# Artificial Intelligence & the steel industry: Developing an automated quality inspection system

*Author:*
Anastasia PARAMORE
BSc MSc

*Supervisors:*
Dr. Xianghua XIE
Dr. Andrew C. TAPPENDEN
Dr. Simon G. LEWIS

*A thesis submitted to Swansea University in fulfilment of the requirements for the degree of Doctor of Engineering*

SWANSEA UNIVERSITY
August 23, 2021

This work is funded by the European Social Fund (ESF) through the European Union's Convergence programme administered by the Welsh Government.

# Abstract

Anastasia PARAMORE

*Artificial Intelligence & the steel industry: Developing an automated quality inspection system*

This thesis explores possible improvements to the existing defect detection systems used by Tata Steel Europe in South Wales. Exploring the possibilities of accuracy increases and further uses for in-house surface quality systems presents Tata Steel Europe with methods and considerations on how to improve their current surface defect management tools. This research considers three different aspects of the defect management process and tests alternatives to the systems either currently used or not yet in place.

The first section considers an ensemble object detection method using Gabor filters, histograms, and a random forest classifier. The algorithm was applied originally as an ensemble method and then as a case-study it was split into two combinations of the methods to determine what level of ensemble complexity was optimal. The images used were a set of various types of steel defect images provided by Tata Steel Europe. The ensemble method achieved acceptable results compared to the existing systems, but the medium-complexity method was optimal regarding overall accuracy, false negative rate, and speed. The second section used a set of weld hole images which were split into three quality grades by defining characteristics of the feature. These images were used to test three different neural networks, an R-CNN version of GoogLeNet, and Faster R-CNN versions of ResNet-50 and ResNet-101. These networks were tasked with classifying the quality of weld holes in steel. Accurately detecting weld holes is vital in steel production as certain production processes require speed changes for welds. As the hole punches degrade over time, so does the quality of the weld hole. When weld holes are of poor quality, they can be missed or wrongly detected. All three networks detected the weld holes very well, but classifying the quality grade was approximately 60% accurate for both ResNets and 79% accurate for the GoogLeNet. These tests highlighted the importance of data quantity and quality, including lack of bias in data. The third section looks at how colour filters and greyscale methods can affect the images used for detection and classification. An investigation into coloured light sources has not been fully explored at Tata Steel in South Wales before. Having worked with the image data for sections one and two, it highlighted how important the quality of these images is. Steel defect samples were supplied, alongside their lab-confirmed defect label. Scans were taken of clean and oiled steel samples with different coloured filters and a variety of common greyscale methods were used to turn these images from RGB colour to greyscale images. The greyscale values of defect to clean steel were calculated for a specific region of interest on each steel sample. This value was used as a measure of contrast between clean and defect steel. The yellow filter with the Decolorize method produced the highest contrast image, higher than using no filter at all. For oiled samples, using no filter with the Decolorize method produced the best contrast and the orange filter with the Decolorize method produced the best contrast out of the filtered images. The greyscale methods used had significant effect on the contrast of the image.

The significance of this thesis is that it informs of the difficulties in developing surface inspection of steel defects due to several factors, such as environment and variation of defect shape, size, colour, and frequency. The work undertaken in this study has highlighted the need for a larger pilot line to further research both the capturing of the perfect image, and finding the most accurate detection and classification methods for each grade of steel.

# Declaration of Authorship

I, Anastasia PARAMORE, declare that this thesis titled, “Artificial Intelligence & the steel industry: Developing an automated quality inspection system” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: [redacted]

Date: 23/08/2021

# Acknowledgements

There are several people who I would like to thank for their support during the creation of this piece of work - whether it be financially, academically, or emotionally. I would like to thank the M2A team for all of their support at Swansea University during the last few years. I would like to thank my academic supervisors, Dr. Andrew C. Tappenden and Dr. Xianghua Xie, for all of their help and guidance throughout this project. I would like to thank Tata Steel Europe, the technical team at Trostre, and my industrial supervisor Dr. Simon Lewis, who has been a great mentor during this process and a supportive manager during my time at the company. I would like to thank the rest of the surface inspection department in Port Talbot for all of their insight, and support, and for welcoming me into their team.

Finally, I would like to thank my partner, Wolfram, for inspiring me, for always believing in me, for his continued support, and for his uplifting advice.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **AI** | **A**rtificial **I**ntelligence |
| **CAPL** | **C**ontinuous **A**nnealing **P**rocessing **L**ine |
| **CCD** | **C**harge-**C**oupled **D**evice |
| **CIE** | **C**ommitte **I**nternational de l′**E**clairage |
| **CNN** | **C**onvolutional **N**eural **N**etwork |
| **FN** | **F**alse **N**egative |
| **FNR** | **F**alse **N**egative **R**ate |
| **FP** | **F**alse **P**ositive |
| **FPR** | **F**alse **P**ositive **R**ate |
| **GPU** | **G**raphics **P**rocessing **U**nit |
| **HSL** | **H**ue **S**aturation **L**ightness |
| **HSV** | **H**ue **S**aturation **V**alue |
| **LED** | **L**ight **E**mitting **D**iode |
| **R-CNN** | **R**egional - **C**onvolutional **N**eural **N**etwork |
| **RGB** | **R**ed **G**reen **B**lue |
| **RID** | **R**olled **I**n **D**ebris |
| **ROI** | **R**egion **O**f **I**nterest |
| **SVM** | **S**upport **V**ector **M**achine |
| **TLR** | **T**ata **L**ogical **R**elease |
| **TN** | **T**rue **N**egative |
| **TNR** | **T**rue **N**egative **R**ate |
| **TNS** | **T**otal **N**umber of **S**amples |
| **TP** | **T**rue **P**ositive |
| **TPR** | **T**rue **P**ositive **R**ate |

# Nomenclature

| | |
|---|---|
| $(x, y)$ | The usual spatial coordinates in 2-D |
| $(x, y, z)$ | The usual spatial coordinates in 3-D |
| $[i, j]$ | Singular pixel location, with respect to the usual spatial coordinates |
| $\alpha, \beta$ | Sharpness of the Gaussian along major and minor axis, parallel and perpendicular to the wave, respectively |
| $\eta$ | The learning rate used when training a network |
| exp | The exponential constant |
| $\gamma, \zeta$ | The spatial aspect ratios for major and minor axis, respectively |
| $\imath$ | Represents an imaginary number |
| $\kappa / \mathcal{K}$ | Denotes the $\kappa$-th class / the total number of classes |
| $\lambda$ | Wavelength |
| $\mathcal{F}$ | Frequency |
| $\mathcal{G}_{name}$ | RGB to greyscale method of that particular name |
| $\mu_X(\lambda), \mu_Y(\lambda), \mu_Z(\lambda)$ | The CIE colour matching functions |
| $\nabla^2 f$ | Laplacian of a function $f$ |
| $\pi$ | The mathematical constant, 3.1415... |
| $\psi()$ | Energy transform applied to Gabor filter responses |
| $\sigma$ | The spread of the Gaussian |
| $\sigma(z)$ | Where sigma is used as a function, it represents an activation function for a neuron in a neural network |
| $\sigma_s$ | Sigmoid function, producing the ouput for a sigmoid neuron |
| $\mathbf{w}$ | A vector of weights for a neural network |
| $\mathbf{x}$ | A vector of neuron inputs |

| | |
|---|---|
| $\theta[i,j]$ | Array of orientations of the gradient for image $I[i,j]$ |
| $\theta$ | The anti-clockwise rotation of the Gaussian and the plane wave |
| $\varepsilon(x,y)$ | Averaging convolution |
| $a$ | Neuron outputs |
| $a_k$ | Neuron outputs to be aimed for, e.g. the known classes |
| $b$ | The bias of a neuron, ie. how easily it will fire |
| $C$ | Generic undefined cost function used for examples |
| $C(\lambda)$ | Amount of photons at wavelength $\lambda$ |
| $C_{quad}$ | The quadratic cost function |
| $d$ | Maximum deviance reduction method |
| $f(K)$ | A function of Luminance used in $\mathcal{G}_{Lightness}$ calculation |
| $f(x,y)$ | Generic function used for examples |
| $f[i,j]$ | Generic function used for examples |
| $F_{g_1}, F_{g_2}$ | Gaussian filter in one and two dimensions |
| $F_{Gabor}$ | Gabor filter |
| $f_{illumination}$ | The amount of source illumination on the scene being imaged |
| $F_{mid}$ | Mid-point filter |
| $F_{min}, F_{max}$ | Minimum and maximum filters |
| $f_{reflected}$ | The amount of illumination reflected by objects in a scene |
| $g$ | Grey value |
| $G_f[i,j,\sigma]$ | Gaussian filter with inputs of pixel locations $i,j$ and spread control $\sigma$ |
| $G_x/G_y$ | Gradient between grey values in the x-direction/in the y-direction |
| $Gini(T)$ | Gini's diversity index |
| $h(x,y)$ | Laplacian of Gaussian |
| $I(x,y)$ | A 2-D greyscale image |
| $I_f$ | Filtered image array |

| | |
|---|---|
| $K$ | Luminance value, not to be confused with *Luminance* the greyscale method |
| $k$ | Number of bits in an image |
| $L$ | Number of grey levels in an image |
| $l, L$ | The $l$-th layer out of the total number of layers $L$ |
| $M[i, j]$ | Array of magnitudes of the gradient for image $I[i, j]$ |
| $M_{Roberts}$ | Roberts operator used to calculate the magnitude of gradient |
| $M_{Roberts}[f[i, j]]$ | Roberts operator used to calculate the magnitude of gradient at specific point $[i, j]$ of function $f$ |
| $M_{Sobel}$ | Using Sobel operator to calculate the magnitude of gradient |
| $M_{sup}[i, j]$ | Array of surpressed magnitudes of the gradient for image $I[i, j]$ |
| $n \times m$ | The total number of pixels in an image in the x and y directions, representing its size |
| $n$ | Total number of items, e.g $n$ number of inputs |
| $p(\kappa)$ | The proportion of classes of class $\kappa$ that reach a node |
| $p(L(\kappa))$ | The proportion of classes of class $\kappa$ that reach the left node |
| $p(R(\kappa))$ | The proportion of classes of class $\kappa$ that reach the right node |
| $P[i, j], Q[i, j]$ | Array of partial derivatives of a filtered image $I_f$ |
| $s_x, s_y$ | Partial derivatives in the $x$ and $y$ directions |
| $T$ | The training data set |
| $t$ | The constant used in the energy transform |
| $w_1, ..., w_n$ | One to $n$ number of neuron weights for a neural network |
| $w_j$ | The $j$-th neuron weight |
| $W_{x,y}$ | Averaging window |
| $X, Y, Z$ | Scalars used to calculate the $(x, y, z)$ coordinates in the CIE chromaticity diagram |
| $X, Y$ | Generic inputs and outputs |
| $x_1, ..., x_n$ | One to $n$ number of neuron inputs for a neural network |

| | |
|---|---|
| $X_j$ | $\in X$, the $j$-th input of a set $X$ of inputs which are a proper subset of $\mathbf{x}$ neuron inputs for a neural network |
| $x_j$ | The $j$-th neuron input |
| $z$ | defines the relationship between weights, inputs and bias that goes into an activation function |
| a* | A measure on the red-green scale |
| B | Blue value from an RGB colour |
| B′ | The gamma corrected blue value from an RGB colour |
| b* | A measure on the blue-yellow scale |
| G | Green value from an RGB colour |
| G′ | The gamma corrected green value from an RGB colour |
| L* | Lightness measured from 0 to 100 |
| R | Red value from an RGB colour |
| R′ | The gamma corrected red value from an RGB colour |
| Y% | Light transmission percentage |

# Chapter 1

# Introduction

## 1.1 Motivation for the project

Steel is a diverse and heavily used product across the world, with over 64 countries producing it [1]. The World Steel Association statistics, which accounts for 99% of the worlds steel output, reported that in the month of March 2018 over 148,000,000 tonnes of steel were produced [1]. In such a competitive industry, it is imperative that companies maintain their customer base by producing steel that meets the standards required. Manual inspection is a long and tedious process. Accuracy is entirely dependent on human error; such as loss of attention, fast line speeds, and differing opinions between inspectors. The use of advanced surface inspection systems helps companies to achieve their aim of increasing accuracy and efficiency.

