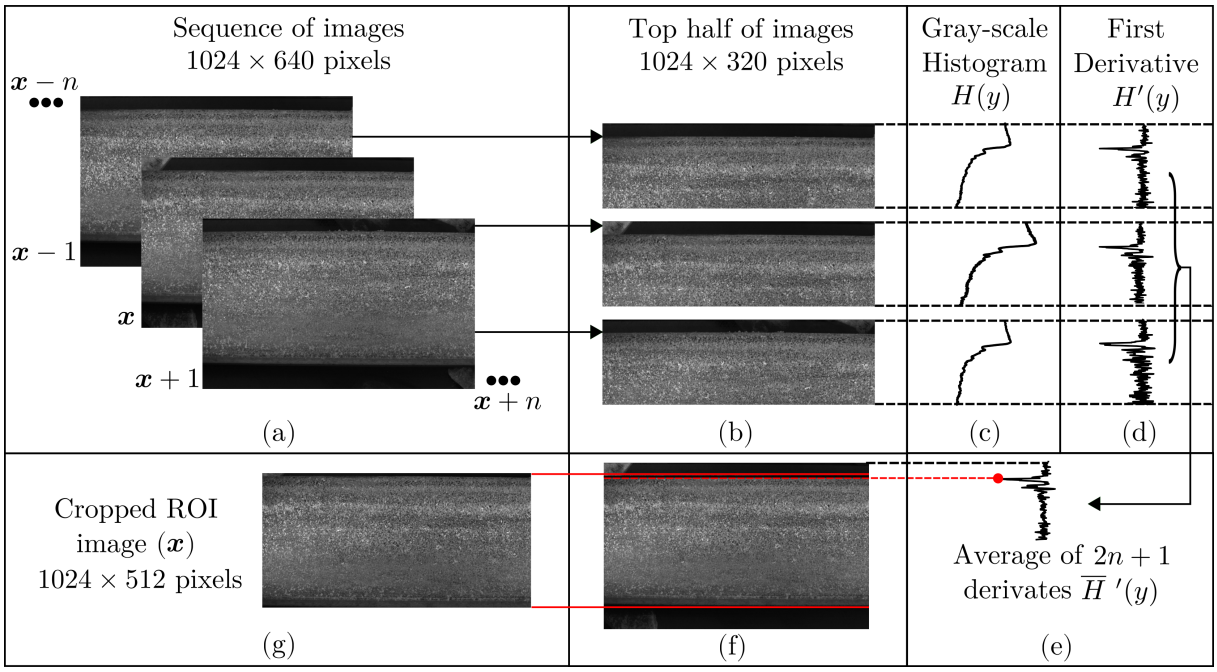


Figure 5.1: Pre-processing procedure. (a) The sequence of acquired images. (b) The top half of the images. (c) Grey-scale histogram. (d) First derivative. (e) Average derivative. (f) Located edges. (g) Cropped ROI image. Best viewed by zooming in on the electronic copy.

The derivative of the grey-scale histogram was approximated by finite forward differences. The results of this first derivative can be seen in Figure 5.1(d), where the location of the gauge corner can be interpreted by the large peak in the derivative which is caused by the difference in light intensity between the rail surface and the background. The numerical derivative is expressed by Equation 5.2, where pi is a single row of pixels ($pi = 1$).

$$H'(y) = \lim_{pi \to 0} \frac{H(y + pi) - H(y)}{pi} \tag{5.2}$$

During the development of this algorithm, it was noticed that when a crosstie was present in the image, the grey-scale histogram would not sufficiently show the edge. In return, the derivative would not have a peak at the gauge corner. To compensate for this, we designed the algorithm to consider n images before and after the image $\boldsymbol{x}$. One would not expect the gauge corner to shift drastically between consecutive frames starting from, $\boldsymbol{x} - n$ up until $\boldsymbol{x} + n$. We then locate the edge based on the peak value of the average derivative of $2n + 1$ images (refer to Figure 5.1(e) and Equation 5.3).

$$\overline{H}\,'(y) = \frac{1}{2n+1} \sum_{i=-n}^{n} H_{\boldsymbol{x}+i}(y) \tag{5.3}$$

Considering that many defects take place at or near the gauge corner, we add a 10-pixel buffer to the location of the detected edge (Equation 5.3), as to ensure the gauge edge is fully captured. The head width of the rail was measured to be 70 mm (Figure 1.1(a)) which is approximately 500 pixels. Therefore, once the gauge corner has been found and the 10-pixel buffer has been added, we assume the field corner is positioned 512 pixels away from this location. Figure 5.1(f) depicts the gauge corner by the dashed red line, whereas the solid top red line includes the 10-pixel buffer and the solid bottom red line is positioned 512 pixels away. Finally, we crop the ROI as can be seen in Figure 5.1(g).

The described pre-processing algorithm was successfully applied with $n = 3$, to all 11830 acquired images where the correct ROI was extracted. The algorithm was robust at finding the gauge corner even when this edge was positioned towards the centre or the top portion of the image.

5.2 Processing

The ROI images from the pre-processing step is used in this processing step. We use the trained VAE model to infer reconstructions ($\boldsymbol{x'}$) of the query images $\boldsymbol{x}$. Since the model was trained exclusively on non-defective images of a rail surface, we expect the model to have only captured the essence of healthy images. Therefore, given an unseen healthy image we expect all regions within the image to be correctly reconstructed. Whereas, given an unhealthy image we do not expect the model to be able to reconstruct the defected regions within the image. Under this notion we can compare $\boldsymbol{x}$ to $\boldsymbol{x'}$ in order to locate regions within the image that are incorrectly reconstructed. This allows us to segment any defects located within an image.

The foremost basis of using a deep learning model such as our VAE, is that we can learn the dynamic conditions in which images are acquired at, to ensure a fair comparison between $\boldsymbol{x}$ and $\boldsymbol{x'}$ is always possible. This implies that regardless of the environmental conditions and/or the rail surface conditions, we can still compare these images because $\boldsymbol{x'}$ will represent whatever conditions $\boldsymbol{x}$ were acquired at.

During inference we can use multiple Monte Carlo samples ($L > 1$), so that we can compare numerous reconstructions $\boldsymbol{x'}$ to a single $\boldsymbol{x}$. We can then segment defects based on an average comparison. A description of the processing step is outlined in Figure 5.2, where the architecture of the encoder Q and decoder P are described in Table 4.1 with trained parameter θ^*.

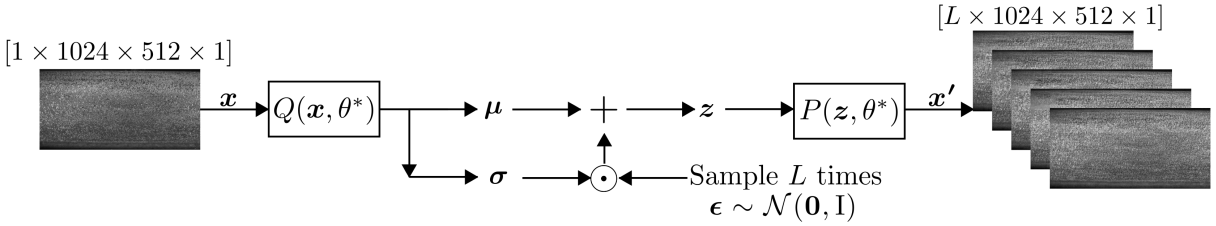


Figure 5.2: Processing procedure.

In order to show the variance in the reconstruction of L samples, Figure 5.3 depicts the standard deviation (assuming a Gaussian distribution) within 16 samples of the reconstructions shown in Figure 5.2. We can see there is little variance in the edges, especially the field edge. The majority of the variance takes place on the running surface. The average standard deviation in the figure is 0.075, with a maximum and minimum of 0.22 and 0.004 respectively.

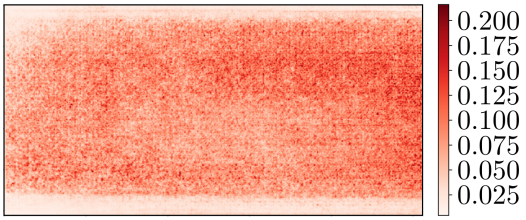


Figure 5.3: Variance within reconstructed samples, $L = 16$.

5.3 Post-processing

The original $\boldsymbol{x}$ and reconstructed $\boldsymbol{x'}$ images from the processing step (VAE) are used in this post-processing step. The premise of this sub-chapter is to compare $\boldsymbol{x}$ to $\boldsymbol{x'}$ in order to correctly identify healthy and defected images. Given that an image is healthy, our goal is to ensure that no regions are segmented. Whereas, given that an image is unhealthy, our goal is to segment any defects that are present within the image.

The proposed pre-processing procedure can be split into three steps. Firstly, we compare local statistics of $\boldsymbol{x}$ and $\boldsymbol{x'}$ in order to obtain an average similarity index over L samples. We then perform binary thresholding on the average similarity index. Finally, we apply morphological transforms on the binary threshold results in order to segment any defects and to remove noise induced from the previous two steps. An anomaly score is based on the outcome of the morphological operation.

5.3.1 Image Comparison

As discussed in Chapter 4.1.1, element-wise metrics which compare pixels to pixels are insufficient to represent the difference between two images. Most of the popular image quality assessments such as the SSIM metric, use local statistics computed with a local window which moves pixel-by-pixel over the entire image [94]. Wang *et al.* [90] proposed that instead of using a square window, one should rather use a circular-symmetric Gaussian weighting function $\boldsymbol{w}$, with a specified standard deviation $\sigma_{\boldsymbol{w}}$, normalized to unit sum ($\sum\sum w(x,y) = 1$). Such an operation exhibits locally isotropic properties. We follow this approach to calculate the mean intensity values $\boldsymbol{\mu_x}$ and $\boldsymbol{\mu_{x'}}$ of images $\boldsymbol{x}$ and $\boldsymbol{x'}$ respectively.

This operation is implemented as a convolution of the kernel $\boldsymbol{w}$ shifted over all the intensity values $I(x,y)$ in an image. This is equivalent to applying a symmetrical two-dimensional Gaussian blurring/smoothing function over an image. Equation 5.4 illustrates this calculation at a single location (x,y) within the full 1024×512 pixel image, where $*$ indicates a mathematical convolution and $\boldsymbol{w}$ has an odd window size of $W \times W$. An illustration of this calculation can be seen in Appendix B.3.

$$\mu(x,y) = (I * w)(x,y) = \sum_{x_w=0}^{W} \sum_{y_w=0}^{W} \left[I\Big(x + \frac{1-W+2x_w}{2},\ y + \frac{1-W+2y_w}{2}\Big) w(x_w,\ y_w) \right] \tag{5.4}$$

In essence, this function gives us a weighted average of the intensity values surrounding point (x,y). Points closer to (x,y) have a larger weighting than points further away. The results of applying a 5×5 Gaussian weighting function with $\sigma_{\boldsymbol{w}} = 1.0$ over $\boldsymbol{x}$ and $\boldsymbol{x'}$ can be seen in Figure 5.4.

We then compare the mean intensity values using Equation 5.5 for multiple reconstructions ($L > 1$) to get an averaged error. Note that $\boldsymbol{x}$, $\boldsymbol{x'}$, $\boldsymbol{\mu_x}$, $\boldsymbol{\mu_{x'}}$ and $\boldsymbol{E}$ are all still at the size of 1024×512.

$$E(x,y) = \frac{1}{L} \sum_{l=1}^{L} \left[\boldsymbol{\mu}_{\boldsymbol{x'}}^{(l)} - \boldsymbol{\mu_x} \right] \tag{5.5}$$

Figure 5.5 illustrates the average error over 16 samples. The defects can be seen in the figure. The E values will be close to 0 when the reconstructions are similar to the original image. Defects will appear to be either darker ($E > 0$) or brighter ($E < 0$) than the reconstructions.

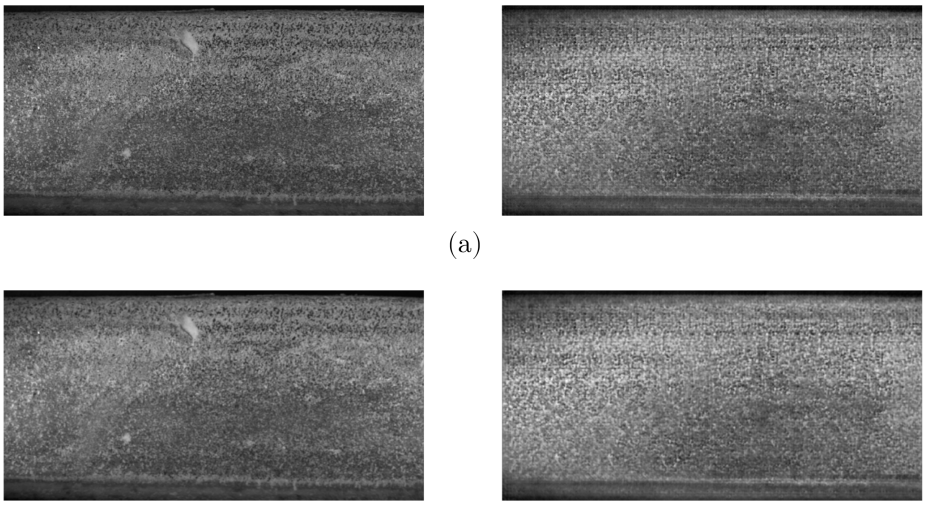


Figure 5.4: Original and reconstructed image. (a) Before smoothing. (b) After smoothing.