Whilst visual inspection systems are popular, there are many factors that can make surface inspection difficult in steel mills. There is only one chance to take the perfect photo as the steel passes by the camera at very high speeds so conditions need to be optimised as best as is possible. Dirt and oil can be a problem when trying to take clear images as, in some areas, cameras can get dirty quite quickly. The set-up of the whole camera and lighting rig is very important too. Camera angles, distances and types of lighting [2] will all have an effect on the quality of the images taken and the feature data generated. Depending on the configuration, a defect can be well-defined or completely invisible in an image.

Using black-box systems can prove challenging when problems occur and current in-house systems can require a large amount of upkeep. The main difficulty with highly complex or unknown systems is trying to explain how a system produced an incorrect answer and then work out what needs to be improved to get the correct answer in future. At Tata Steel Europe in South Wales they use a combination of black-box systems and their own in-house systems to manage the product as it goes through the stages of production. These systems are continually being optimised to increase accuracy and efficiency in all areas of product and defect management.

## 1.2 The Aims and Impact of the Project

This project 'Artificial Intelligence and the steel industry: development of an automated quality inspection system' was undertaken with Tata Steel Europe. It establishes how existing quality inspection systems work and, using some of the data and steel samples from the Port Talbot and Trostre sites, explores possible methods that can be used to increase efficiency and accuracy in defect management. This work has been split into three distinct areas of work, considering applications of different methods in each chapter. The computational work carried out in this thesis used Matlab versions 2015b - 2018b [3].

Until the 1990s, quality inspection at the Trostre site had been a manual process, requiring a person to go through each steel coil, checking for defects. The journey to automated inspection began with camera installations and black box software. However the surface quality team wanted to have functionality and optimisation beyond what could be achieved with commercially available software alone. This led to the development of a program, called the Tata Logical Release (TLR), which aids in surface quality inspection. There are cameras on the production line which take images of the strip steel. A black-box system detects features of every kind and sends this data to TLR. These defects are then filtered down different paths depending on the outcome of a number of rules about their appearance and classified as different types of defects. The defects which are a cause for concern are then highlighted for the inspection manager to check over. This quality inspection process enables Tata to ensure that their steel is suitable for the next stages of production, is made to client specifications, and enables an informed decision of next steps if these criteria are not met.

The TLR takes raw data from the inspection system on the line, along with other input data, such as pinhole detectors, and calculates lots of different parameters about the feature, such as length and intensity of grey values. Boundaries are set to determine acceptable measures for each parameter and the decision of whether the feature is acceptable is based on the number of parameters which exceed their boundaries. Capturing optimal images means that the data which TLR receives will have clearly distinguishable defect boundaries and textures. Ensuring that the TLR system continues to function well and accurately means that efficiency and communication lines will be maintained and that yield is optimised.

## 1.3 Overview of the work

### 1.3.1 Part 1

This chapter investigates a method using random forest classification and Gabor filters to detect common gross defects in steel using a set of images from Tata Steel Europe's Trostre site. The impact of the use of Gabor filters is explored, finding that the proposed method had a fairly high rate of accuracy, but still misclassified some defects. The resulting method is compared with existing detection results from Tata's current systems and it was found that the proposed method decreases the false negative rate, but does also slightly decrease the overall accuracy compared to the existing system on this production line. Further testing was completed to compare the proposed method with two variations of it to see what effects this had on speed and accuracy. One variation is a method which uses just a random forest classifier and the other variation is just the proposed method with the Gabor filters removed. It was found that the more methods in the ensemble, the slower the classifier was to train - which is to be expected. Speed should be evaluated further on a pilot line with an industrial-level computer system. As for accuracy, it was found that the overall accuracy decreases as histograms and Gabor filters are used in the ensemble, but the rate of missed defects decreases significantly also. Therefore an appropriate balance must be found between overall accuracy and missed defects.

### 1.3.2 Part 2

This chapter investigates the use of transfer learning, R-CNNs, and Faster R-CNNs to determine the quality of weld holes in steel. The punches that create weld holes wear down with use and inspection systems struggle to identify poorly-punched holes. Identifying weld holes is vital to ensure machines are running at an appropriate speed when welds pass through certain sections of the production line. By establishing whether the punch is still punching good quality weld holes, the punch can be replaced before weld holes start to get missed due to poor quality. Three pre-trained networks were trained, adapted, and tested, Faster R-CNN ResNet-50, Faster R-CNN ResNet-101, and R-CNN GoogLeNet, on a small, hand-labelled data set of weld hole images. These weld holes were split into three defined categories based on the quality of the hole edge and surrounding punched area. It was found that although the R-CNN

GoogLeNet performed the best, none of the methods excelled at detecting the differences between the quality of weld holes and, in their current state, would not be suitable for this application.

### 1.3.3 Part 3

This chapter investigates coloured inspection of steel, as opposed to the infrared lighting and greyscale systems currently used, and how the contrast between defects and non-defects is affected. There were four questions to be explored; the impact of RGB-to-greyscale methods on contrast between defects and steel, the impact of colour filters on contrast between defects and steel, whether the optimal RGB-to-greyscale method is still optimal under different colour filters, and the impact of green filters on the contrast between oiled and clean steel. The colours of steel were explored, with a colour palette produced of defect and non-defect colours. The effect of different RGB-to-greyscale methods was investigated on both the filtered images and the steel colour palette. Colour images of clean and oiled steel surface were taken with a variety of colour filters in place to investigate the filter's impact on contrast. These images were also put through a binary decision tree classifier to evaluate how the RGB-to-greyscale method and the filter colour affected classification results. It was found that the Decolorize method was generally the optimal RGB-to-greyscale method and somewhere on the orange-yellow-light green spectrum should be further explored as the optimal colour. No colour of filter outperformed the non-filtered image, but further types of oil could be tested to confirm this. This project and the collaboration with the surface quality department at Tata Steel Europe in Port Talbot has inspired a pilot-line project to further explore optimal configurations of image acquisition, image processing, and detection/classification methods for various steel grades.

## 1.4 Thesis Outline

The rest of the thesis is organised as follows:

**Chapter 2** covers the background knowledge required for the work undertaken.

**Chapter 3** presents part 1 of the work, considering steel defect detection using Gabor filters and random forest methods.

**Chapter 4** presents part 2 of the work, considering the application of R-CNNs and Faster R-CNNs of the GoogLeNet and ResNet networks to the classification of weld hole quality.

**Chapter 5** presents part 3 of the work, considering the effects of greyscale methods and coloured light on defect contrast to see what improvements could be made to the defect images which are fed into classifiers.

**Chapter 6** discusses the conclusions of the work and any further work suggestions.

# Chapter 2

# Background

## 2.1 Introduction

This section covers the background knowledge acquired over the course of the candidature and reviews of the relevant literature which was required to begin the project. Further chapter-specific literature reviews are at the beginning of each chapter of research. Section 2.2 explains the steel production process. Section 2.3 explains the different types of defects that are experienced in steel production and their root causes. Section 2.4 describes expert systems and the development of Artificial Intelligence. Section 2.5 explains the different visual inspection methods and the associated work being carried out around them. Section 2.6 describes the applications of these methods in industry and the real world.

## 2.2 Steel production process

This section will cover the basics of steelmaking and production processes such as hot-rolling and cold-rolling. Steel is made from iron and a combination of small amounts of other elements such as carbon, manganese, aluminium, and many others. These added elements all have an effect on the properties of the steel produced. Common elements found in steel are listed in Table 2.1 with their uses.

TABLE 2.1: The commonly occurring elements in steel and their effects on material properties [4].

| Element | Their use as an addition in steel |
|---|---|
| Carbon | Found in all steels and affects hardness and strength [5, 6]. |
| Manganese | Found in all steels as it exists in iron ore, but extra is often added for increased strength. |
| Phosphorus | Kept at low level, but is sometimes added for extra strength. |
| Sulphur | Kept at low levels as higher levels can lead to a decrease in yield strength [7]. |
| Nitrogen | Kept as low as possible, but can be added for extra strength. |
| Aluminium & Silicon | Used as de-oxidants, but can be added as an alloying addition. |
| Niobium, Vanadium, Boron, & Titanium | Added to increase strength by forming hard precipitates. |
| Chromium, Copper, Nickel & Molybdenum | Typically residuals in strip steel, but can be added deliberately. E.g. to produce stainless steel. |

The full steelmaking process undertaken from start to finish in South Wales is shown in Figure 2.1. Steelmaking begins when raw materials are heated in the blast furnace to make molten iron. This is then mixed with oxygen and scrap metal in a converter to produce steel [8]. Modern converters can take up to 350 tonnes and convert it into steel in about 15 minutes [9]. The oxygen is blown onto the steel at a very high pressure which combines with the unwanted elements, eliminating them from the molten mix [9]. The amount of scrap metal used dictates the temperature of the metal, so this amount varies [9]. Once refined and tested for composition, the steel is continuously cast into slabs. These slabs then undergo further processing in the hot mill and/or cold mills. The sections of production that will be focused on are those that take place after continuous casting; hot rolling, pickling, galvanizing, and continuous annealing; as these are the areas where the TLR is used.

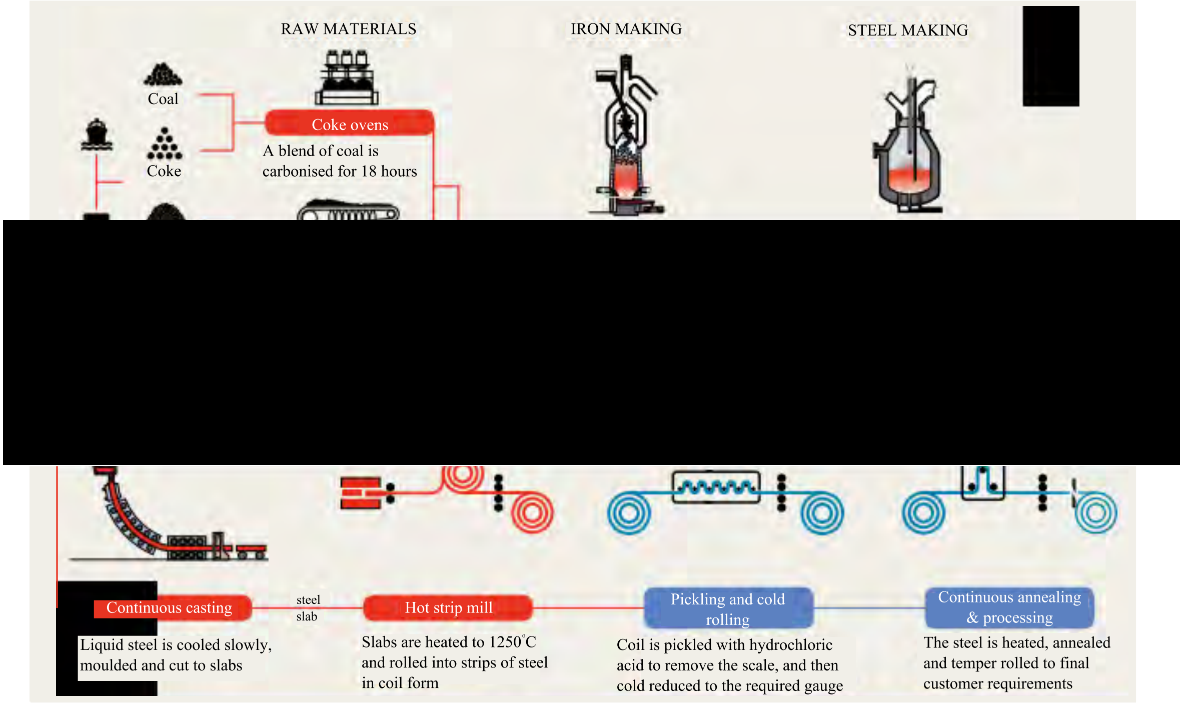
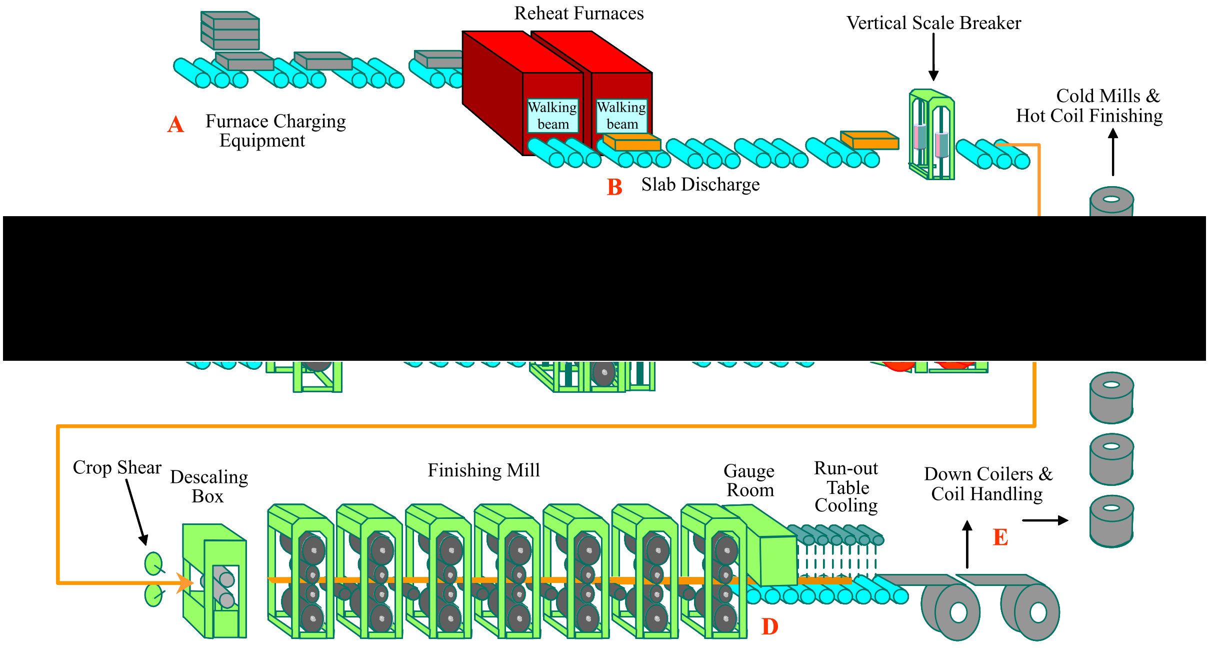

RAW MATERIALS
IRON MAKING
STEEL MAKING
[redacted]
Coal
Coke
Coke ovens
A blend of coal is carbonised for 18 hours
[redacted]
[redacted]
Continuous casting
Liquid steel is cooled slowly, moulded and cut to slabs
steel slab
Hot strip mill
Slabs are heated to 1250˚C and rolled into strips of steel in coil form
Pickling and cold rolling
Coil is pickled with hydrochloric acid to remove the scale, and then cold reduced to the required gauge
Continuous annealing & processing
The steel is heated, annealed and temper rolled to final customer requirements

FIGURE 2.1: The steel making process [10].

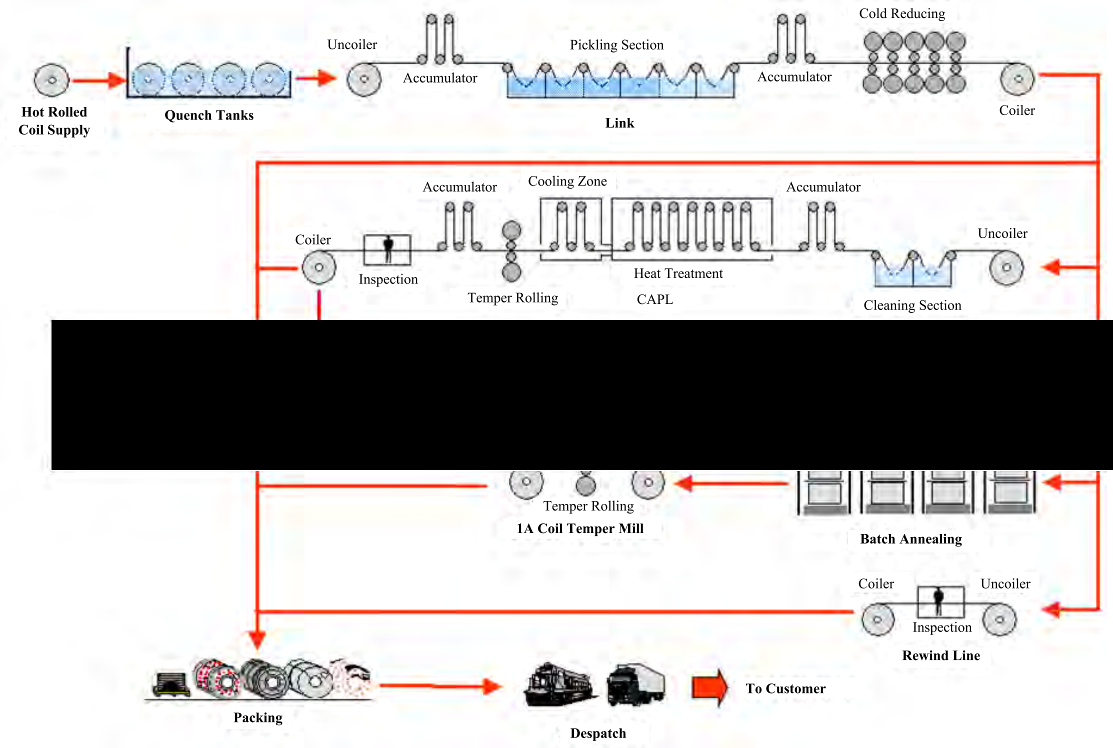
Figure 2.2 shows the hot mill process. The hot mill is an important tool in customising the steel to meet the customer's requirements [11]. The slab is heated uniformly in a reheat furnace to the required temperature of 1150 – 1300°C. Any scale that has formed in the reheat furnace will be removed. The edging and rougher mills then squeeze the slab to its required coil width and reduce the slab thickness to around 30mm depending on its specification. The finishing mill reduces the thickness of the coil to the final gauge and achieves the shape and material properties that are required by the customer. Any remaining scale is removed and the coil is ready to be moved to the next process [11].



[redacted]

FIGURE 2.2: The hot mill process [12].

The linked cold mill at Port Talbot, shown in Figure 2.3, starts with quenching coils to decrease their temperature. Uncoiled, the strips are then pickled to remove any scale formed during earlier processes. The next step is cold reduction where the aims are to reduce the coil to a tighter tolerance gauge, to apply certain surface textures and form any required shapes [4]. The 'CAPL' section in Figure 2.3 is the continuous annealing processing line (CAPL) where the strip is softened to achieve required mechanical properties. This is achieved by heat treatment and temper rolling.



[redacted]

FIGURE 2.3: The cold mill layout [12].

The heat cycle the strip undergoes is shown in Figure 2.4. Heating is very rapid, followed by a short soak then rapid cooling. Unlike the batch annealing process, continuous annealing takes several minutes instead of several days [4]. Temper rolling improves the shape of the strip, gives any surface texture required and, most importantly, lowers the yield point to prevent Luders lines from forming [4].

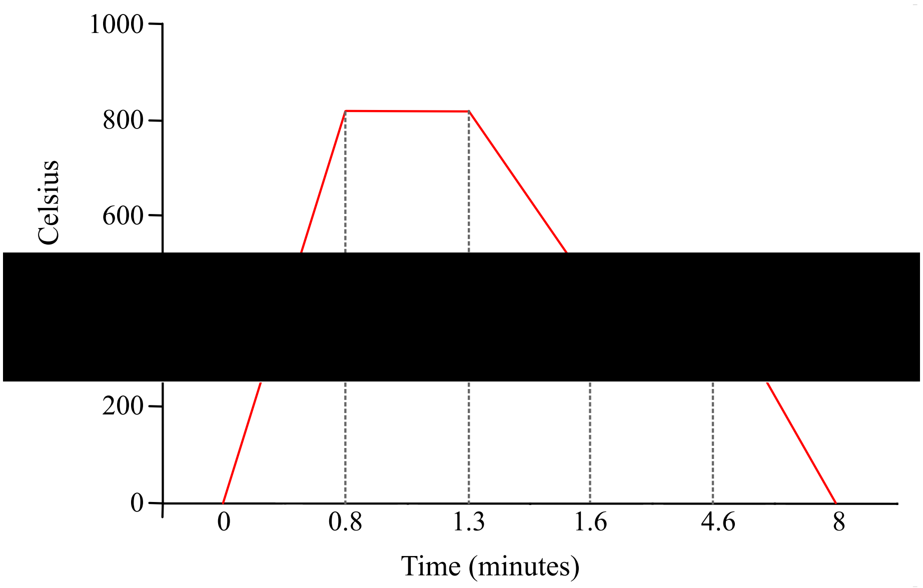

[redacted]

FIGURE 2.4: The heat treatment cycle for continuous annealing [4].

Luders lines are a common occurrence in the stress-strain curves of steels. In the stress-strain curve on the left in Figure 2.5, the curve has a sharp yield point from which the material continues to deform as the flow stress decreases until it reaches a lower yield point and then the material hardens as stress increases [13].