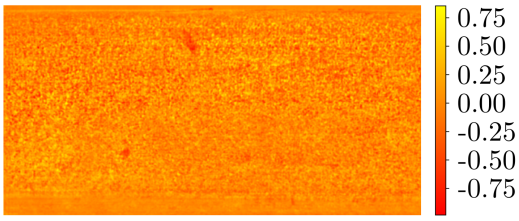


Figure 5.5: Average error, $L = 16$.

5.3.2 Thresholding

Thresholding is the simplest method to perform image segmentation. We apply thresholding over the continuous values of $\boldsymbol{E}$ to produce a binary image which indicates abnormal regions within $\boldsymbol{E}$. Binary images are described by only white (1) and black (0) pixels. The results are also referred to as True (1) or False (0).

As defects may be either brighter or darker than the reconstructions, we apply thresholding in both directions. Equation 5.6 and Equation 5.7 indicate both thresholds, where the thresholding scalar constants c_d and c_b, can be determined using a validation dataset. Subscripts d and b refer to the defected region being either darker or brighter than the reconstructed region respectively. We further discuss the thresholding constants in Chapter 5.4.1. Note that we use the negative of c_b in Equation 5.7.

$$T_d(x,y) = \begin{cases} 1 & \text{if, } E(x,y) > c_d \\ 0 & \text{else} \end{cases} \tag{5.6}$$

$$T_b(x,y) = \begin{cases} 1 & \text{if, } E(x,y) < -c_b \\ 0 & \text{else} \end{cases} \tag{5.7}$$

The final threshold map is simply the combination of Equation 5.6 and Equation 5.7, as shown by Equation 5.8.

$$T(x,y) = T_d(x,y) + T_b(x,y) \tag{5.8}$$

Figure 5.6 depicts thresholding results on an error map, with $c_d = 0.32$ and $c_b = 0.36$. From the figure one can see that both thresholds (Equation 5.6 and Equation 5.7) are capable of detecting different aspects of the defect and that the combination (Figure 5.6(e)) of the thresholds describes the defect well. The final threshold still contains unnecessary (noise) pixels which are mistaken as faults. The detected defect is also not fully enclosed.

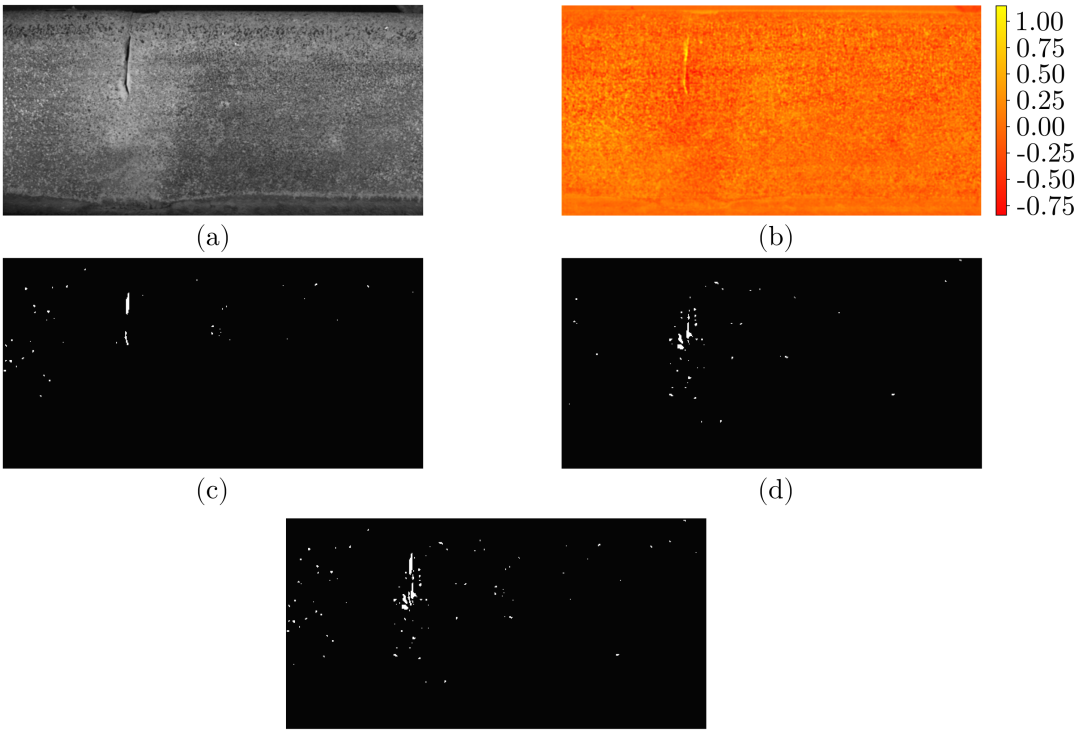


Figure 5.6: Threshold procedure. (a) Original image. (b) Average error, $L = 16$. (c) Dark threshold. (d) Bright threshold. (e) Final (combined) threshold.

5.3.3 Morphological Operations

Mathematical morphology was primarily developed by French mathematicians Georges Matheron and Jean Serra [67]. Morphological image processing is used with the goal to remove noise and to enclose the defects shown in the final binary threshold image. We use the two most basic morphological operations called, erosion and dilation to resolve these issues. The purpose of each operation is implied by its name, erosion removes pixels from object boundaries, whereas dilation adds pixels to object boundaries.

The nature of these morphological operations, such as when to remove/add pixels is determined by a pre-defined structuring element (or kernel). The structuring element $\boldsymbol{s}$ is a small binary window of specified shape and size ($S \times S$) which moves pixel-by-pixel over the entire binary threshold image $T(x, y)$. Once again we can implement this operation as a convolution (Equation 5.9 with odd window size), hence why the structuring element is also called a kernel. An illustration of this calculation can be seen in Appendix B.4.

$$M(x,y) = (T * s)(x,y) = \sum_{x_s=0}^{S} \sum_{y_s=0}^{S} \left[T\Big(x + \frac{1-S+2x_s}{2},\ y + \frac{1-S+2y_s}{2} \Big) s(x_s,\ y_s) \right] \tag{5.9}$$

The unit sum of the defined structuring element is an integer, $i_s = \sum\sum s(x, y)$. We use this integer to define the rules for erosion as expressed by Equation 5.10. The rules for dilation are shown by Equation 5.11.

$$M_e(x, y) = \begin{cases} 1 & \text{if, } M(x, y) = i_s \\ 0 & \text{else} \end{cases} \tag{5.10}$$

$$M_d(x, y) = \begin{cases} 1 & \text{if, } M(x, y) > 0 \\ 0 & \text{else} \end{cases} \tag{5.11}$$

In essence, the output of erosion is 1, only if all of the True binary values in the structuring element coincide with True binary threshold values. Whereas, the output of dilation is 1, if at least one of the True binary values in the structuring element coincide with a True binary threshold value.

Erosion and dilation are commonly combined to describe additional morphological operations. Morphological closing is defined as dilation followed by erosion, which is useful in closing or connecting gaps in images. Morphological opening is defined as erosion followed by dilation, which is useful in removing noise from images.

Opening and closing can resolve the issues of noise and gaps we had in our thresholding results, hence we use closing followed by opening on the final threshold result as depicted by Figure 5.7. Both closing and opening have a circular structuring element with radius 5 and 3 respectively. From the figure we can see how closing has enclosed the defect and how opening has removed the noise. The last dilation in opening (Figure 5.7(f)) is the final output of the inference algorithm.

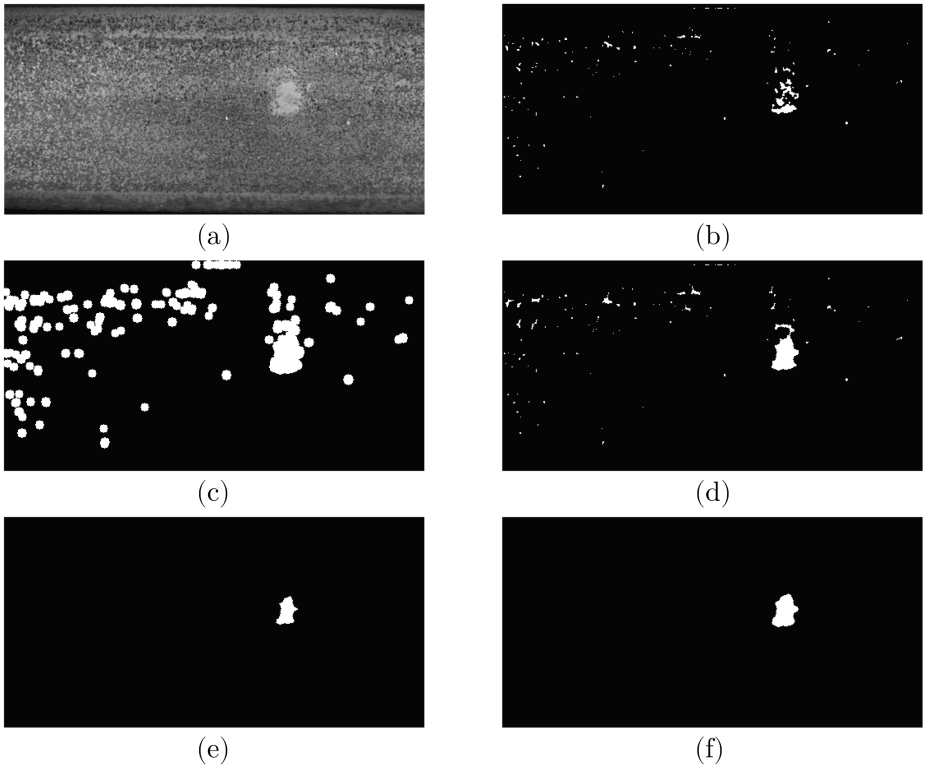


Figure 5.7: Morphological procedure. (a) Original image. (b) Final threshold. (c) Closing: dilation. (d) Closing: erosion. (e) Opening: erosion. (f) Opening: dilation.

5.3.4 Anomaly Score

Finally we determine the accuracy of our inference algorithm by comparing the output to a ground truth result, Figure 5.8. A healthy image is correctly classified, when the output of the inference algorithm is black with no segmented regions. Whereas, a unhealthy image is correctly classified, when there is some overlap between the output of the inference algorithm and the ground truth defects. For example, Figure 5.7(f) sufficiently and correctly segments the defect shown in Figure 5.8.

Figure 5.8: Ground truth segmented defect.

5.4 Discussion

All of the algorithms described in this chapter can handle a batch of images in a highly parallel manner. An analogy can be drawn between adaptive thresholding techniques and the proposed inference algorithm. The processing step produces adapted reconstructions which closely resemble the input. The original and reconstructed images are compared to one another, to produce an adapted error in which we apply simple thresholding techniques to segment the defects.

5.4.1 Threshold Constants

The larger the thresholding constants, the less likely anomalies will be detected and the higher the constants, the more likely anomalies will be detected. Note that if the chosen/calculated constants are the same ($c_d = c_b$), then Equation 5.5 should express the absolute difference and only a single thresholding constant is needed.

In order to determine possible thresholding constants, we perform a mesh grid search on the constants using the validation dataset (unseen healthy images). Figure 5.9 depicts the threshold constants which vary between all images being detected as unhealthy (150) and all images being detected as healthy (0). The red star in Figure 5.9 indicates the lowest combination of threshold values which correctly detects no defects, $c_d = 0.28$ and $c_b = 0.31$. We find that in general c_d values are less than c_b values. The results in Figure 5.9 concur with this observation.

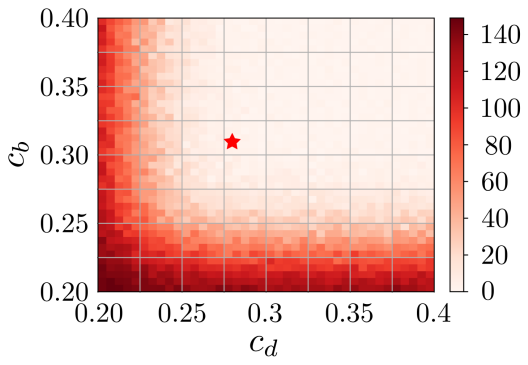


Figure 5.9: Anomaly score on the validation dataset using a mesh grid of threshold constants.

5.4.2 Chosen Parameters

In Chapter 6 we perform experiments on our described inference algorithm. The final inference algorithm predominately consists of the parameters mentioned throughout this chapter. The only parameters we vary are the thresholding constants (c_d and c_b) and the number of samples (L) used. For clarification we repeat the fixed chosen parameters.

We consider $n = 3$ frames before and after the image we extract the ROI from. We perform smoothing with a 5×5 circular-symmetric Gaussian weighting function with a standard deviation of $\sigma_{\boldsymbol{w}} = 1.0$, normalized to unit sum. Morphological closing and opening use a circular structuring element with radius 5 (window size 11×11) and 3 (window size 7×7) respectively.

Chapter 6

Experimental Investigation

As the proposed deep learning method has not been used in railway condition monitoring (or elsewhere) before, we perform numerical experiments to evaluate our algorithm with real world data. The test dataset as described in Table 2.1, consists of 150 healthy images and 150 unhealthy images with corresponding ground truth results.

In addition to our proposed algorithm which operates with images at 1024×512 pixels, we also utilize algorithms with lower resolution images at 512×256 and 256×128 pixels. The original images were resized using bicubic interpolation. These two additional algorithms were trained and implemented in a similar manner to the described procedure in Chapter 4 and 5. The architecture of these models are reported in Appendix C.

We performed experiments to evaluate the accuracy and speed of all three models under different conditions. The accuracy of the algorithms is assessed using receiver operating characteristic (ROC) curves. Firstly, we determine the impact of the dimensionality of a VAEs latent space. We also investigate the performance of the algorithms using different Monte Carlo samples. Finally, we compare the inference speed of the algorithms under different conditions.