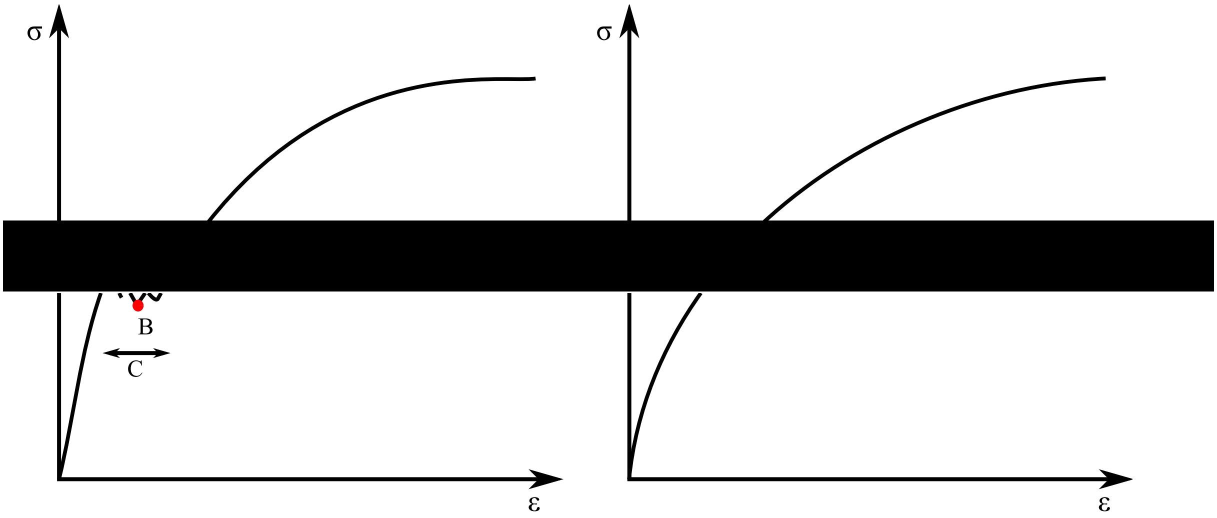

[redacted]

FIGURE 2.5: A stress-strain curve with (left) and without (right) a sharp yield point [13].

Point A on the left curve is the upper yield point and point B is the lower yield point. The section C on the graph is when the Luders lines form. A material with a stress-strain curve without a sharp yield point looks like the one on the right in Figure 2.5. The luders lines are an issue where a smooth surface finish is required because the presence of luders lines gives the material a rough surface [13]. Using a material without a sharp yield point allows work hardening to set in when plastic deformation begins, which spreads deformation uniformly across the material resulting in a smooth surface after deformation [13].

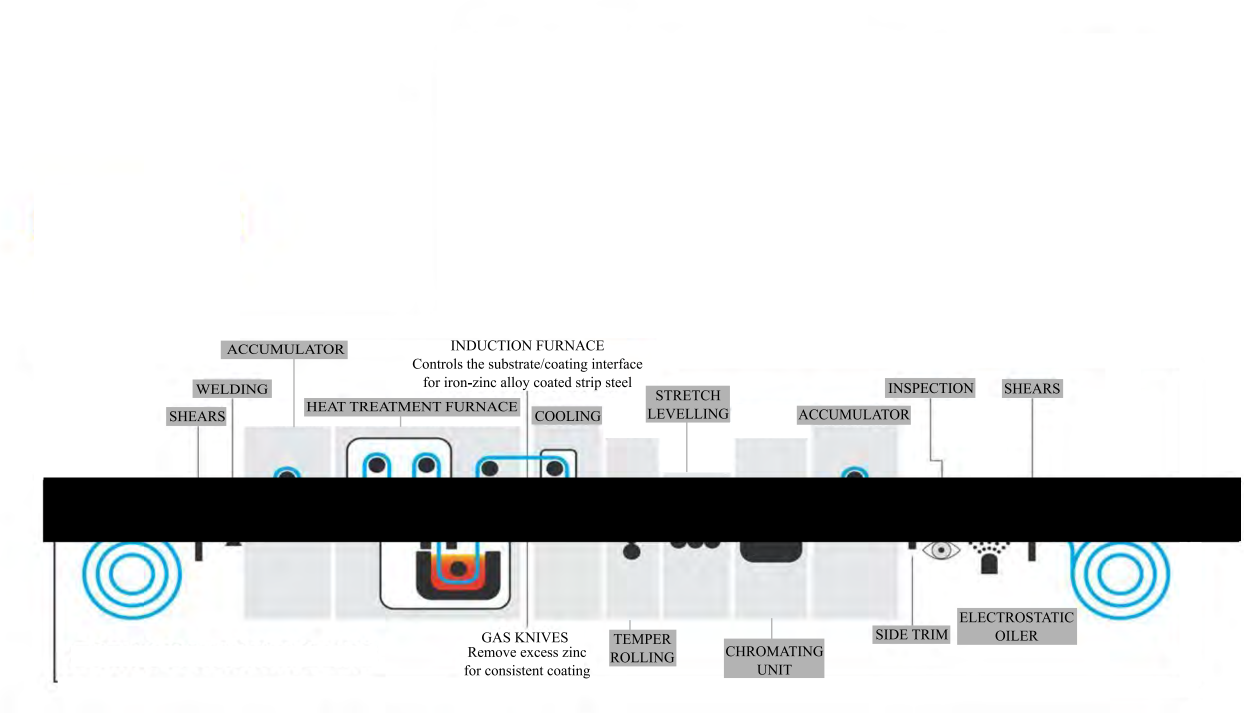

[redacted]

FIGURE 2.6: The zodiac line [12].

## 2.3 Features in steel

There are many different types of features that can occur during steel production and processing. This section will describe the common defects which occur after certain processes, as they are described in [12] unless other sources are stated. These photographs were provided in a generic company powerpoint presentation and as such there is no specific data on size or scale of the images in this section. However, to give an idea of scale, it would be reasonable to assume these images were either taken by cameras on the production line (so approximately 30-60cm away from the surface, depending on the source), or the pictures were taken by a person with a camera, likely less than 1m away from the sample.

Throughout the hot mill, cold mill, and pickling lines at Tata Steel in South Wales, the most common defects picked up on the defect recognition cameras are laminations, rolled in debris, edge laminations and scale. On the galvanising line the most common occurring defects are laminations, tension digs, scabs, holes, blisters, scratches and edge damage.

The rate at which each type of feature occurs varies between processes. Appendix B shows a graph of all features which occurred on the tin-plate line over a 2 week period. Some defects, for example damaged edges or certain types of scratches, only occurred a few times in that two week period. Whereas features such as dirt and scale are detected over 10,000 times. These sorts of numbers are to be expected. Damage that makes a product unfit for purpose occurs rarely in comparison to features which are typically rectified further along in the production process, such as dirt.

**Laminations** Laminations, seen in Figure 2.7, are a common surface feature that is found in isolated sections throughout the steel production process. They occur in the steel plant and can happen for three reasons:

1. Non-metallic inclusions during steel making.
2. Clusters of aluminium oxide forming during steel making.
3. Incompletely scarfed slab cracks.

Laminations can be recognised as a blistering strip in the rolling direction of the strip with varying levels of severity. They cause a weakness in the strip which can be a safety concern for certain processes.

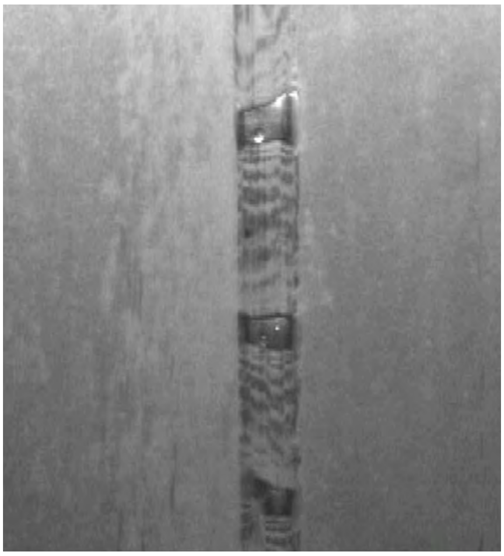


FIGURE 2.7: A lamination from the hot mill.

**Rolled in debris** Rolled in debris, featured in Figure 2.8, is a common problem when work rolls are used as any debris can be rolled into the strip. This looks similar to a lamination or scab and often has peeled and irregular edges. As with laminations, rolled in debris causes a weakness in the strip which again could be a safety concern for certain processes.

**Scale** Scale is shown in Figure 2.9 and there are four possible causes of scale:

1. The worn work roll surfaces on the finishing stands pick up scale from the strip and impress the pattern back on to the strip.
2. Scale formed in the reheat furnaces or between roughing stands is not properly removed by the water jets due to either inadequate water pressure or a blocked jet.
3. Small pieces of scale that are not washed off between finishing stands then pickle out leaving deep pits in the surface of the strip.
4. Single pieces of scale get rolled in to the strip during hot rolling and are not pickled out.
5. Due to their composition, some steels, such as high-silicone steels, are more likely to have scale which is difficult to remove.

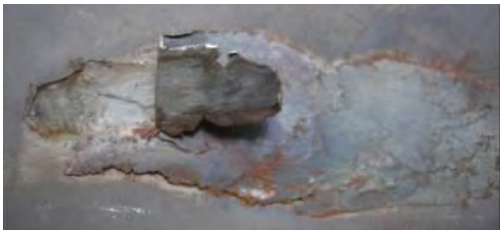


FIGURE 2.8: Rolled in debris.

FIGURE 2.9: Heavy scale from the hot mill.

**Tension digs** Tensions digs, seen in Figure 2.10, are created during post coiling by interlap slippage. They look like lots of small gouges on the surface of the strip and can occur anywhere on the strip. Tension digs cause damage to further processes and are not suitable for any applications of coils.

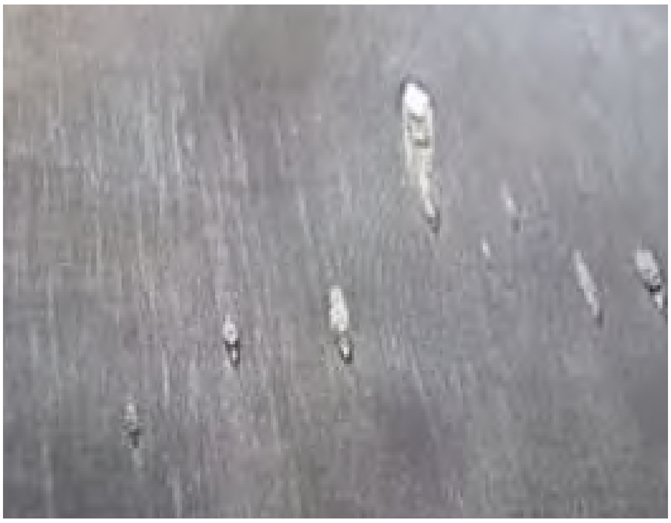


FIGURE 2.10: Tension digs from post coiling.

**Scabs** Scabs are often confused with other defects such as rolled in scale, but the way to distinguish it is that scabs are ductile when bent, whereas scale is brittle and crumbly [14]. Scabs have scale beneath them and cracks may be present in severe cases due to stress concentration from the scab causing the material beneath to crack [14]. Scarfing is a process where the surface of a slab is melted and an oxygen jet used to dislodge oxidised steel from the surface; this removes scabs prior to rolling. [15].

**Holes** A discontinuity in the strip which extends from the lower surface to the upper surface, holes can range in size from a pinhole to large irregular holes, but whatever the size they are a serious problem for manufacturers of cans and other sheet metal products [16].

**Blisters** One of the reasons why argon is added during slab casting is that it helps to move out any non-metallic inclusions and impurities [17]. However, trapped gases and impurities can create blisters in the surface of the steel as seen in Figure 2.11. They usually appear after hot rolling or annealing and are easily seen after galvanising as they have a tadpole-like appearance. Blisters can cause problems in applications such as automotive where a good surface finish is required [17].

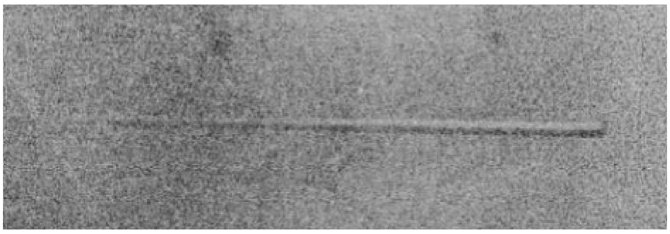


FIGURE 2.11: A blister forms on the surface of the strip.

**Scratches** Scratches, seen in Figure 2.12, are caused by stationary mill furniture or by debris being held against a moving strip. Any amount of the coil can be affected, but scratches are most often seen on the ends of the coil. Severe scratches will cause weakness in the strip which leads to processing issues and shallower scratches may be structurally ok, but can show through a painted finished product.

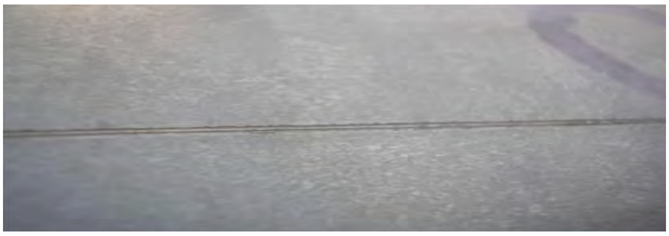


FIGURE 2.12: Scratches which can originate from anywhere in the production process.

**Edge damage** Damage to the edges of a roll can happen when there are defects on the edge of the strip which make it weak thus more susceptible to cracking. Damage can also occur when the strip snags on mill furniture. Edge damage is most commonly found on the ends of the strip, but can occur on any of the edges. Like most features, this causes weakness in the strip which can be a safety concern.

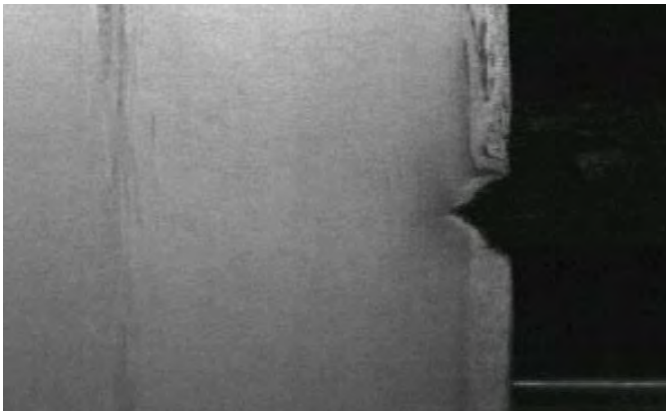


FIGURE 2.13: Edge damage from the hot mill.

## 2.4 Expert AI systems

This section details the history and development of artificially intelligent systems and how they are used in society today. The development of expert systems is an area of research within the field of artificial intelligence which focusses on complex decision making [18]. An expert system is defined as 'a computer system that simulates human experts in a given area of specialisation' [19]. This means that in order to be considered an expert system, it must have the ability to process and learn information, to demonstrate learning and reasoning by making acceptable and explainable decisions, and be able to communicate with the user and other systems [19].

In the chapter History and Applications in the book Expert Systems [20], John Durkin discusses the progress of computing since the first digital computers in the 1940s. The idea of expert intelligent systems became a possible reality in the 1950s when early AI research developed systems capable of playing games such as chess [21] and checkers [22], however funding cuts shortly followed after a review highlighted the lack of return for investing in AI research. The first commercial expert system was developed in 1978, called XCON, to select computer parts for Digital Equipment Company; eight years later the system was saving them $20 million. By the 1980s, expert systems were being installed in various businesses with all manner of applications across the world, examples of which can be found in Table 2.2 taken from Table 1 in Expert Systems [20].

TABLE 2.2: Major Application Areas of Expert Systems [20].

| | | |
|---|---|---|
| Agriculture | Environment | Meteorology |
| Business | Geology | Military |
| Chemistry | Image processing | Mining |
| Communications | Information management | Power systems |
| Computer systems | Law | Science |
| Education | Manufacturing | Space technology |
| Electronics | Mathematics | Transportation |
| Engineering | Medicine | |

A report in The Conversation discusses the four types of artificial intelligence; reactive, limited memory, theory of mind, and self-aware [23]. The first two are the types one can encounter in daily life, whereas the latter two types are only conceptual or works in progress. A reactive machine will only respond to its given input and is not capable of learning based on previous experience. A limited memory machine has the ability to learn from a bank of data and make decisions based on it. Most of the popular AI researched and discussed currently is of this type, such as autonomous driving and virtual assistants. Theory of mind refers to an artificially intelligent machine which would have the ability to form an understanding of beings and objects in the world and their behaviours, and form representations of the world around them. The final step would be the ability to be self-aware; a machine which would be able to form representations about itself.

The required components of an expert system are a knowledge base, which is a set of rules built from knowledge from experts in the appropriate field, and an inference engine, which will draw deductions based on the rules in the knowledge base [24]. These rules typically form 'if A, then B' statements where the logic is either boolean or fuzzy. An example of boolean and fuzzy logic can be seen in Figure 2.14 [25]. Boolean logic is binary; true or false, 1 or 0, a sample belongs in a class or does not. Fuzzy logic is used to describe vagueness - to what extent does something fit in a class.

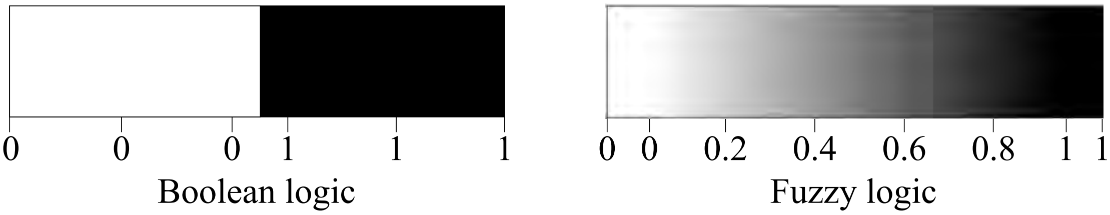


FIGURE 2.14: An example of boolean and fuzzy logic.

Expert systems can be used in a variety of applications, including banking, traffic control, medicine and manufacturing. The main advantages for using an expert system are [26]:

- Expert systems cannot forget.
- Increase in efficiency.
- Consistent decision making for similar situations.
- Can be reproduced.
- Can hold more knowledge than any individual person.

However there are still disadvantages such as the inability to be creative, no common sense, and not being able to recognise if a problem falls outside of its area of expertise [26].

The next phase of the industrial revolution is transforming traditional industry through the use of new technology, called *Industry 4.0*. With this movement to utilise technology, quality control is an area in which expert systems are being developed. At Tata Steel Europe, moving from human inspection of products to intelligent and automated real-time systems has led to decreases in the number of missed faults and the data collected can be used for root cause analysis of issues. The concept of Industry 4.0 aims to combine all of these expert systems which are currently used individually and create a smarter way to manufacture; this has many benefits including, but not limited to [27]:

- Predictive maintenance
- Asset management and tracking
- Optimise energy usage
- Improve customer satisfaction
- Improve health and safety
- Improve efficiency
- Increased production

## 2.5 Visual inspection methods

This section describes how automated visual inspection systems are used for product quality control. Visual inspection begins with data gathering. In the case of quality inspection this typically means capturing images of the product. As well as images, there could also be other data collected and input to help with decision-making; such as heat profiles, speed or force readings, and hole testing by shining light through the product. This data is then processed to gain the most useful information in the smallest amount of data, to minimise cost. This processed data is then fed to a classifier of some description; whether it is a rule-based system or a neural network, a classifier will be take the processed data and make a decision about it.

### 2.5.1 Image acquisition methods

This section describes the considerations and methods of acquiring image data. In visual inspection image acquisition is achieved using a digital camera. Cameras are placed at the end of production lines so quality control can be carried out in real time. These images can be pieced together to create a picture of the entire coil with all of its features.

To acquire digital images, a sensor (typically a charge coupled device (CCD)) is used to obtain electrical charges which are proportional to illumination [28]. A CCD typically consists of a 486 × 768 array of photosensitive elements on a $10.5 \times 11\mu$m grid which collect electrical charges during the accumulation phase; these charges are then converted to electric voltage [28].

**Definition 1.** *'An image may be defined as a two dimensional function I(x,y), where x and y are spatial co-ordinates. The amplitude of I at any pair of co-ordinates (x,y) is called the intensity or grey level of the image at that point. When (x,y) and amplitude values of I are all finite, discrete quantities, we call the image a digital image.' [29].*

An image has two factors [29]:

1. $f_{illumination}(x,y)$, the amount of source illumination on the scene being imaged.

2. $f_{reflected}(x,y)$, the amount of illumination reflected by objects in the scene.

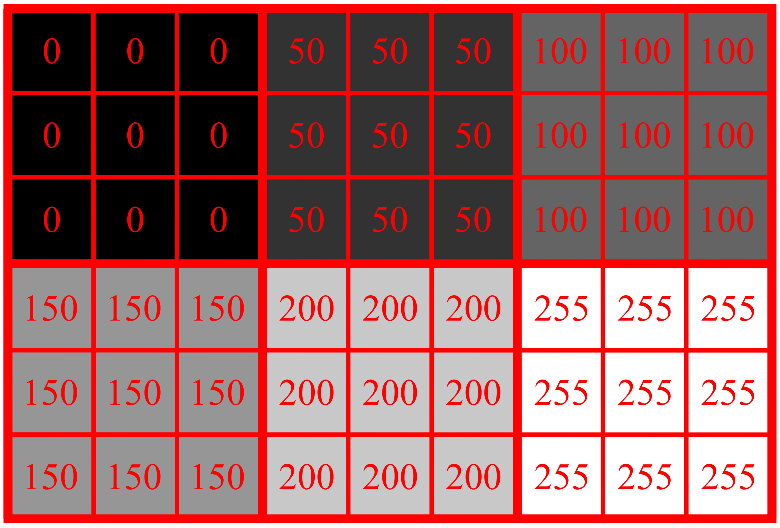
This can be represented as $I(x,y) = f_{illumination}(x,y) \cdot f_{reflected}(x,y)$ where $0 < f_{illumination}(x,y) < \infty$ depends upon the illumination source and $0 < f_{reflected}(x,y) < 1$ where 0 means no reflection and 1 means total reflection [29].

To represent a digital image of $n \times m$ pixels, the function $I(x,y)$ is represented as a matrix seen in equation (2.1) where $m, n$ are positive, real numbers [29].

$$I(x,y) = \begin{pmatrix} I(0,0) & I(0,1) & \cdots & I(0,n-1) \\ I(1,0) & I(1,1) & \cdots & I(1,n-1) \\ \vdots & \vdots & & \vdots \\ I(m-1,0) & I(m-1,1) & \cdots & I(m-1,n-1) \end{pmatrix} \tag{2.1}$$

The grey level, $g$, has been digitised in the range $[0, L-1]$ called the dynamic range, meaning there are $L$ levels [29]. In terms of storing an image, the grey level is calculated in terms of bits, $k$, where $L = 2^k$ meaning an image with a 256 grey level is an 8 bit image [29].