6.1 Receiver Operating Characteristic

ROC curves are used to evaluate the diagnostic performance of binary classification models. These curves give flexibility in how we present the performance of algorithms by using different thresholds. This allows operators, such as engineers to interpret ROC curves in order to trade-off performance characteristics of a model. ROC curves of different models can be directly compared to one another without selecting threshold values.

A confusion matrix is used to calculate the results of a ROC curve. Table 6.1 represent our confusion matrix, where defected images are the positive class and healthy images are the negative class. Perfect classification is defined when the false negative (FN) and false positive (FP) values are 0, so that the true positive (TP) and true negative (TN) values are 150.

Table 6.1: Confusion matrix.

		Actual class	
		Positive	Negative
Predicted class	Positive	TP	FP
	Negative	FN	TN
Total		**150**	**150**

ROC curves represent the true positive rate (TPR) as a function of the false positive rate (FPR), as described by Equation 6.1 and 6.2 respectively. The TPR is also referred to as recall, sensitivity or the probability of fault detection. The FPR is also referred to as the negative specificity (1−specificity) or the probability of false alarm.

$$\text{TPR} = \frac{\text{TP}}{\text{TP+FN}} = \frac{\text{TP}}{150} \tag{6.1}$$

$$\text{FPR} = \frac{\text{TN}}{\text{TN+FP}} = \frac{\text{TN}}{150} \tag{6.2}$$

Typical binary classification models only have a single threshold which is used to define the ROC curve by increasing the threshold value so that TPR and FPR decrease from 1 to 0. Each threshold value is used to calculate TPR and FPR based on the classification performance using the test dataset. In our model we have two threshold values (c_d and c_b) which we use to define our ROC curve. We increase both thresholds in a mesh grid manner, as done in Chapter 5.4.1. By using a mesh grid of values we do not get a single ROC curve but rather a ROC scatter plot, as can be seen in Figure 6.1(a).

The scatter points are not evenly distributed across the figure, due to how the mesh grid was defined (linear spacing between a large threshold range). Therefore, we do not use the average results but, we rather define our ROC curve as the best TPR results for each FPR. The final ROC curve shown in Figure 6.1(b), can be used by operators to select desirable characteristics. For example, the blue star leads to, TPR = 141/150 = 0.94 and FPR = 74/150 = 0.49 which correspond to threshold constants, $c_d = 0.21$ and $c_b = 0.45$. These results are interpreted as, correctly identifying 141 defected images and 74 healthy images.

A scalar performance metric for ROC curves are defined by the area under the curve (AUC). A model which randomly predicts results (coin toss) will result in, AUC = 0.5, as shown by the dashed lines in Figure 6.1. A higher AUC result indicates better performance in a model. Perfect classification would result in a ROC curve which resembles a step function where, TPR = 1 for all FPR so that AUC = 1.

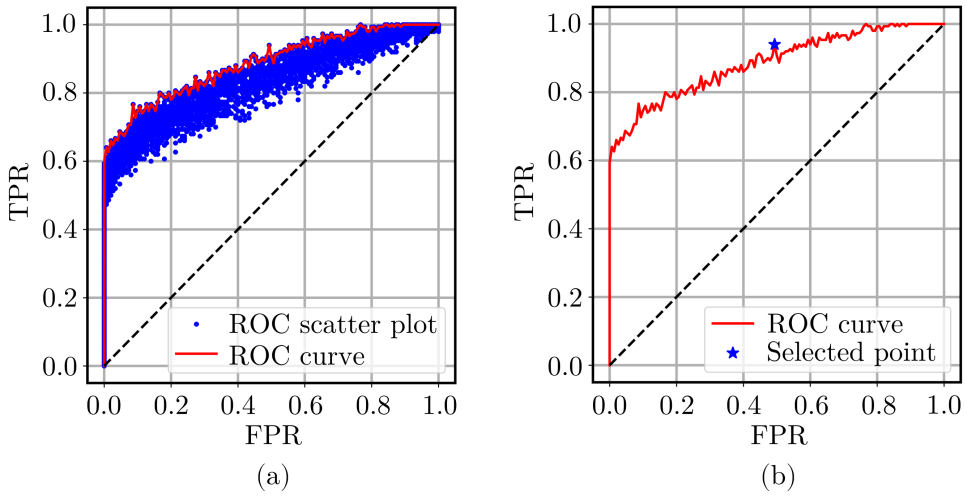


Figure 6.1: ROC results on the test dataset. (a) ROC scatter plot and the best ROC curve with optimal threshold constants. (b) Final ROC curve with selected characteristics.

6.2 Dimensionality of Encoded Latent Space

The quality of reconstructed images in a VAE is dependent on the dimensionally M of the latent space. If the dimensionally M is too large, the model simply copies its inputs to perfectly reconstruct the results. This is an issue as we do not want defected regions to be reconstructed. Nevertheless, if the dimensionally M is too small, the model produces poor reconstructions which cannot even reconstruct the healthy regions within an image. Figure 6.2 illustrates reconstructed results for models trained with different latent space dimensions ($Mc = M/8$).

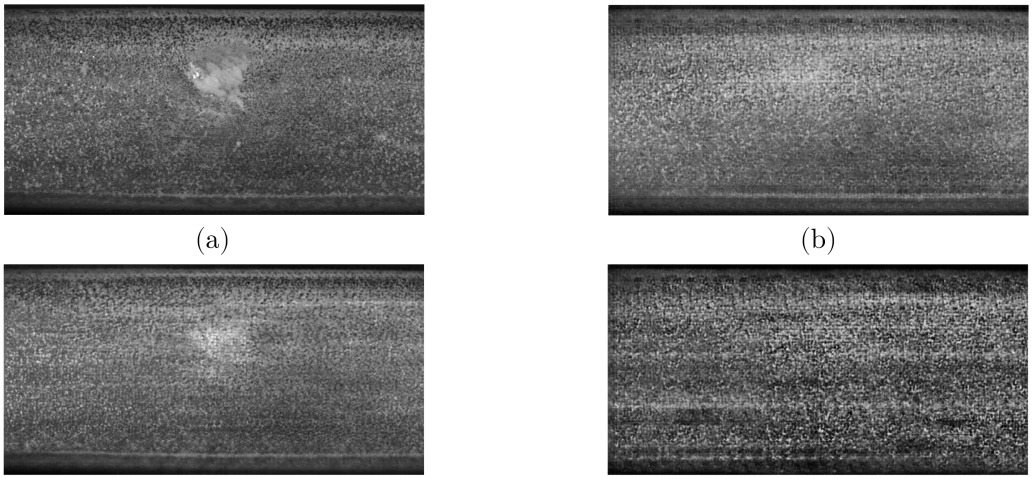


Figure 6.2: Images at resolution 1024×512. (a) Original image. (b-c) Reconstructed images with: (b) $M_c = 512$, (c) $M_c = 1024$, (d) $M_c = 128$.

We performed experiments to determine the impact in our model's performance based on the dimensionality of the channels M_c in the VAEs latent space. The network architecture of all three models are described in Chapter 4.3.2, Appendix C.1 and C.2. Each time we adjust M_c we need to train an additional new model. Due to the computationally expensive nature in training deep networks with high resolution images, we only performed an in-depth search on M_c values with our model trained at a resolution of 256×128 pixels. The results are depicted by Figure 6.3, with $L = 16$.

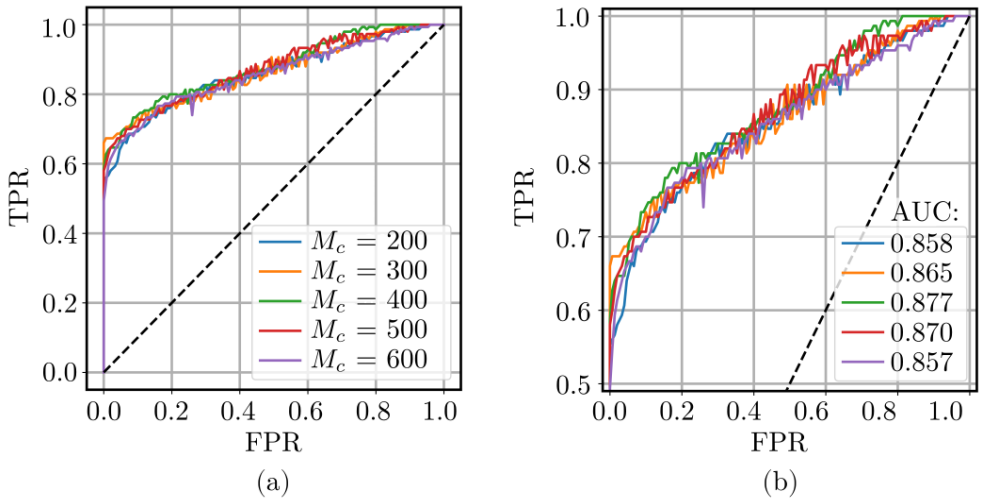


Figure 6.3: ROC curve.(a) Varying dimensionality M_c at 256×128 resolution. (b) Zoomed-in.

In Figure 6.3 we can see how the AUC values initially increase with M_c until the best results are achieved at $M_c = 400$, after which the AUC values decrease with an increase in M_c. The ratio between the input dimensionality and the best M_c values is, ratio $= (256)(128)/400 = 81.92$. We use this ratio to determine an initial guess for M_c values when training at higher resolutions.

We performed additional searches on M_c values for our models at a resolution of 1024×512 and 512×256 pixels. We found good results with $M_c = 512$ for both of the higher resolution models. Figure 6.4 depicts the ROC curves for the best M_c values at all three resolutions, with $L = 16$. From Figure 6.4 we can see a drastic improvement in the ROC curve between the models trained at 256×128 and 512×256, however, we only gain a slight improvement with the highest resolution model. The improvement in higher resolution models is because these models can detect smaller defects than the lower resolution models, as illustrated in Appendix B.5. Appendix B.6 contains additional reconstructed images to emphasise on the quality of the results. Table 6.2 contains a summary of the results.

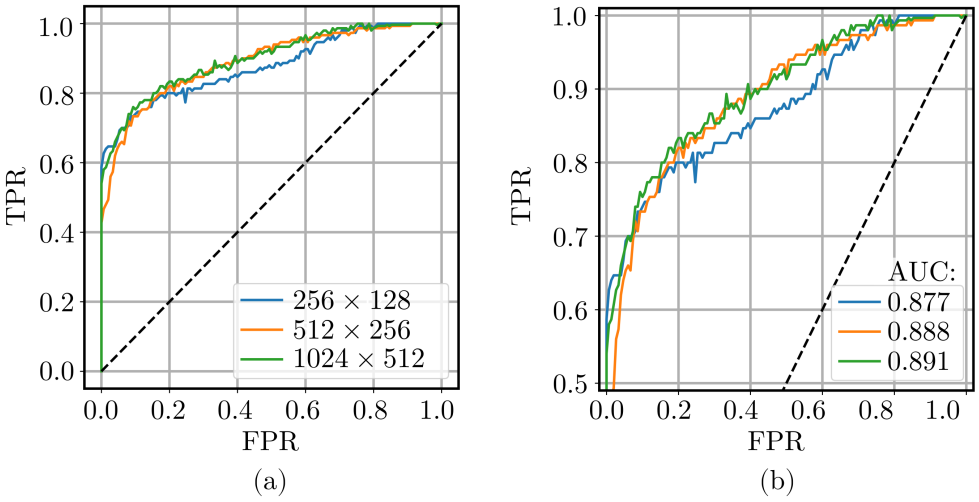


Figure 6.4: ROC curve. (a) Best M_c values for all resolutions. (b) Zoomed-in.

Table 6.2: Results from experiments on dimensionality M_c.

Model resolution	M_c	L	AUC
256×128	400	16	0.877
512×256	512	16	0.888
1024×512	512	16	0.891

6.3 Number of Monte Carlo Samples

During inference we can use multiple Monte Carlo samples (L) to get an averaged error between numerous reconstructions, and a single input image. We performed experiments to determine the influence in our algorithms performance based on how many samples are used.

We use the models with optimal M_c values, as described by Table 6.2. ROC curves produced for different L values are very similar where individual curves are not easily distinguishable from one another, similar to the results shown in Figure 6.3. We also use many different samples which contribute towards producing ROC curves which are not easily interpretable. Hence, we

only display the AUC results from the experiments, as depicted by Table 6.3.

The sampling procedure is random, therefore, we performed multiple experiments at a single L value to get an averaged AUC result. In Table 6.3 we can see that in general the AUC results increase with L. The largest increase in results takes place between, $L = 1$ and $L = 16$, after which the results remain fairly constant.

Table 6.3: Results of experiments based on the number of samples used.

	Average AUC for Model resolution		
Number of samples, L	256×128 with, $M_c = 400$	512×256 with, $M_c = 512$	1024×512 with, $M_c = 512$
1	0.872	0.880	0.882
2	0.872	0.882	0.884
4	0.874	0.884	0.888
8	0.877	0.884	0.888
16	0.877	0.888	0.891
32	0.882	0.888	0.892
64	0.883	0.888	0.892

6.4 Inference Speed

Inspection systems are expected to be able to perform in real-time, as it is infeasible to store the immense amount of image data recorded over long hauls. We performed experiments to determine the inference speed of our proposed algorithm under varying conditions.

We can calculate the number of frames (images) our algorithm can process in a second (fps) by assuming the pre-processing, processing and post-processing algorithms run in series. Equation 6.3 and Figure 6.5 describe the relationship between the length of the rail surface (a) that a camera captures at a frame rate, to the velocity (V) in which the camera moves. As mentioned in Chapter 2.3, $a = 0.15\ m$.

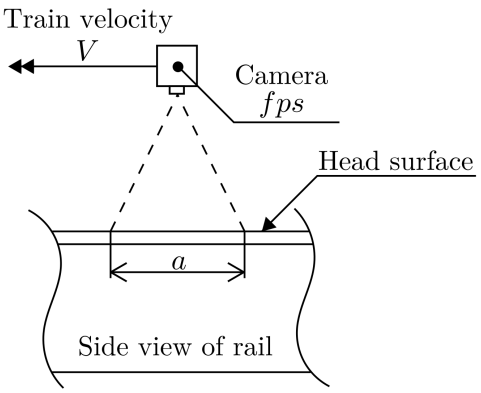


Figure 6.5: Relationship between fps and the velocity a train travels at.

$$V = a \cdot fps = 0.15 fps \tag{6.3}$$

Table 6.4 depicts the inference speed for all models with multiple sample. The table also includes the highest possible fps for each portion of the inference algorithm, along with a percentage indicating where the computation time is spent. The computer hardware specifications were an Intel Core i7-4790 CPU @ 3.60GHz, with 32GB of RAM and a single Nvidia GeForce GTX Titan X GPU (12GB VRAM). The computer software specifications were Ubuntu 16.04, Python 3.6, Tensorflow v1.10 with CUDA 9.0.

In Table 6.4 the train's velocity varies between 4 to 518 km/h. We can exceed the world's fastest world's fastest railway trains: Fuxing Hao CR400AF/BF that has a continuous operation speed of approximately 400 km/h. The fewer samples used, the quicker the algorithm. The post-processing step is initially the most computationally expensive operation, however, with more samples it becomes less expensive. The processing step is the opposite as it becomes the most computationally expensive operation as more samples are used. From the table we can also see that the model resolution considerably impacts the speed.

Table 6.4: Inference algorithm speed for all models.

Model resolution	Number of samples, L	Train km/h	Inference fps	Possible fps with (%) contribution to inference		
				Pre-process	Process	Post-process
256×128	1	518	960	12860 (7)	2179 (44)	1978 (49)
	2	448	829	12860 (6)	1621 (52)	1957 (42)
	4	341	631	12860 (5)	1081 (58)	1717 (37)
	8	232	430	12860 (3)	639 (68)	1461 (29)
	16	138	255	12860 (2)	357 (71)	962 (27)
	32	76	140	12860 (1)	183 (76)	621 (23)
	64	40	74	12860 (1)	87 (85)	530 (14)
512×256	1	178	330	2831 (12)	988 (33)	601 (55)
	2	157	290	2831 (10)	742 (39)	573 (51)
	4	126	233	2831 (8)	495 (47)	521 (45)
	8	90	166	2831 (6)	297 (56)	437 (38)
	16	55	101	2831 (4)	163 (62)	297 (34)
	32	32	59	2831 (2)	79 (75)	256 (23)
	64	17	32	2831 (1)	41 (77)	142 (22)
1024×512	1	51	94	853 (11)	295 (32)	164 (57)
	2	44	81	853 (9)	212 (39)	155 (52)
	4	34	63	853 (7)	152 (42)	123 (51)
	8	24	44	853 (5)	79 (55)	110 (40)
	16	15	27	853 (3)	42 (64)	82 (33)
	32	8	15	853 (2)	21 (73)	60 (25)
	64	4	8	853 (1)	11 (76)	35 (23)

Deep learning algorithms scale well with multiple GPUs as illustrated by the results in Table 6.5. Virtual machine instances were hired from Google Cloud Platform, with hardware specifications of 32 vCPUs, 120GB of RAM and four Nvidia Tesla V100 GPUs (each with 16GB VRAM). The computer software specifications were Debian (9), Python 3.6, Tensorflow v1.12 with CUDA 10.0.

From Table 6.5 we can see how initially the speed almost doubles with each additional GPU. The speed improvement is less drastic given more samples and higher resolution models, due to VRAM limitations. The train's velocity varies between 17 to 3288 km/h, which greatly exceeds the capabilities of any train. Evidently our algorithm scales well with additional GPUs. We can see that the Tesla V100 GPU are faster than the GeForce GTX Titan X GPU, this is due to the fact that these GPUs came out more recently and that they have more VRAM. Note that currently it is possible to have computers with up to 8 GPUs.

Table 6.5: Inference algorithm speed for all models using multiple GPUs.

Model resolution	Number of samples, L	Train velocity (km/h) for multiple GPUs.			
		1 GPU	2 GPUs	3 GPUs	4 GPUs
256×128	1	1317	2368	3160	3288
	2	1166	2051	2782	3245
	4	961	1722	2212	2671
	8	719	1200	1709	2003
	16	476	834	1120	1232
	32	285	509	675	769
	64	159	278	367	425
512×256	1	559	998	1342	1465
	2	498	858	1186	1295
	4	416	736	986	1062
	8	311	530	737	852
	16	207	362	476	558
	32	123	215	276	315
	64	69	120	162	181
1024×512	1	170	292	401	455
	2	149	258	337	406
	4	120	204	268	326
	8	86	143	192	220
	16	55	94	122	145
	32	32	55	75	86
	64	17	30	42	44

6.5 Discussion

From the results we have proved that our proposed NET is capable of learning the complex non-linear nature of healthy high-dimensional images of the surface of a rail. The reconstructed images shown throughout this dissertation are of excellent quality, where one can see how the reconstructed images resemble the conditions of the original input image. This is apparent as we notice how the characteristics of small features such as the rust is reproduced. The adaptive reconstructions allow for a fair comparison to be made between the original input and the reconstructed images, which enables our proposed approach to achieve excellent results.

The results presented in this chapter indicate a constant trade-off between accuracy and speed. The accuracy of the models improve when using higher resolution images and more Monte Carlo samples, but this comes with additional computational expenses. We have shown the importance of selecting a latent dimensionality so that good quality reconstructions are obtained without simply reproducing the input. By training our NET exclusively with healthy images, we have shown that defected regions are not reproduced, given that a good latent dimensionality is used.

Our results demonstrate that higher resolution models are capable of achieving results with better accuracy. The improvement in accuracy is because higher resolution models can detect smaller defects than lower resolution models. There is not much of an improvement in our obtained results between the two higher resolution models (256×128 and 512×256) and we suspect that this is because our test dataset contains very few small defects. One should also note that it is significantly more difficult to train deep learning models which utilize high-resolution images.

We have shown that there is diversity in the reconstructed images when using multiple Monte Carlo samples. Our results demonstrate that by using more samples we get a better average of the discrepancy between the original and reconstructed images. If we use only a few samples and perform inference multiple times on the same image, the results may vary were defects might be segmented or not deepening on the sampling. The average from multiple samples diminishes the stochastic nature of sampling, which leads to consistent results with higher accuracy.

The inference algorithm was designed to be computationally efficient, where a batch of images can be processed at once in a highly parallel manner. We have demonstrated that our proposed algorithm is capable of performing inspection in real-time at speeds which exceed the world's fastest train. We have shown how more complex models are more computationally expensive. Even though these models may be expensive, we have demonstrated that real-time inspection is still possible as deep learning models scale well with multiple GPUs. Although the results we show are impressive, one should note that better results would be obtained if the algorithm were implemented in C programming language by a high performance computing expert. We also expect better results in the future as GPU technologies are developing at a rapid rate, with performance improvements increasing exponentially with a 3.5 month-doubling time [37] (compared to Moore's Law which used to have an 18-month doubling period [43]).

In conclusion we have presented performance characteristics for our proposed algorithm under numerous conditions. Depending on the requirements one may have for an inspection system we have provided multiple solutions to select from in order to suit such needs.

Chapter 7

Conclusion and Recommendations

Rail surface defects are a critical issue in the rail industry. Most of the surface defects can be rectified through rail grinding if the faults are found early enough. Computer vision techniques are a promising solution to detect discrete surface defects in a fast, cheap and accurate manner. The majority of computer vision research for surface defects has utilized hand-crafted features. These approaches are not robust to variations in environmental and rail surface conditions. Deep learning approaches which learn features from data are an apparent solution. In this research we proposed novel deep generative models that learn the complex distribution of high-dimensional images taken of the surface of a rail.

Image data was acquired through an experimental rig which we designed and built. A total of 11830 high-dimensional (1024×512 pixels) images were acquired and used to train and test our proposed algorithms. This new dataset will be made publicly available. The dataset largely consists of healthy images, but also a few defected images. Due to the sparsity in defected images (which is common on most railways), we used a novelty detection approach in which we learned the distribution of only healthy images and detected defects as deviations from this.

Generative models are the ideal solution to model the likelihood of healthy data in an end-to-end deep learning framework. We provided an in-depth discussion on VAEs and GANs, indicating where these approaches have been used in anomaly detection. Both VAEs and GANs have complementary advantages and disadvantages. For this reason, we proposed a novel approach to combine the two models in order to obtain a high-dimensional bidirectional model. We developed a novel loss function which promotes a better reconstruction loss. We also progressively grow our models during training in order to produce high-dimensional images.

An inference algorithm which is used to detect surface defects was described in detail. A pre-processing step was used to extract the ROI of the rail surface from the background through detecting the gauge corner edge. The learned VAE model was used in a processing step in order to reproduce adaptive healthy representations of query images. This deep generative model was robust to variations in environmental and rail surface conditions. A post-processing step was used for unsupervised segmentation. Simple thresholding techniques were used to segment defected regions which are abnormal to the healthy class. Morphological operations were used to enclose detected defects and to clean-up additional noise which may be present in thresholding results.

Numerical experimental investigations were performed on the proposed inference algorithms in order to evaluate the performance on real world data. We utilized models which operate with images at three different resolutions: 1024×512 pixels, 512×256 pixels and 256×128 pixels. We have shown the importance of selecting a latent dimensionality for the VAE so that good qual-

ity reconstructions are obtained without simply reproducing the input (defected regions). Our experimental results demonstrate a constant trade-off between the speed and accuracy of the algorithms. Accurate results are possible by using higher resolution models with many Monte Carlo samples. Although these high accuracy models are computationally expensive, we have shown that our proposed algorithms are very efficient and scale well with multiple GPUs. We have evidence that our proposed models are sufficiently efficient, that they exceed processing requirements for the world's fastest train.

In conclusion our dissertation has catered to a gap in railway research to provide a deep learning approach for computer vision. The proposed models and algorithms have great characteristics and we hope this work will ignite further research from others.

7.1 Recommendations

The following recommendations are made for future work:

1. The image dataset should be built upon, to add more images which are diverse. For example, images which include: different rail reflection properties, different environmental conditions (rain, mud, snow, etc.), turns, switches and crossings. This new dataset should be used to further ensure that the proposed approach is robust to dynamic non-linear environmental and rail surface conditions
2. Transfer learning should be considered. The model must be able to handle new data which differs from the already seen training data. We should not need to train a completely new model given a new dataset.
3. Adversarial attacks should be explored, however, we believe this issue can be alleviated given more training data, and by using domain experts to verify all detected faults. By doing so, we can add the false alarm cases into the training dataset, so that a recurrence is less likely.
4. Labelled data (from domain experts) could be used to diagnose detected defects. This could be coupled with connected components analysis, to point out the location, size and shape of segmented regions. This classification can be either per image or per pixel (semantic segmentation).
5. Utilize multiple sensors (all three cameras, and potentially other sensors).
6. Analyse corrugation defects using recurrent neural networks.

Appendices

Appendix A

Additional VAE Equations

A.1 KL Loss

The KL divergence in a VAE [45] between two Gaussian distributions. We can analytically solve the solution, with M the dimensionality of $\boldsymbol{z}$.

$$-D_{KL}(q_\phi(\boldsymbol{z}) \mid\mid p_\theta(\boldsymbol{z})) = \int q_\phi(\boldsymbol{z}) \log p_\theta(\boldsymbol{z}) \, d\boldsymbol{z} - \int q_\phi(\boldsymbol{z}) \log q_\phi(\boldsymbol{z}) \, d\boldsymbol{z} \tag{A.1}$$

$$-D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \mid\mid p_\theta(\boldsymbol{z})) = \int q_\phi(\boldsymbol{z}) \left(\log p_\theta(\boldsymbol{z}) - \log q_\phi(\boldsymbol{z})\right) d\boldsymbol{z} \tag{A.2}$$

with,

$$\int q_\phi(\boldsymbol{z}) \log p_\theta(\boldsymbol{z}) \, d\boldsymbol{z} = \int \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma^2}) \log \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \boldsymbol{I}) \, d\boldsymbol{z} \tag{A.3}$$

$$\int q_\phi(\boldsymbol{z}) \log p_\theta(\boldsymbol{z}) \, d\boldsymbol{z} = -\frac{M}{2} \log(2\pi) - \frac{1}{2} \sum_{m=1}^{M} (\mu_m^2 + \sigma_m^2) \tag{A.4}$$

and,

$$\int q_\phi(\boldsymbol{z}) \log q_\phi(\boldsymbol{z}) \, d\boldsymbol{z} = \int \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma^2}) \log \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma^2}) \, d\boldsymbol{z} \tag{A.5}$$

$$\int q_\phi(\boldsymbol{z}) \log q_\phi(\boldsymbol{z}) \, d\boldsymbol{z} = -\frac{M}{2} \log(2\pi) - \frac{1}{2} \sum_{m=1}^{M} (1 + \log \sigma_m^2) \tag{A.6}$$

Therefore,

$$-D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \mid\mid p_\theta(\boldsymbol{z})) = \frac{1}{2} \sum_{m=1}^{M} (1 + \log \sigma_m^2 - \mu_m^2 - \sigma_m^2) \tag{A.7}$$

Appendix B

Additional Figures

B.1 ProGAN and BigGAN Generated Images

Figure B.1: ProGAN generated celebrity faces at a resolution of 1024×1024 [41]. Reproduced with the authors' permission.