For example, Figure 2.15 has six different grey levels. Each grey level is represented by a $3x3$ red grid, where each box in that grid represents a pixel. The red numbers are the grey level of that pixel. So in a greyscale image, the shade black is represented by a grey level 0, the shade white is represented by a grey level of 255, and various shades of grey are in the range $(0, 255)$.



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 50 | 50 | 50 | 100 | 100 | 100 |
| 0 | 0 | 0 | 50 | 50 | 50 | 100 | 100 | 100 |
| 0 | 0 | 0 | 50 | 50 | 50 | 100 | 100 | 100 |
| 150 | 150 | 150 | 200 | 200 | 200 | 255 | 255 | 255 |
| 150 | 150 | 150 | 200 | 200 | 200 | 255 | 255 | 255 |
| 150 | 150 | 150 | 200 | 200 | 200 | 255 | 255 | 255 |

FIGURE 2.15: An example of different grey values (red values) and the shade of grey they produce (background shade).

A practical example how grey values represent images can be seen in Figures 2.16 and 2.17. The original image of the scratch can be seen in Figure 2.16 and Figure 2.17 is a 3-D plot of the grey values for the same image. Each pixel represents 0.24mm of the steel's surface. Each square in the plot represents a pixel, its row and column number within the original image are the $x$ and $y$ axis coordinates, and the $z$ axis coordinate represents the grey value of the pixel. Where the scratch gets darker in the original image, a crevice is created in the 3-D plot. The crevice covers approximately 4 pixels, making the scratch about 1mm wide.

FIGURE 2.16: A greyscale scratch image.

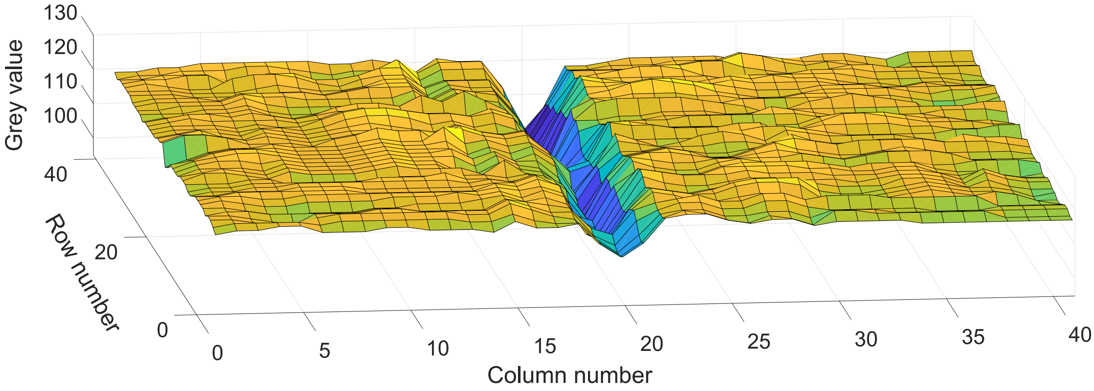


FIGURE 2.17: A 3-D plot of the grey values of image 2.16.

### Analog vs digital

The main differences between analog and digital cameras are highlighted in Table 2.3. Analog cameras are cheaper and simpler, but come with limitations in resolution and frame rate; whereas digital cameras are more expensive, but provide high resolution images at high speeds [30]. If high resolution images and fast speeds are required, then spending the extra investment on digital cameras may be more suitable.

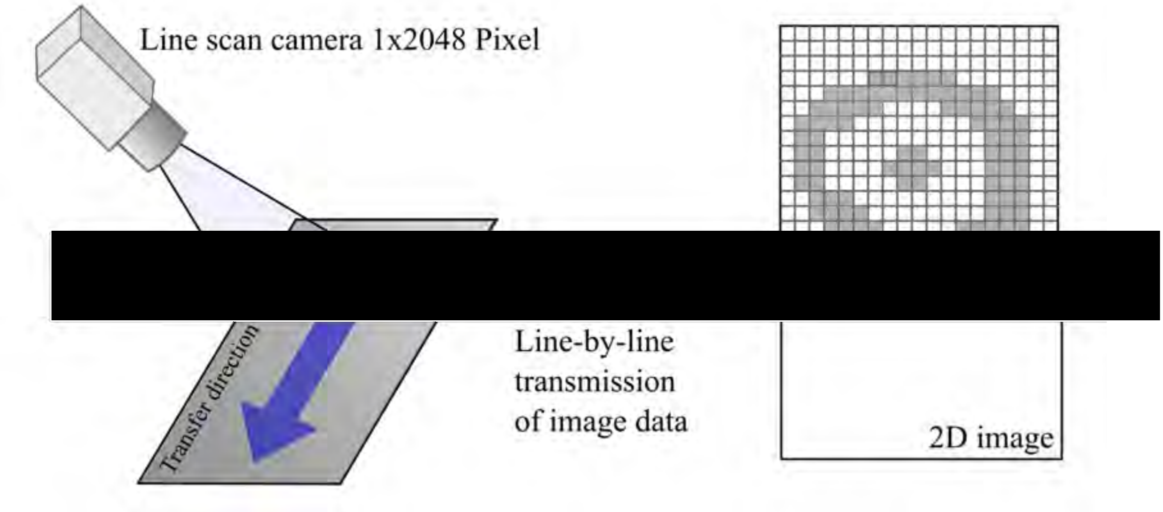
TABLE 2.3: Comparison of analog vs digital cameras [30].

| | **Analog cameras** | **Digital Cameras** |
|---|---|---|
| Vertical resolution | Limited by bandwidth of the analog signal | Not limited in horizontal or vertical directions giving high resolution |
| Sensors | Standard size | Large numbers of pixels and sensors available due to unlimited bandwidth, giving high resolution |
| Display requirements | Computers and capture boards not necessary, but can be used for digitising | Computers (and in some cases capture boards) required for display |
| Ease of integration | Printing and recording easily integrated into system | Can be transmitted in low bandwidth for ease |
| Quality | Susceptible to noise and interference leading to loss in quality | Digital output signal so minimal loss in quality |
| Frame rates | Limited | High frame rates and fast shutter speeds |

### Line-scan vs. area-scan cameras

Selecting the correct type of camera for a visual inspection system is crucial to achieving accuracy in feature classification. The type of camera required depends on several different factors such as how detailed the image needs to be, what lighting conditions are possible, what is it trying to capture an image of, will the camera fit onto the process line and many other considerations. The two main types used are line scan and area scan cameras, seen in Figure 2.18 and 2.19. Table 2.4 highlights the differences between the two types.

Area scan cameras are widely used in image acquisition. They capture an image in one exposure cycle using a large matrix of image pixels [31]. Area scan cameras can suffer from light variation from the top of the image to the bottom when photographing a round surface, but correction algorithms exist which can generally fix the problem by adjusting background levels accordingly. For this type of camera, any defect which features in multiple images will have those images stitched together to create a full feature image.


Line scan camera 1x2048 Pixel
Transfer direction
Line-by-line transmission of image data
2D image

[redacted]

FIGURE 2.18: An example of a line scan camera [32].

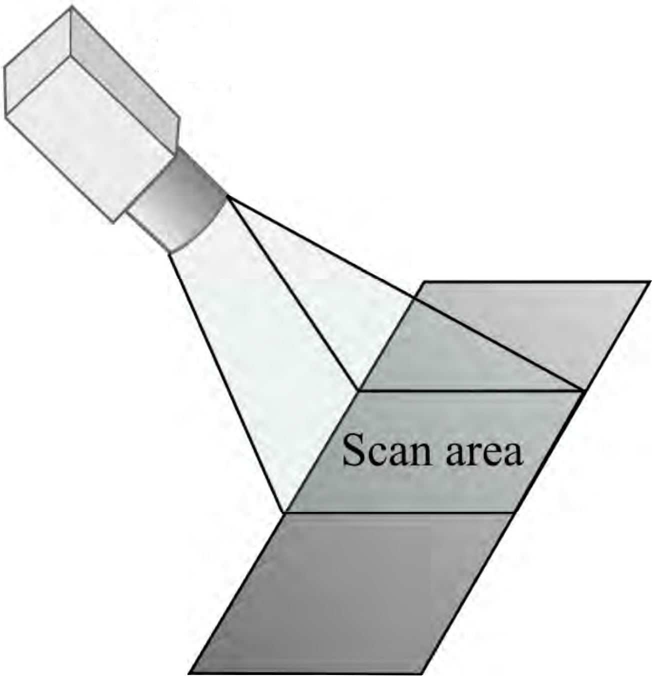


FIGURE 2.19: An example of an area scan camera.

A line scan camera has a single row of pixels which capture image data very quickly so continuous images can be reconstructed in a much higher resolution than area scan images [33]. A line scan camera uses signal detection to calculate the normalised signal level with upper and lower tolerances. However where long consistent features exist, such as scratches, the signal may eventually decide that the scratch signal is the normalised signal level due to its consistency, meaning the actual length of the scratch may not be known.

TABLE 2.4: Comparison of area scan vs line scan cameras [30].

| **Area scan cameras** | **Line scan cameras** |
|---|---|
| Typically 4:3 ratio | Linear sensor |
| Large sensors | Larger sensors |
| High-speed applications | High-speed applications |
| Fast shutter times | Constructs images one line at a time |
| Lower cost than line scan | Object moves under sensor |
| Wider range of applications | Suitable for capturing wide objects |
| Easy setup | Complex alignment and timing required, but simple lighting |

Installation of cameras on steel production lines can be very difficult. Depending on which line the cameras are on, cameras will need to be kept clean to allow clear images to be taken. Retrofitting cameras onto existing production lines can be difficult due to space limitations. Whereas area scan cameras can be pointed anywhere on the strip, line scan cameras have to be pointed at a roll, which have to be contrasting colour to steel, to ensure the camera can detect the edge of the strip. If space is a limitation then it is important to consider which camera would be most appropriate. Despite only requiring simple lighting, optimising line scan cameras is far more critical than area scan cameras and a lot more time is spent getting the clearest image [33].