Figure B.2: BigGAN generated ImageNet samples at a resolution of 512×512 [11]. Reproduced with the authors' permission.

B.2 Coordinate Convolution

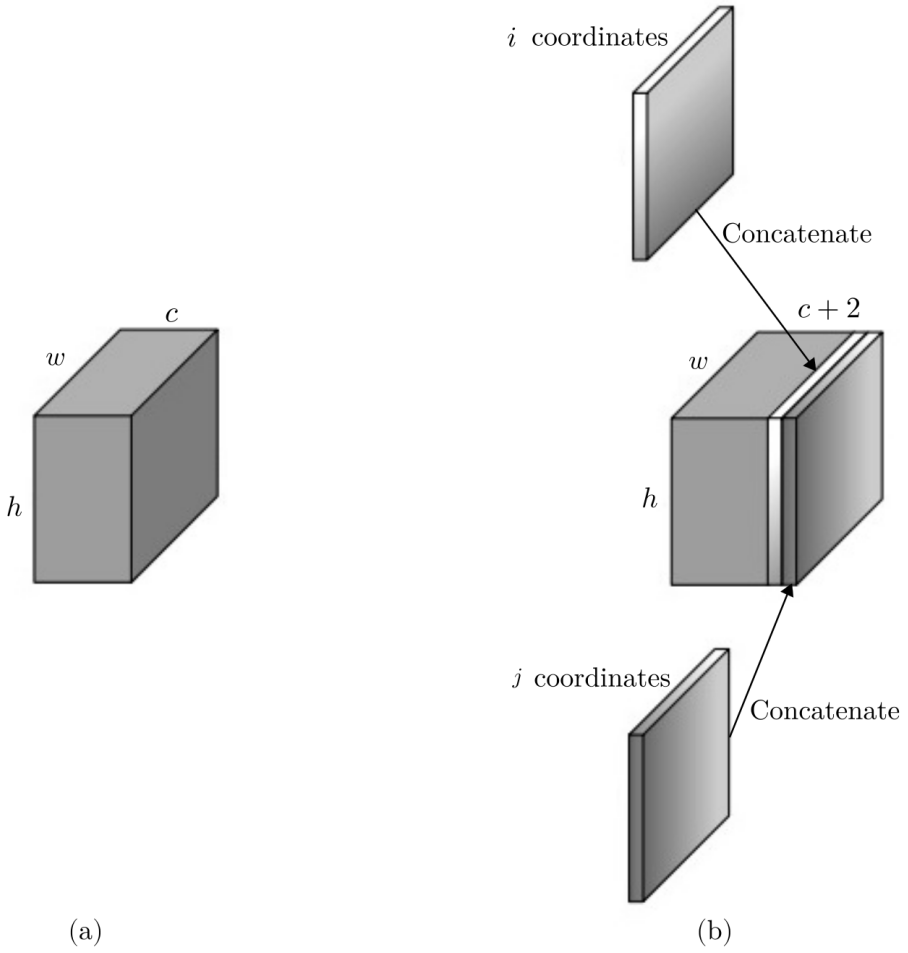


Figure B.3: Convolution input. (a) Usual convolution layer. (b) CoordConv layer [55].

B.3 Image Smoothing

3×3 Gaussian kernel $\boldsymbol{w}$ with, $\sigma_{\boldsymbol{w}} = 1.0$ and $\sum\sum w(x, y) = 1$. Kernel applied to Figure B.4.		
0.07511361	0.1238414	0.07511361
0.1238414	0.20417996	0.1238414
0.07511361	0.1238414	0.07511361

(a) (b)

Figure B.4: Single white pixel image. (a) Before smoothing. (b) After smoothing.

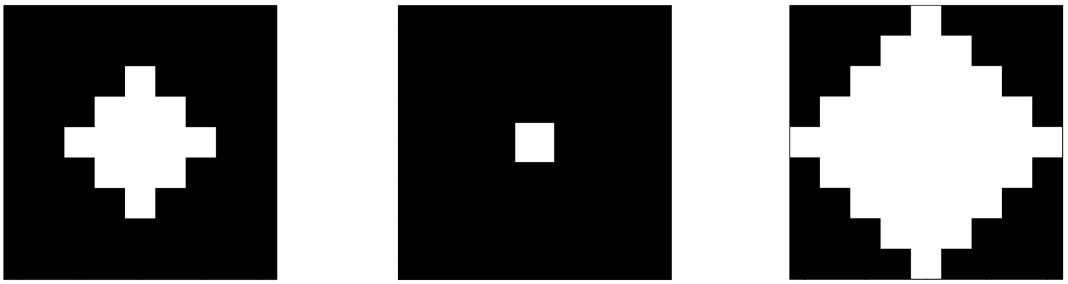
B.4 Morphological Operations

3×3 Cross-shaped kernel. Kernel applied to Figure B.5.		
0	1	0
1	1	1
0	1	0

(a) (b) (c)

Figure B.5: Morphological operations. (a) Original image. (b) Erosion applied to original image. (c) Dilation applied to original image.

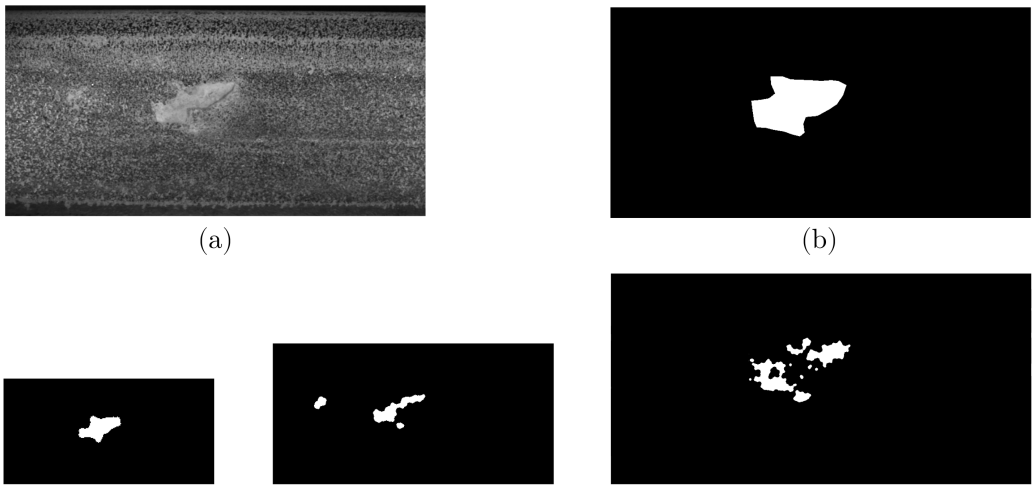
5×5 Circular kernel. Kernel applied to Figure B.6.				
0	0	1	0	0
0	1	1	1	0
1	1	1	1	1
0	1	1	1	0
0	0	1	0	0



(a) (b) (c)

Figure B.6: Morphological operations. (a) Original image. (b) Erosion applied to original image. (c) Dilation applied to original image.

B.5 Defect Localisation for Different Models



(a) (b) (c) (d) (e)

Figure B.7: Large size defect. (a) Original image. (b) Ground truth. (c-e) Anomaly score for model resolution: (c) 256×128, (d) 512×256, (e) 1024×512.

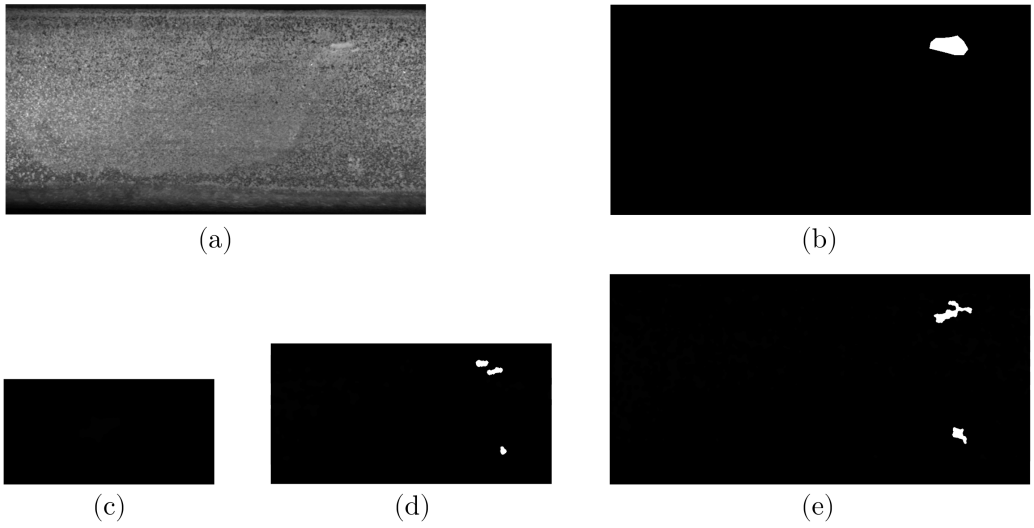


Figure B.8: Medium size defect. (a) Original image. (b) Ground truth. (c-e) Anomaly score for model resolution: (c) 256×128, (d) 512×256, (e) 1024×512.

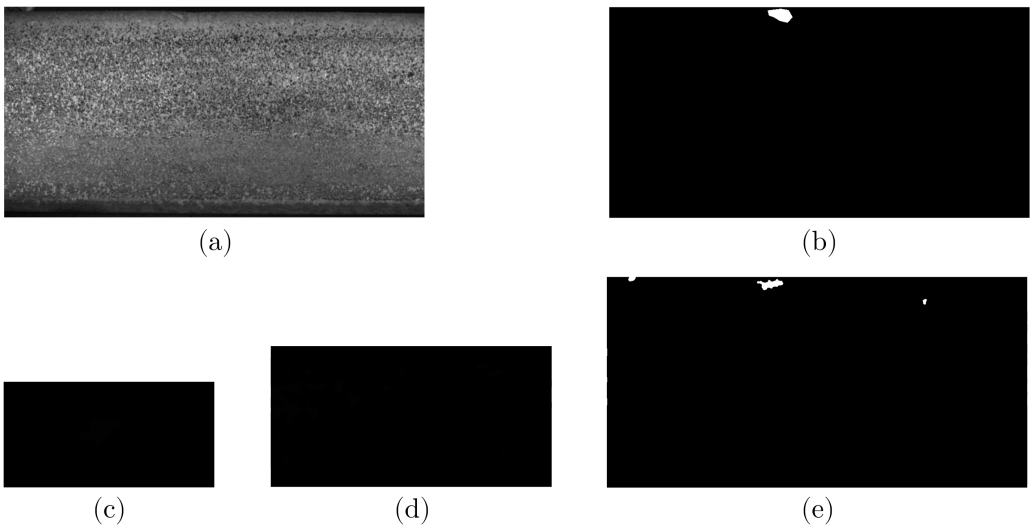


Figure B.9: Small size defect. (a) Original image. (b) Ground truth. (c-e) Anomaly score for model resolution: (c) 256×128, (d) 512×256, (e) 1024×512.

B.6 Reconstructed Results

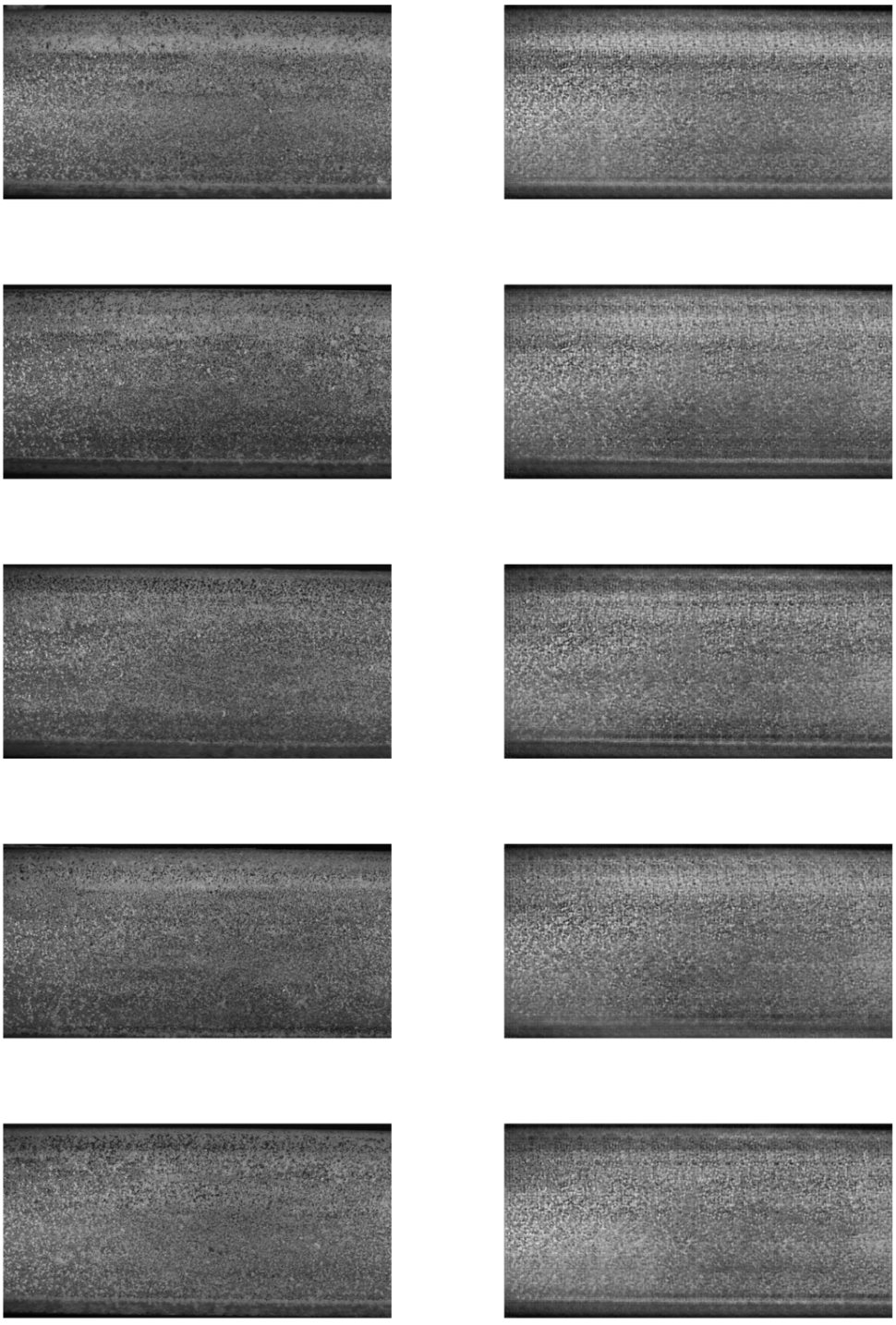


Figure B.10: Original (left column) and reconstructed (right column) images from the healthy test dataset at a resolution of 1024×512, with $M_c = 512$.

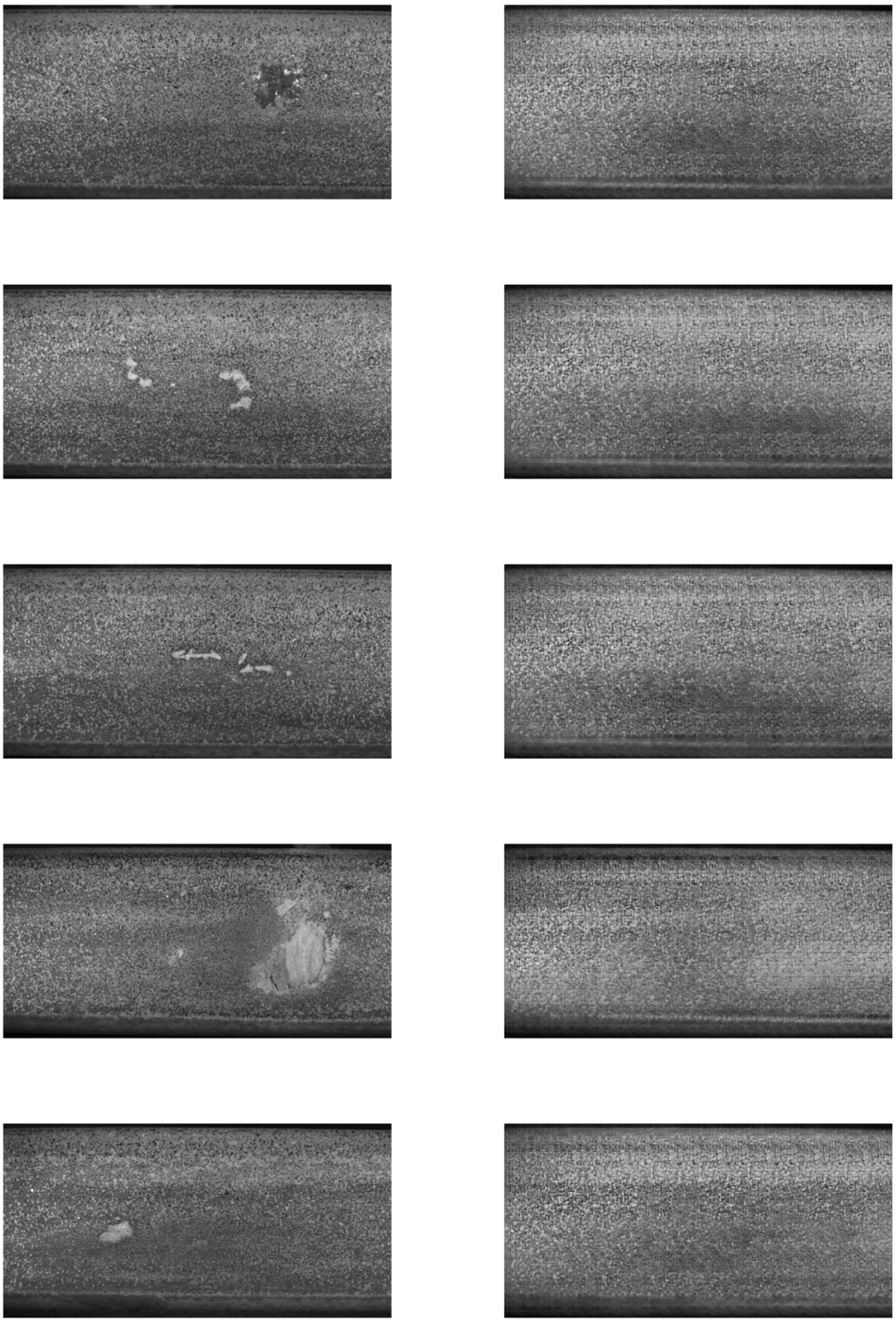


Figure B.11: Original (left column) and reconstructed (right column) images from the unhealthy test dataset at a resolution of 1024×512, with $M_c = 512$.

Appendix C

Additional Network Architectures

C.1 Model Architecture at Resolution 256×128

Table C.1: VAE architecture for images at 256×128 resolution.

Encoder θ	Activation Function	Output Shape w × h × c
Input: $\boldsymbol{x}$	-	$256 \times 128 \times 1$
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv	LReLU	$128 \times 64 \times 16$
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv	LReLU	$64 \times 32 \times 32$
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv	LReLU	$32 \times 16 \times 64$
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv	LReLU	$16 \times 8 \times 128$
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv	LReLU	$8 \times 4 \times 256$
Output: $\boldsymbol{\mu}$, $\boldsymbol{\sigma}^2$		
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv,	linear,	$4 \times 2 \times M_c$,
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv	linear	$4 \times 2 \times M_c$

Decoder θ	Activation Function	Output Shape w × h × c
Input: $\boldsymbol{z}$	-	$4 \times 2 \times M_c$
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv^T	LReLU	$8 \times 4 \times 256$
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv^T	LReLU	$16 \times 8 \times 128$
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv^T	LReLU	$32 \times 16 \times 64$
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv^T	LReLU	$64 \times 32 \times 32$
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv^T	LReLU	$128 \times 64 \times 16$
Output: $\boldsymbol{x}'$		
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv^T	tanh	$256 \times 128 \times 1$

Table C.2: Discriminator architecture for images at 256×128 resolution.

Discriminator ϕ	Activation Function	Output Shape w × h × c	f-layer
Input: $\boldsymbol{x}$ / $\boldsymbol{x}'$	-	$256 \times 128 \times 1$	-
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv	ReLU	$128 \times 64 \times 16$	f_6
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv	ReLU	$64 \times 32 \times 32$	f_5
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv	ReLU	$32 \times 16 \times 64$	f_4
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv	ReLU	$16 \times 8 \times 128$	f_3
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv	ReLU	$8 \times 4 \times 256$	f_2
$\substack{k=[3\times3]\\s=[2,2]}$ CoordConv	ReLU	$4 \times 2 \times 512$	f_1
Fully-connected	sigmoid	$1 \times 1 \times 256$	f_0
Output: $D(\boldsymbol{x})$ / $D(\boldsymbol{x}')$			
Fully-connected	sigmoid	$1 \times 1 \times 1$	-

C.2 Model Architecture at Resolution 512×256

Table C.3: VAE architecture for images at 512×256 resolution.

Encoder θ	Activation Function	Output Shape w × h × c
Input: $\boldsymbol{x}$	-	$512 \times 256 \times 1$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	LReLU	$256 \times 128 \times 8$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	LReLU	$128 \times 64 \times 16$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	LReLU	$64 \times 32 \times 32$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	LReLU	$32 \times 16 \times 64$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	LReLU	$16 \times 8 \times 128$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	LReLU	$8 \times 4 \times 256$
Output: $\boldsymbol{\mu}$, $\boldsymbol{\sigma}^2$ $^{k=[3\times3]}_{s=[2,2]}$ CoordConv, $^{k=[3\times3]}_{s=[2,2]}$ CoordConv	linear, linear	$4 \times 2 \times M_c$, $4 \times 2 \times M_c$

Decoder θ	Activation Function	Output Shape w × h × c
Input: $\boldsymbol{z}$	-	$4 \times 2 \times M_c$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv^T	LReLU	$8 \times 4 \times 256$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv^T	LReLU	$16 \times 8 \times 128$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv^T	LReLU	$32 \times 16 \times 64$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv^T	LReLU	$64 \times 32 \times 32$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv^T	LReLU	$128 \times 64 \times 16$
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv^T	LReLU	$256 \times 128 \times 8$
Output: $\boldsymbol{x}'$ $^{k=[3\times3]}_{s=[2,2]}$ CoordConv^T	tanh	$512 \times 256 \times 1$

Table C.4: Discriminator architecture for images at 512×256 resolution.

Discriminator ϕ	Activation Function	Output Shape w × h × c	f-layer
Input: $\boldsymbol{x}$ / $\boldsymbol{x}'$	-	$512 \times 256 \times 1$	-
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	ReLU	$256 \times 128 \times 8$	f_7
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	ReLU	$128 \times 64 \times 16$	f_6
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	ReLU	$64 \times 32 \times 32$	f_5
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	ReLU	$32 \times 16 \times 64$	f_4
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	ReLU	$16 \times 8 \times 128$	f_3
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	ReLU	$8 \times 4 \times 256$	f_2
$^{k=[3\times3]}_{s=[2,2]}$ CoordConv	ReLU	$4 \times 2 \times 512$	f_1
Fully-connected	sigmoid	$1 \times 1 \times 256$	f_0
Output: $D(\boldsymbol{x})$ / $D(\boldsymbol{x}')$ Fully-connected	sigmoid	$1 \times 1 \times 1$	-

Appendix D

Deep Learning Networks

D.1 Neural Networks

Artificial Neural Networks are a machine learning method which allows one to make predictions of an output given an input. A Neural Network (NN) can build a complex high-dimension non-linear mapping between inputs and outputs.

The logic behind a NN is fairly simple and intuitive. Figure D.1 represents a neural network which takes 3 input variables (input layer), the inputs then pass into a hidden layer containing 3 nodes, and then into the final layer which contains 2 outputs (binary classification). The connection from one node to the next is called an edge, which contains a weight. Each edge has a weight, which is initialised randomly, and is later updated to reduce the classification error. We use the training data as our inputs, which are mapped into outputs. The input variables multiplied by their weights are sent into each node of the hidden layer. This result then passes through a non-linear activation function to give a numerical value associated with each node in the hidden layer. These results are then multiplied by the weights connected to the output layer, to give the predicted output.

The process of passing the input variables through the hidden layers into the output is known as forward propagation. The predicted output is then compared to the actual known output to produce a scalar loss value for optimisation. Backpropagation calculates the gradient of the loss/error with respect to all the weights. The weights are then updated to reduce the error using the gradient information. The combination of forward propagation, backpropagation and weight updating is known as training the network. The weights are adjusted multiple times until the classification error becomes small. With the 'optimal weights' known, predictions can be made on new inputs simply through a forward pass (inference). In essence a NN is trained by adjusting the weights to reduce the prediction error.

This chapter contains a full description of NNs and CNNs for supervised classification. Note that the nomenclature in this chapter is not that same as the dissertation.

D.1.1 Network Architecture

When designing a NN, only the number of input and output nodes are known. The components left for the designer to choose is the type of activation function, the number of hidden nodes and the number of hidden layers [48]. Note that a NN is considered a shallow NN if only one hidden layer is used, if more then one hidden layer is used then we have a deep NN.

As mentioned deep learning implies a NN which has multiple hidden layers. These additional

layers assist in building a more complex function. Deep NNs achieved significantly higher classification ability than a shallow network, as deep architectures are able to learn a much higher level relationship between features [48]. Shallow networks are used when the input contains features which strongly correlate to the output.

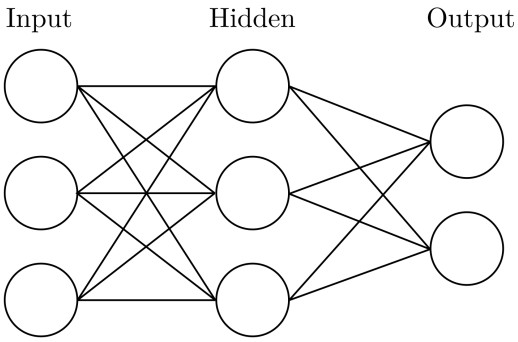


Figure D.1: Shallow neural network.

D.1.2 Activation Functions

The most commonly used activation functions are, the sigmoid, hyperbolic tan and rectified linear (ReLU) [72] activation function, which are expressed by Equation D.1, Equation D.2 and Equation D.3 respectively. These activation functions ensure non-linear mapping between layers. The derivative of these functions are also mathematically tractable, which allows for backpropagation to take place. Traditionally the sigmoid function has been used due to the historical concept of the behaviour of true neurons. However, more recently the ReLU function has gained popularity. This is due to the fact that a ReLU function does not suffer from the vanishing gradient problem unlike sigmoid functions. Another benefit is that a neuron can be turned off using the ReLU activation function. This allows non-linear sparse mapping to be accomplished between the input and output. Note leaky rectified linear unit (LReLU) [60] has also become popular. Equation D.4 expresses a LReLU function, with $leak = 0.01$.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{D.1}$$

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{D.2}$$

$$f(x) = \begin{cases} x & \text{if, } x > 0 \\ 0 & \text{else} \end{cases} \tag{D.3}$$

$$f(x) = \begin{cases} x & \text{if, } x > 0 \\ 0.01x & \text{else} \end{cases} \tag{D.4}$$

When there is more then two outputs (non-binary classification) in a NN, the softmax function is commonly used in the output layer. The softmax function is expressed by Equation D.5. The purpose of the softmax nonlinearity is to transform the outputs into normalised probabilities.

$$f(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}} \tag{D.5}$$

A virtual input of +1 is generally added to each layer in the network except the last (output layer). This virtual input is called the bias.