### Lighting considerations

National Instruments suggests that there are three criteria one should aim to meet when deciding on lighting for a machine vision system [2]. These are:

1. Maximize the contrast on features of interest

2. Minimize contrast elsewhere
3. Robustness

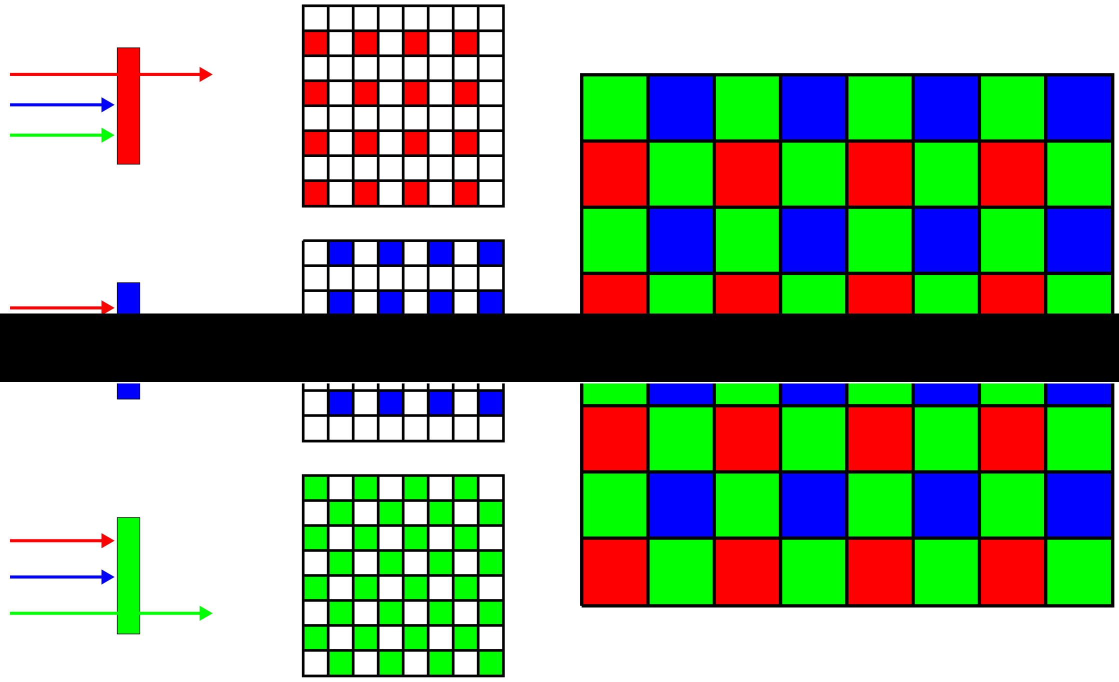
Some lighting types will perform better on some types of features than others, so satisfying criteria number three can be a challenge when there is variation in the features or the surrounding environment.

Halogen, flourescent and LED lighting are the most commonly used types of lighting in machine vision inspection systems [2]. Halogen bulbs may be gradually phased out in the coming years as they require up to five times more power than LEDs [34]. Over large areas flourescent lighting may be more cost effective than LEDs. However over small areas, LEDs are the most cost-effective and exceed other lighting types in longevity [2].

**Colour** When automated visual inspection systems were first being widely installed across businesses, greyscale cameras were the only technology that could keep up with production speeds and produce images of a suitable quality resolution. Three channel colour cameras were not feasible in real-time on a steel production line as they require more data processing. Technology has been continually developing and using digital colour cameras is now a possibility which can be considered for real-time inspection.

One way a camera can detect colour is by using a Bayer filter, which can be seen in Figure 2.20. The Bayer filter is designed with twice as many green filters than red or blue due to the human eye's natural bias towards green light. Each square represents a pixel; in the example of a red filtered pixel, the red intensity will be measured directly and the blue and green colour information for that pixel will be interpolated based on neighbouring blue and green pixel data [35]. This type of colour camera is suitable when colour accuracy and fine details are not required [35].

Another method by which a camera can detect colour is by using a beam splitter to direct the incoming light onto three separate sensors; each with a red, blue or green filter between the light and sensor. This provides full colour information, achieving better contrast, vivid colours, and a highly detailed image [35].



[redacted]

FIGURE 2.20: A Bayer filter [36].

**Lighting techniques** There are four methods of lighting used in machine vision applications; back lighting, bright field, partial bright field, and dark field lighting. Back lighting places the object between the light and the camera and is typically used for finding holes in objects or measuring the width of an object [2].

Bright field lighting; either full or partial, is achieved by placing the light source near to the camera, directly or at a slight angle, over the surface. This means a surface defect will show up darker, as an absence of light. Dark field lighting is achieved by lighting at an angle closer to the material surface, meaning light is only reflected towards the camera where a surface defect is present. An example of dark and bright field lighting can be seen in Figure 2.21 [37].

On a steel production line an inspection system will generally follow the set up in Figure 2.22. Although multiple cameras may be used to cover the full width of the strip of steel. These cameras will all be of identical specifications, with the same resolutions. These cameras are also at the same fixed distance and angle from the strip. However the configuration of resolution, camera type, distance, and angles will be tailored specifically for each installation depending on the design of the production line the cameras are being installed on and what type of material is being inspected. For the image data used in chapter three this

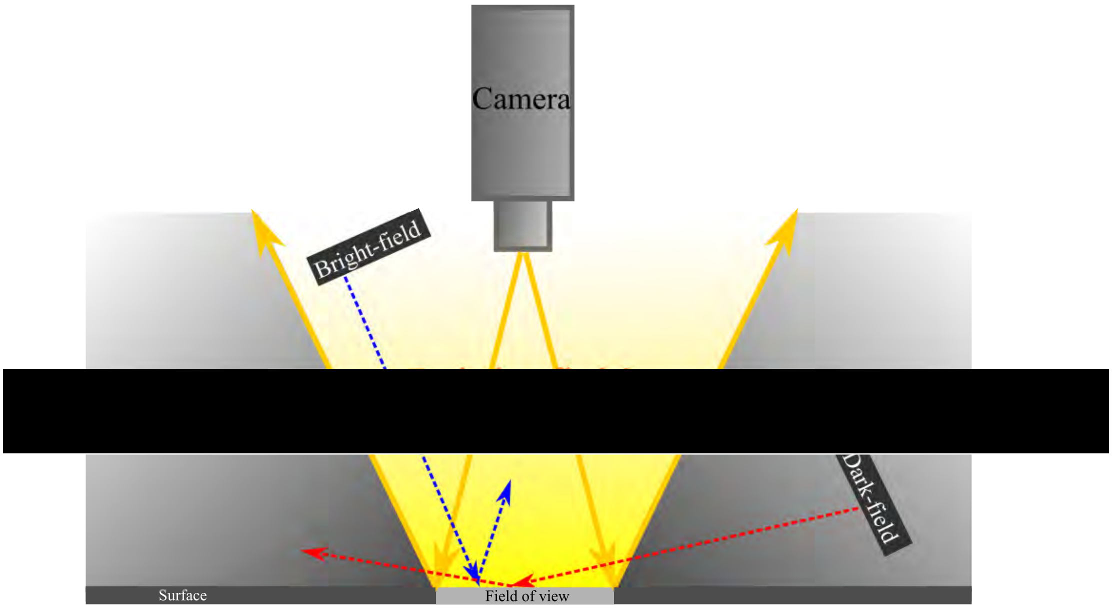

[redacted]

FIGURE 2.21: An example of bright-field and dark-field lighting [37].

fixed distance was 60cm. The steel moves through at anything up to a maximum speed of approximately 600 metres per minute and cameras have a frame rate of 35 frames per second so that a map of the entire strip can be produced. These images will then be processed and defects detected and classified. This information will then appear on screen for an inspector to double check and decide whether the strip should continue on its production route or requires rework, having a section cut out, or being recycled.

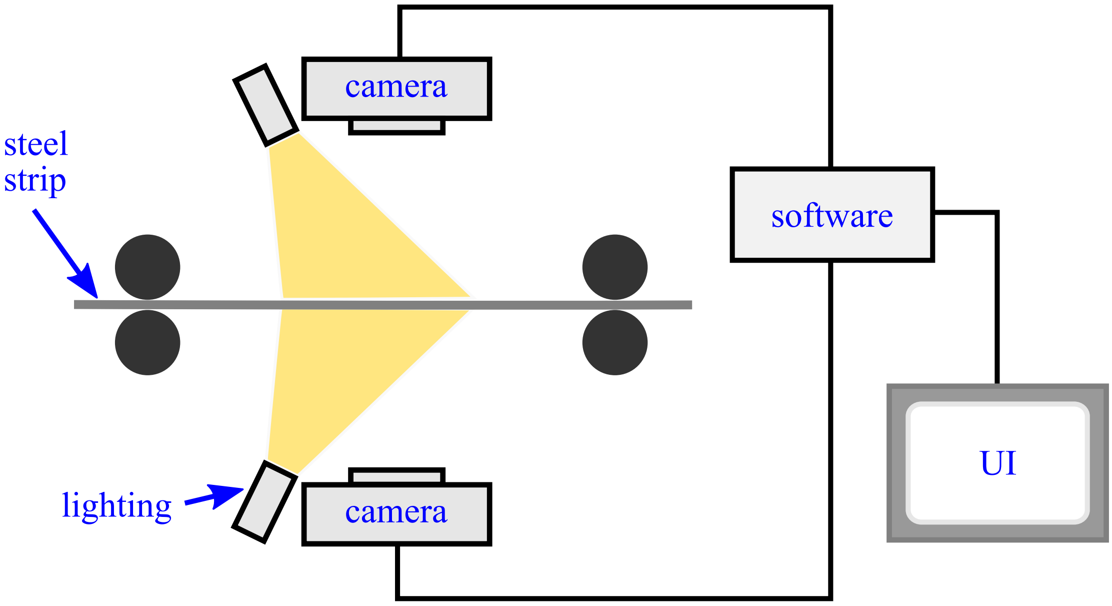


FIGURE 2.22: An example of an inspection system on a steel production line.

### 2.5.2 Edge detection techniques

This section will look at basic concepts behind edge detection and some common edge detection methods as discussed in *Machine Vision* [38]. First, some definitions will be established.

**Definition 2.** *Edges can be defined as significant local changes in image intensity and they occur on the boundaries between different regions in an image.*

**Definition 3.** *An edge point, $[i, j]$ where $i$ is in the x-direction and $j$ is in the y-direction, is a point in an image where there is a significant local intensity change.*

**Definition 4.** *An edge detector is an algorithm that detects and displays all of the edge points in an image.*

The set of edge points produced by an edge detector has two subsets: 'edges' which are correctly identified as edges and 'false edges' which are not in fact edges, but have been wrongly identified as edges. The set of missing edges can also be defined as those edges which were not identified by the edge detector.

Detecting significant local changes in grey values can be done by calculating an approximation of the gradient $G_x$ and $G_y$ in the $x$ and $y$ directions respectively, seen in equation (2.2).

$$G_x \cong f[i, j+1] - f[i, j] \text{ and } G_y \cong f[i, j] - f[i+1, j] \tag{2.2}$$

There are three steps to edge detection methods.

1. Filtering: Filters are used to reduce noise in the image, but there is an aspect of balancing required here as reduced noise also means losing the strength of edges. Noise should be reduced enough that it doesn't get wrongly picked up as an edge, but not so much that the edges which should be captured also get filtered out.

2. Enhancement: By calculating the gradient magnitude, the image is enhanced to emphasize pixels in areas of significant local changes in intensity.

3. Detection: Typically a gradient threshold is applied as the criterion to determine which points are edge points and which are not. The larger the gradient, the bigger the difference in intensity meaning it is likely to be an edge point.

There have been many edge detectors developed over the last 20 years. The function $f(x, y)$, in Figure 2.23, represents the change in intensity that occurs at an edge. It shows how the first and second derivatives can be used to determine that an edge is present by using thresholds and zero crossings.