D.1.3 Forward Propagation

Forward propagation through a NN is used both during training of the NN as well as after training to make predictions on new unseen data. First we construct linear compilations of the input variables $x_1, ..., x_D$ [9] in the form:

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{D.6}$$

where $j = 1, ..., M$ and $x_0 = +1$ (bias).

There are M nodes in the first hidden layer (superscript of (1)) and w_{ji} are the weights. The activation's a_j are transformed using a non-linear activation function $f(\cdot)$ to give:

$$z_j = f(a_j) \tag{D.7}$$

z_j becomes the second hidden layer (or the output layer).

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j \tag{D.8}$$

where $k = 1, ..., K$ and $x_0 = +1$. There are K nodes in the second layer. If this layer is a hidden layer we would pass a_k through an additional non-linear hidden layer. Whereas, if it is the output layer we may pass a_k through a non-linear softmax function, Equation D.9.

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \text{softmax}\left[\sum_{j=0}^{M} w_{kj}^{(2)} f\Big(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\Big)\right] \tag{D.9}$$

This feed forward model is simply a non-linear function from a set of input variables x_i to a set of output variable y_k controlled by a vector $\boldsymbol{w}$ of adjustable parameters [9].

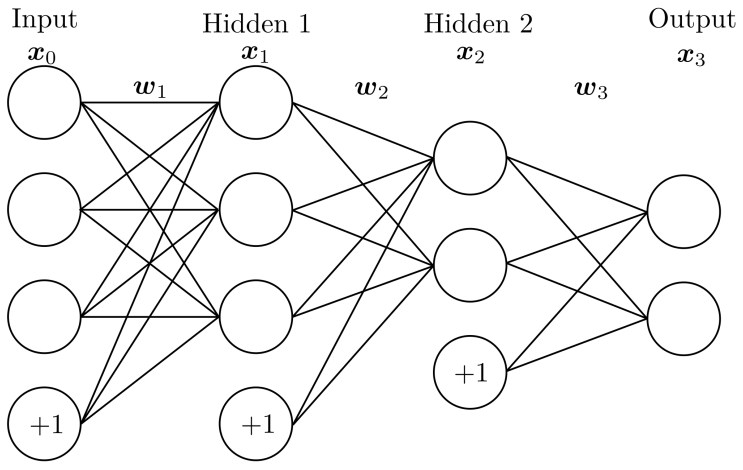


Figure D.2: Two layer (deep) neural network.

D.1.4 Backward Propagation

Backpropagation is performed by measuring the error of the final result, and then moving backward in your network towards the inputs, to determine an update rule to adjust the weights to

minimise the loss. The mean squared error (MSE) is expressed by Equation D.10, where x_3 is the prediction and t is the known correct outcome.

Back propagation equations are derived by repeatedly applying the chain rule. We start at the last layer (final outputs), and move backward. Considering the NN in Figure D.2 one would perform the following calculations:

$$\mathcal{L} = \frac{1}{2}(\boldsymbol{x}_3 - \boldsymbol{t})^2 \tag{D.10}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_3} = (\boldsymbol{x}_3 - \boldsymbol{t})\frac{\partial \boldsymbol{x}_3}{\partial \boldsymbol{w}_3} = [(\boldsymbol{x}_3 - \boldsymbol{t}) \odot f_3'(\boldsymbol{w}_3\boldsymbol{x}_2)]\frac{\partial(\boldsymbol{w}_3\boldsymbol{x}_2)}{\partial \boldsymbol{w}_3} = [(\boldsymbol{x}_3 - \boldsymbol{t}) \odot f_3'(\boldsymbol{w}_3\boldsymbol{x}_2)]\boldsymbol{x}_2^T$$

where $\odot$ indicates element-wise matrix multiplication. Let:

$$\boldsymbol{\delta}_3 = (\boldsymbol{x}_3 - \boldsymbol{t}) \odot f_3'(\boldsymbol{w}_3\boldsymbol{x}_2)$$

$$\therefore \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_3} = \boldsymbol{\delta}_3\boldsymbol{x}_2^T$$

One then moves on to the second hidden layer:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_2} = (\boldsymbol{x}_3 - \boldsymbol{t})\frac{\partial \boldsymbol{x}_3}{\partial \boldsymbol{w}_2} = [(\boldsymbol{x}_3 - \boldsymbol{t}) \odot f_3'(\boldsymbol{w}_3\boldsymbol{x}_2)]\frac{\partial(\boldsymbol{w}_3\boldsymbol{x}_2)}{\partial \boldsymbol{w}_2} = \boldsymbol{\delta}_3\frac{\partial(\boldsymbol{w}_3\boldsymbol{x}_2)}{\partial \boldsymbol{w}_2}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_2} = \boldsymbol{w}_3^T\boldsymbol{\delta}_3\frac{\partial(\boldsymbol{x}_2)}{\partial \boldsymbol{w}_2} = [\boldsymbol{w}_3^T\boldsymbol{\delta}_3 \odot f_2'(\boldsymbol{w}_2\boldsymbol{x}_1)]\frac{\partial(\boldsymbol{w}_2\boldsymbol{x}_1)}{\partial \boldsymbol{w}_2} = [\boldsymbol{w}_3^T\boldsymbol{\delta}_3 \odot f_2'(\boldsymbol{w}_2\boldsymbol{x}_1)]\boldsymbol{x}_1^T$$

$$\therefore \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_2} = \boldsymbol{\delta}_2\boldsymbol{x}_1^T$$

Then onto the first hidden layer:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_1} = [\boldsymbol{w}_2^T\boldsymbol{\delta}_2 \odot f_1'(\boldsymbol{w}_1\boldsymbol{x}_0)]\boldsymbol{x}_0^T = \boldsymbol{\delta}_1\boldsymbol{x}_0^T$$

As a general formula one can use Equation D.11 and Equation D.12.

$$\boldsymbol{\delta}_L = (\boldsymbol{x}_L - \boldsymbol{t}) \odot f_L'(\boldsymbol{w}_L\boldsymbol{x}_{L-1}) \tag{D.11}$$

$$\boldsymbol{\delta}_i = \boldsymbol{w}_{i+1}^T\boldsymbol{\delta}_{i+1} \odot f_i'(\boldsymbol{w}_i\boldsymbol{x}_{i-1}) \tag{D.12}$$

with the subscript L referring to the output layer.

D.1.5 Updating Weights

Backpropagation allows one to easily calculate the gradient of the error for each set of weights using Equation D.13.

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_i} = \boldsymbol{\delta}_i\boldsymbol{x}_{i-1}^T \tag{D.13}$$

In order to minimise the loss function (Equation D.10), we update the weights in the negative direction of the gradient loss function with respect to those weights (gradient descent). This is performed by Equation D.15, using the updated step of Equation D.14.

$$\Delta\boldsymbol{w}_i = -\eta\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_i} \tag{D.14}$$

$$\boldsymbol{w}_i = \boldsymbol{w}_i + \Delta \boldsymbol{w}_i \tag{D.15}$$

Where η is called the learning rate. With large datasets the weights are usually updated using a mini-batch approach. This implies the weights are updated after a certain batch of the data has been passed through the network. Up dating the weights using the entire dataset is called an epoch. Multiple epoches are used during training.

The above update rule is based on simple gradient descent. There exists several algorithms which use the gradient information to update the weights in a more efficient manor, one of the most popular training algorithms is Adam [44].

D.2 Convolutional Neural Networks

In 1998 a research paper [49] by Yann LeCun and Lon Bottou introduced the CNN. A typical layer of a CNN consists of 3 stages:

1. Convolution stage: This stage performs several convolutions in parallel to produce a set of linear activations
2. Detector stage: Each linear activation is then passed through a nonlinear activation function.
3. Pooling stage: This stage modifies the output further.

The combination of these 3 stages is called a convolution layer. A shallow CNN contains only one layer and a deep CNN contains more than one. A fully connected layer (simple NN) follows after the last convolution layer. This then leads to the output layer.

D.2.1 Convolutional Stage

A convolution function $(*)$ is expressed by Equation D.16, with a kernel of size $m \times n$ (the weights) and where i and j is the portion of the image where the convolution takes place. The convolution is performed over an $m \times n$ region of the image. Further convolutions are performed over the entire image by sliding the kernel around the image.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n) \tag{D.16}$$

The purpose of the convolution stage is to extract features from the input space. Each kernel can be thought of as feature identifiers. Multiple kernels can be used in one convolution stage to extract multiple features. Each kernel has numerical values (weights) which are initially set to be random, and then are adjusted such that useful features are learned to reduce the classification error.

D.2.2 Pooling Stage

The primary purpose of the pooling stage is to reduce the spatial dimensions of the input array. For this reason this stage is commonly also known as down-sampling. Much like the convolution stage, in pooling a sliding window moves across the input transforming the values into a summarised form of the input. For example the max pooling function will return the maximum output within a rectangular area. This rectangular area shifts along the whole image to give a summary of the results.

Pooling helps make the representation become approximately invariant to translations in the

input [32]. Invariance to local translation can be very useful for when one cares more about whether a feature exits rather then the exact location of the feature. By reporting summary statistics, pooling reduces the input size to the next convolution layer or fully connected layer. The reduction in dimensional drastically improves the computational efficiency.

D.2.3 Fully Connected Layer

The final layer in a CNN is a fully connected layer. This layer is just simply a NN, deep or shallow. The input to the NN is the output of the last convolutional layer, and the output of the NN is the predicted results. Again a softmax function is used for multiclass classification.

In essence, the convolutional layers learn features which are then sent to the fully connected layer which performs the classification based on the incoming features. A NN is trained using backpropagation to adjusting the weights which connected the layers. A CNN works almost identically, again backpropagation is used to adjust the weights in the fully connected layer, and the weights in the kernels are also adjusted through backpropagation. Therefore during training optimal weights are learnt for the kernels so that useful features are extracted, and optimal weights are learnt in the fully connected layer so that accurate classification can be performed.

Bibliography

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

[2] C. Alippi, E. Casagrande, F. Scotti, and V. Piuri. Composite real-time image processing for railways track profile measurement. *IEEE Transactions on Instrumentation and Measurement*, 49(3):559–564, 2000.

[3] J. An and S. Cho. Variational Autoencoder based Anomaly Detection using Reconstruction Probability. *Technical Report*, pages 1–18, 2015.

[4] R. Andersson. Surface defects in rails. *RailCorp Engineering Manual Track Surface Defects in Rails*, 1.2:30, 2013.

[5] M.N. Bassim, S.St. Lawrence, and C.D. Liu. Detection of the onset of fatigue crack growth in rail steels using acoustic emission. *Engineering Fracture Mechanics*, 47(2):207–214, 1994.

[6] Y. Bengio. Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning*, 2:1–127, 2009.

[7] M. Bentoumi, P. Aknine, and G. Bloch. On-line rail defect diagnosis with differential eddy current probes and specific detection processing. *The European Physical Journal - Applied Physics*, 23(3):227–233, 2003.

[8] P. Bergmann, S. Löwe, M. Fauser, D. Sattlegger, and C. Steger. Improving Unsupervised Defect Segmentation by Applying Structural Similarity to Autoencoders. 2018. `http://arxiv.org/abs/1807.02011`.

[9] C. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[10] Brite/Euram. Integrated Study of Rolling Contact Fatigue (ICON) European Commission DGX111 Brite/Euram III Project). *Contract BRPR-CT96-0245 Project Programme, Brussels*, 1997-1999.

[11] A. Brock, J. Donahue, and K. Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. http://arxiv.org/abs/1809.11096, 2018.

[12] D.F. Cannon, K. Edel, S.L. Grassie, and K. Sawley. Rail defects: an overview. *Fatigue & Fracture of Engineering Materials & Structures*, 26:865–886, 2003.

[13] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. 2016. `http://arxiv.org/abs/1606.03657`.

[14] G. Cottrell, P. Munro, and D. Zipser. Learning internal representations from gray-scale images: An example of extensional programming. *In Conference of the Cognitive Science Society*, 1987.

[15] A. Creswell and A.A. Bharath. Inverting The Generator Of A Generative Adversarial Network (II). (Section VII), 2018. `http://arxiv.org/abs/1802.05701`.

[16] R. Dahl, M. Norouzi, and J. Shlens. Pixel recursive super resolution. 2017. `http://arxiv.org/abs/1702.00783`.

[17] J. Dearden. Rail failures on British railways: incidence, and preventive measures. *Railway Gazette*, (113):509–512, 1957.

[18] J. Dearden. Rail failures on British railways: reporting, analysis and prevention. *Railway Gazette*, (121):148–150, 1965.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. 2009.

[20] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial Feature Learning. *International Conference on Learning Representations*, (2017), 2017. `http://arxiv.org/abs/1605.09782`.

[21] A. Dosovitskiy and T. Brox. Generating Images with Perceptual Similarity Metrics based on Deep Networks. 2016. `http://arxiv.org/abs/1602.02644`.

[22] A.K. Dubey and Z.A. Jaffery. Maximally Stable Extremal Region Marking-Based Railway Track Surface Defect Sensing. *IEEE Sensors Journal*, 16(24):9047–9052, 2016.

[23] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville. Adversarial Learned Inference. *International Conference on Learning Representations*, (2017), 2017.

[24] S. Faghih-Roohi, S. Hajizadeh, A. Núñez, R. Babuska, and B. Schutter. Deep convolutional neural networks for detection of rail surface defects. *2016 International Joint Conference on Neural Networks (IJCNN)*, 19:2584–2589, 2016.

[25] H. Feng, Z. Jiang, F. Xie, P. Yang, J. Shi, and L. Chen. Automatic fastener classification and defect detection in vision-based railway inspection systems. *IEEE Transactions on Instrumentation and Measurement*, 63(4):877–888, 2014.

[26] J. Gan, Q. Li, J. Wang, and H. Yu. A Hierarchical Extractor-Based Visual Rail Surface Inspection System. *IEEE Sensors Journal*, 17(23):7935–7944, 2017.

[27] J. Gan, J. Wang, H. Yu, Q. Li, and Z. Shi. Online Rail Surface Inspection Utilizing Spatial Consistency and Continuity. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pages 1–11, 2018.

[28] X. Gibert, V.M. Patel, and R. Chellappa. Material classification and semantic segmentation of railway track images with deep convolutional neural networks. *IEEE International Conference on Image Processing (ICIP)*, pages 621–625, 2015.

[29] X. Gibert, V.M. Patel, and R. Chellappa. Robust fastener detection for autonomous visual railway track inspection. *Proceedings - 2015 IEEE Winter Conference on Applications of Computer Vision, WACV 2015*, pages 694–701, 2015.

[30] X. Gibert, V.M. Patel, and R. Chellappa. Deep Multitask Learning for Railway Track Inspection. *IEEE Transactions on Intelligent Transportation Systems*, 18(1):153–164, 2017.

[31] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 9, 2010.

[32] I. Goodfellow, Y Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[33] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. *Advances in Neural Information Processing Systems 27*, pages 2672–2680, 2014.

[34] S. Hajizadeh, A. Núñez, and D.M.J. Tax. Semi-supervised Rail Defect Detection from Imbalanced Image Data. *IFAC-PapersOnLine*, 49(3):78–83, 2016.

[35] M. Hardt, B. Recht, and Y. Singer. Train faster, generalize better: Stability of stochastic gradient descent. 2015. `http://arxiv.org/abs/1509.01240`.

[36] Z. He, Y. Wang, F. Yin, and J. Liu. Surface defect detection for high-speed rails using an inverse P-M diffusion model. *Sensor Review*, 36(1):86–97, 2016.

[37] J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, Md.M.A. Patwary, Y. Yang, and Y. Zhou. Deep Learning Scaling is Predictable, Empirically. pages 1–19, 2017. `http://arxiv.org/abs/1712.00409`.

[38] European Railway Research Institute. Rail Defect ManagementState of the Art. *Utrecht, the Netherlands: European Railway Research Institute. Report D229/RP2 (2nd edn)*, 2000.

[39] A. Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard GAN. 2018. `http://arxiv.org/abs/1807.00734`.

[40] M. Karakose, O. Yaman, M. Baygin, K. Murat, and E. Akin. A new computer vision based method for rail track detection and fault diagnosis in railways. *International Journal of Mechanical Engineering and Robotics Research*, 6(1):22–27, 2017.

[41] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. 2017. `http://arxiv.org/abs/1710.10196`.

[42] A.M. Kaynia, J. Park, and K. Norén-Cosgriff. Effect of track defects on vibration from high speed train. *Procedia Engineering*, 199:2681–2686, 2017.

[43] S. Khan and T. Yairi. A review on the application of deep learning in system health management. *Mechanical Systems and Signal Processing*, 107:241–265, 2018.

[44] D.P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. 2014. `http://arxiv.org/abs/1412.6980`.

[45] D.P. Kingma and M. Welling. Auto-Encoding Variational Bayes. (Ml), 2013. `http://arxiv.org/abs/1312.6114`.

[46] A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012.

[47] A.B.L. Larsen, S.K. Sønderby, H. Larochelle, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. 2015. `http://arxiv.org/abs/1512.09300`.

[48] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. J. Huang. A Tutorial on EnergyBased Learning. *Predicting Structured Data*, pages 191–246, 2006.

[49] Y. LeCun and C. Cortes. MNIST handwritten digit database. 1998. `http://yann.lecun.com/exdb/mnist/`.

[50] Q. Li and S. Ren. A real-time visual inspection system for discrete surface defects of rail heads. *IEEE Transactions on Instrumentation and Measurement*, 61(8):2189–2199, 2012.

[51] Q. Li and S. Ren. A Visual Detection System for Rail Surface Defects. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1531–1542, 2012.

[52] Q. Li, Y. Tan, Z. Huayan, S. Ren, P. Dai, and W. Li. A Visual Inspection System for Rail Corrugation Based on Local Frequency Features. *Proceedings - 2016 IEEE 14th International Conference on Dependable, Autonomic and Secure Computing, DASC 2016, 2016 IEEE 14th International Conference on Pervasive Intelligence and Computing, PICom 2016, 2016 IEEE 2nd International Conference on Big Data Intelligence and Computing, DataCom 2016 and 2016 IEEE Cyber Science and Technology Congress, CyberSciTech 2016, DASC-PICom-DataCom-CyberSciTech 2016*, pages 18–23, 2016.

[53] X. Li, B. Gao, W.L. Woo, G.Y. Tian, X. Qiu, and L. Gu. Quantitative Surface Crack Evaluation Based on Eddy Current Pulsed Thermography. *IEEE Sensors Journal*, 17(2):412–421, 2017.

[54] J. Lin, S. Luo, Q. Li, H. Zhang, and S. Ren. Real-time rail head surface defect detection: A geometrical approach. *IEEE International Symposium on Industrial Electronics*, (ISlE):769–774, 2009.

[55] R. Liu, J. Lehman, P. Molino, F.P. Such, E. Frank, A. Sergeev, and J. Yosinski. An Intriguing Failing of Convolutional Neural Networks and the CoordConv Solution. 2018. `http://arxiv.org/abs/1807.03247`.

[56] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. 2015.

[57] Z. Liu, W. Wang, X. Zhang, and W. Jia. Inspection of rail surface defects based on image processing. *CAR 2010 - 2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics*, 1:472–475, 2010.

[58] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[59] J. Luo, Y. Xu, C. Tang, and J. Lv. Learning Inverse Mapping by Auto-Encoder Based Generative Adversarial Nets. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10635 LNCS:207–216, 2017.

[60] A.L. Maas, A.Y. Hannun, and A.Y. Ng. Rectifier nonlinearities improve neural network acoustic models. *Proceedings of the 30 th International Conference on Machine Learning*, 28:6, 2013.

[61] E. Magel. Rolling Contact Fatigue: A Comprehensive Review. *U.S. Department of Transportation Federal Railroad Administration*, (November):118, 2011.

[62] E. Magel, P. Sroba, K. Sawley, and J. Kalousek. Control of Rolling Contact Fatigue of Rails. *In Proceedings the AREMA 2004 Annual Conference, Nashville, TN, USA*, 2004.

[63] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. 2014. `http://arxiv.org/abs/1412.0035`.

[64] C. Mandriota, M. Nitti, N. Ancona, E. Stella, and A. Distante. Filter-based feature selection for rail defect detection. *Machine Vision and Applications*, 15(4):179–185, 2004.

[65] X. Mao, Q. Li, H. Xie, R.Y.K. Lau, Z. Wang, and S.P. Smolley. Least Squares Generative Adversarial Networks. 2016. `http://arxiv.org/abs/1611.04076`.

[66] F. Marino, A. Distante, P.L. Mazzeo, and E. Stella. A real-time visual inspection system for railway maintenance: Automatic Hexagonal-Headed Bolts Detection. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(3):418–428, 2007.

[67] G. Matheron. *Éléments pour une théorie des milieux poreux.* Paris, Masson et Cie, 1967.

[68] T. Matsubara, R. Tachibana, and K. Uehara. Anomaly Machine Component Detection by Deep Generative Model with Unregularized Score. *International Joint Conference on Neural Networks*, 2018.

[69] Y. Min, B. Xiao, J. Dang, B. Yue, and T. Cheng. Real time detection system for rail surface defects based on machine vision. *Eurasip Journal on Image and Video Processing*, 2018(1):1–11, 2018.

[70] L.F. Molina, E. Resendiz, J.R. Edwards, J.M. Hart, C.P.L. Barkan, and N. Ahuja. Condition Monitoring of Railway Turnouts and Other Track Components Using Machine Vision. *Proceedings of the Transportation Research Board 90th Annual Meeting*, (217):1–17, 2010.

[71] M. Molodova, Z. Li, A. Núñez, and D. Rolf. Automatic Detection of Squats in Railway Infrastructure. *InnoTrans 2014 Report*, 18(1):1980–1990, 2014.

[72] V. Nair and G.E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*, (3):807–814, 2010.

[73] J.F. Nash. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36:48–49, 1950.

[74] A. Odena, C. Olah, and J. Shlens. Conditional Image Synthesis With Auxiliary Classifier GANs. *Proceedings of the 34th International Conference on Machine Learning*, 70, 2017.

[75] M. Papaelias, C. Roberts, and C. Davis. A review on non-destructive evaluation of rails: state-of-the-art and future development. *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, 222:367–384, 2008.

[76] A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. 2015. `http://arxiv.org/abs/1511.06434`.

[77] D.J. Rezende, S. Mohamed, and D. Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 32, 2014. `http://arxiv.org/abs/1401.4082`.

[78] V. Rikhotso, N. Steyn, and Y. Hamam. 3D rail modelling and measurement for rail profile condition assessment. *2017 IEEE AFRICON: Science, Technology and Innovation for Africa*, pages 1522–1527, 2017.

[79] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved Techniques for Training GANs. 2016. `http://arxiv.org/abs/1606.03498`.

[80] B. Sambo, A. Bevan, and C. Pislaru. A novel application of image processing for the detection of rail surface RCF damage and incorporation in a crack growth model. *International Conference on Railway Engineering (ICRE 2016)*, pages 12 (9 .)–12 (9 .), 2016.

[81] Y. Santur, M. Karaköse, and E. Akin. A new rail inspection method based on deep learning using laser cameras. *Artificial Intelligence and Data Processing Symposium*, 2017.

[82] T. Schlegl, P. Seeböck, S.M. Waldstein, U. Schmidt-Erfurth, and G. Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. *Information Processing in Medical Imaging. IPMI 2017. Lecture Notes in Computer Science*, 10265, 2017.

[83] M. Seitzer, G. Yang, J. Schlemper, O. Oktay, T. Wuerfl, V. Christlein, T. Wong, R. Mohiaddin, D. Firmin, and J. Keegan. Adversarial and Perceptual Refinement for Compressed Sensing MRI Reconstruction. *Medical Image Computing and Computer Assisted Intervention (MICCAI 2018)*, 2018.

[84] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. 2014. `http://arxiv.org/abs/1409.1556`.

[85] J.T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for Simplicity: The All Convolutional Net. 2014. `http://arxiv.org/abs/1412.6806`.

[86] M.A. Sutton, F. Matta, D. Rizos, R. Ghorbani, S. Rajan, D.H. Mollenhauer, H.W. Schreier, and A.O. Lasprilla. Recent Progress in Digital Image Correlation: Background and Developments since the 2013 W M Murray Lecture. *Experimental Mechanics*, 57(1):1–30, 2017.

[87] C. Taştimur, M. Karakose, E. Akin, and I. Aydin. Rail defect detection with real time image processing technique. *IEEE International Conference on Industrial Informatics (INDIN)*, pages 411–415, 2017.

[88] V.R. Vijaykumar and S. Sangamithirai. Rail Defect Detection using Gabor filters with Texture Analysis. *International Conference on Signal Processing, Communication and Networking (ICSCN)*, 3, 2015.

[89] L. Wang, L. Zhuang, and Z. Zhang. Automatic Detection of Rail Surface Cracks with a Superpixel-Based Data-Driven Framework. *Journal of Computing in Civil Engineering*, 33(1):1–9, 2019.

[90] Z. Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[91] H. Yu, Q. Li, Y. Tan, J. Gan, J. Wang, Y. Geng, and L. Jia. A Coarse-to-Fine Model for Rail Surface Defect Detection. *IEEE Transactions on Instrumentation and Measurement*, PP:1–11, 2018.

[92] H. Zenati, C.S. Foo, B. Lecouat, G. Manek, and V.R. Chandrasekhar. Efficient GAN-Based Anomaly Detection. *International Conference on Learning Representations*, (2018):1–7, 2018.

[93] H. Zhang, X. Jin, Q.M.J. Wu, Y. Wang, Z. He, and Y. Yang. Automatic Visual Detection System of Railway Surface Defects With Curvature Filter and Improved Gaussian Mixture Model. *IEEE Transactions on Instrumentation and Measurement*, pages 1–16, 2018.

[94] L. Zhang, L. Zhang, X. Mou, and D. Zhang. A comprehensive evaluation of full reference image quality assessment algorithms. *Proceedings - International Conference on Image Processing, ICIP*, pages 1477–1480, 2012.

[95] X. Zhang, Y. Cui, Y. Wang, M. Sun, and H. Hu. An improved AE detection method of rail defect based on multi-level ANC with VSS-LMS. *Mechanical Systems and Signal Processing*, 99:420–433, 2018.

[96] X. Zhang, N. Feng, Y. Wang, and Y. Shen. Acoustic emission detection of rail defect based on wavelet transform and Shannon entropy. *Journal of Sound and Vibration*, 339:419–432, 2015.

[97] L. Zhuang, L. Wang, Z. Zhang, and K.L. Tsui. Automated vision inspection of rail surface cracks: A double-layer data-driven framework. *Transportation Research Part C: Emerging Technologies*, 92(September 2017):258–277, 2018.