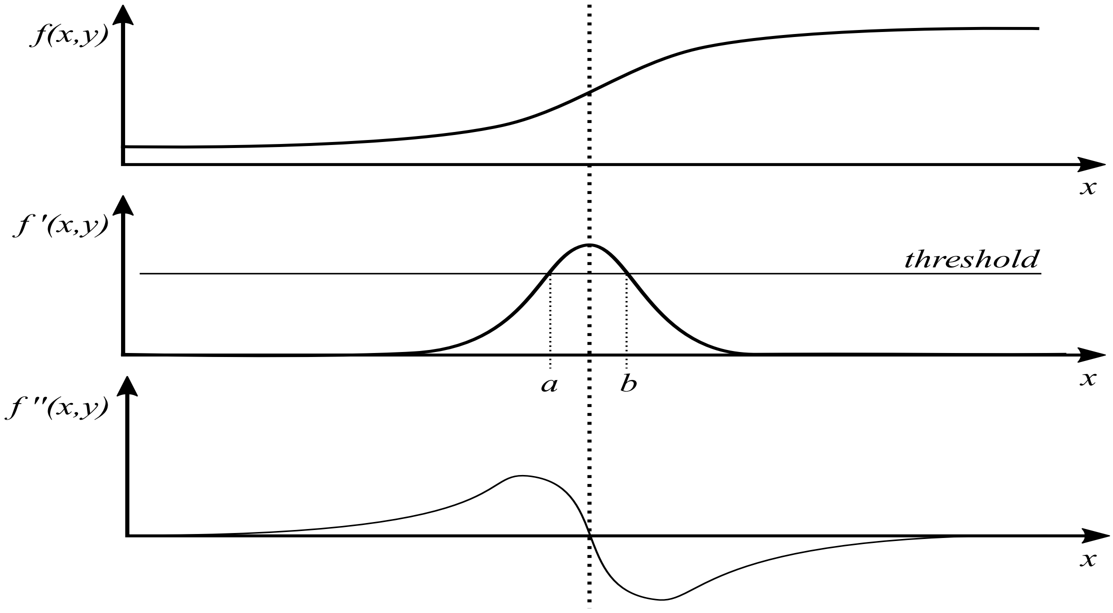


FIGURE 2.23: When using threshold detection, all points between *a* and *b* will be edge points, the midpoint of these corresponds to the zero crossing of the second derivative.

The common edge detectors using the first derivative are the Roberts, Sobel and Prewitt operators. These methods compute the first derivatives and use a threshold criterion to determine the presence of an edge point. The common techniques using the second derivative are the Laplacian operator and the second directional derivative which uses the face that a peak in the first derivative means a zero crossing in the second derivative. Therefore the zero crossings indicate the presence of an edge point. Other methods discussed are the Laplacian of Gaussian and the Canny edge detector.

**Roberts operator** The Roberts cross operator calculates the gradient magnitude as:

$$M_{Roberts}[f[i,j]] = |f[i,j] - f[i+1, j+1]| + |f[i+1, j] - f[i, j+1]| . \qquad (2.3)$$

Using convolution masks, equation (2.3) becomes $M_{Roberts}[f[i,j]] = |G_x| + |G_y|$ where

$$G_x = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & -1 \\ \hline \end{array} \quad G_y = \begin{array}{|c|c|}\hline 0 & -1 \\ \hline 1 & 0 \\ \hline \end{array} \tag{2.4}$$

The Roberts operator computes the gradient at the interpolated point $[i + \frac{1}{2}, j + \frac{1}{2}]$, not at $[i, j]$.

**Sobel operator** A way to avoid calculating the gradient at an interpolated point is to use a convolution mask that is $3 \times 3$ as seen in equation (2.5) and calculate the gradient of the central pixel.

$$\begin{array}{|c|c|c|}\hline a_0 & a_1 & a_2 \\ \hline a_7 & [i,j] & a_3 \\ \hline a_6 & a_5 & a_4 \\ \hline \end{array} \tag{2.5}$$

The Sobel operator calculates the magnitude of the gradient $M_{Sobel} = \sqrt{s_x^2 + s_y^2}$ where equations (2.6) and (2.7) shows how the partial derivatives are computed with $c = 2$.

$$s_x = (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6) \tag{2.6}$$

$$s_y = (a_0 + ca_1 + a_2) - (a_6 + ca_5 + a_4) \tag{2.7}$$

These partial derivatives, $s_x$ and $s_y$ can be applied using their convolution masks in equation (2.8). Emphasis is placed on the pixels closest to the center of the mask.

$$s_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \quad s_y = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array} \tag{2.8}$$

**Prewitt operator** The Prewitt operator is like the Sobel operator, it uses the same labelling defined in equation (2.5) and the $s_x$ and $s_y$ equations, except the $c$ value is set to equal one instead of two. Setting $c = 1$ the convolution masks for the Prewitt operator are defined in equation (2.9). There is no emphasis placed on the location of the pixels in this operator.

$$s_x = \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad s_y = \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array} \tag{2.9}$$

**Laplacian operator** The Laplacian of a function $f(x,y)$, defined in equation (2.10), is equivalent to the second derivative which will equal zero at the location of the edge.

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{2.10}$$

The second derivative is approximated using difference equations, centered about the pixel $[i,j]$ as given in equations (2.11) and (2.12).

$$\frac{\partial^2 f}{\partial x^2} = f[i,j+1] - 2f[i,j] + f[i,j-1] \tag{2.11}$$

$$\frac{\partial^2 f}{\partial y^2} = f[i+1,j] - 2f[i,j] + f[i-1,j] \tag{2.12}$$

This gives the approximation to the Laplacian as:

$$\nabla^2 f \approx \begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array} \tag{2.13}$$

**The second directional derivative** The second directive derivative computes in the direction of the gradient using the formula in equation (2.14). In this notation

$$\begin{aligned} \frac{\partial^2 f}{\partial n^2} &= \frac{(\frac{\partial f}{\partial x})^2 \frac{\partial^2 f}{\partial x^2} + 2\frac{\partial f}{\partial x}\frac{\partial f}{\partial y}\frac{\partial^2 f}{\partial y \partial x} + (\frac{\partial f}{\partial y})^2 \frac{\partial^2 f}{\partial y^2}}{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2} \\ &= \frac{f_x^2 f_{xx} + 2 f_x f_y f_{xy} + f_y^2 f_{yy}}{f_x^2 + f_y^2} \end{aligned} \tag{2.14}$$

The second directional derivative and the Laplacian are affected by noise more so than operators using a single derivative. This means they are not frequently used in machine vision, but can be combined with other methods to make a more robust detection method.

**Laplacian of Gaussian** The Laplacian of Gaussian technique combines Gaussian filtering with the Laplacian to make a less noisy edge detection method. The Gaussian filter smooths the image depending on the value of $\sigma$, the spread of the Gaussian, and the higher the value the better the noise filtering, but more edge information could be lost. Transforming the edges into zero crossings enhances the image data and then these zero crossings are detected to determine edge points. The Laplacian of Gaussian, $h(x,y)$, is calculated as in equation (2.15) [39].

$$h(x,y) = [\nabla^2 g(x,y)] * f(x,y), \tag{2.15}$$

where

$$\nabla^2 g(x,y) = (\frac{x^2+y^2-2\sigma^2}{\sigma^4}) \exp^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

An example of the Laplacian of Gaussian mask for a $5 \times 5$ grid of pixels is shown in equation (2.16).

| | | | | |
|---|---|---|---|---|
| 0 | 0 | −1 | 0 | 0 |
| 0 | −1 | −2 | −1 | 0 |
| −1 | −2 | 16 | −2 | −1 |
| 0 | −1 | −2 | −1 | 0 |
| 0 | 0 | −1 | 0 | 0 |

(2.16)

**Canny edge detector** The Canny edge detector is a multi-stage detector algorithm defined as:

**Definition 5.** 1. *'Smooth the image with a Gaussian filter.*

2. *Compute the gradient magnitude and orientation using finite-difference approximations for the partial derivatives.*

3. *Apply nonmaxima suppression to the gradient magnitude.*

4. *Use the double thresholding algorithm to detect and link edges.' [38]*

Denoting the image as $I[i,j]$, applying the Gaussian filter gives the filtered image:

$$I_f[i,j] = G_f[i,j;\sigma] * I[i,j] \tag{2.17}$$

where $\sigma$ controls the degree of smoothing. First difference approximations, seen in equation (2.18) are then used to calculate the gradients $P[i,j]$ and $Q[i,j]$, calculated in equations (2.19) and (2.20), respectively.

$$G_x = \begin{array}{|c|c|}\hline -1 & 1 \\ \hline -1 & 1 \\ \hline \end{array} \quad G_y = \begin{array}{|c|c|}\hline 1 & 1 \\ \hline -1 & -1 \\ \hline \end{array} \tag{2.18}$$

$$P[i,j] \approx \frac{(I_f[i,j+1] - I_f[i,j] + I_f[i+1,j+1] - I_f[i+1,j])}{2} \tag{2.19}$$

$$Q[i,j] \approx \frac{(I_f[i,j] - I_f[i+1,j] + I_f[i,j+1] - I_f[i+1,j+1])}{2} \tag{2.20}$$

From here the magnitude and orientation of the gradient can be computed as:

$$M[i,j] = \sqrt{P[i,j]^2 + Q[i,j]^2} \tag{2.21}$$

$$\theta[i,j] = arctan(Q[i,j], P[i,j]) \tag{2.22}$$

Where the gradient is large, there will be large values in the magnitude image array, $M[i,j]$. An algorithm called nonmaxima suppression thins these ridges of gradient magnitude by suppressing all values which are not the peak values of the ridge. The first step is to reduce the angle of the gradient, $\theta[i,j]$, as shown in Figure 2.24, by stating that the edge orientation falls in to one of the four categories.

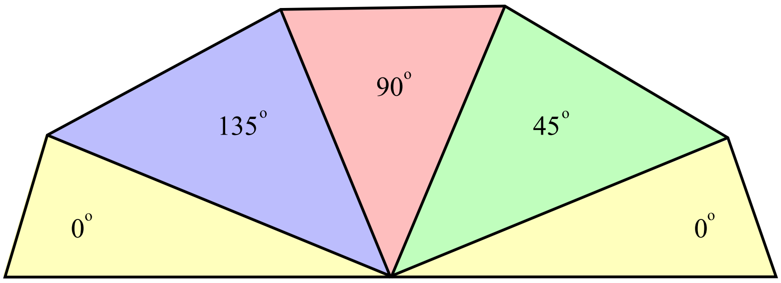


FIGURE 2.24: The four categories that edge orientation falls into, e.g. if an edge is detected at $4^o$ then it is classed as being at $0^o$.

Then a $3 \times 3$ neighbourhood is passed over the array which sets the center element of $M[i,j]$ to zero if it is not the largest element compared to both neighbouring magnitudes. This now means the non-zero values of the new array, $M_{sup}[i,j]$, represent any large difference in image intensity. however there will still be some false edge points in the array, caused by noise or texture.

A threshold is then applied to reduce the amount of noise and false edge points on the suppressed array, $M_{sup}[i,j]$. This gives the final result of an array of the edge points in the image, $I[i,j]$. If the threshold is too low there will still be some false-positives showing noise and if the threshold is too high then there will be some false negatives which will lose some edges and detail of the image. A double-thresholding system can be used, where two arrays are produced; one with a low threshold and one with a high threshold. The low threshold array is then used to fill in the gaps of the high threshold array to create a more accurate final result.

**Performance of edge detectors** The aim of edge detectors is to detect edges by reducing the amount of data and noise in an image. There have been many studies into the performance of edge detectors, comparing aspects such as the probability of false positives and false negatives and the error in the estimated angle and location of an edge compared to the true edge. Several papers [40, 41, 42, 43] concluded that the Canny edge detector performs highly and in some cases out performs all other methods. However, a study of edge detectors should be carried out for the individual application as some can perform better than others for particular styles of images.

### 2.5.3 Image filtering techniques

The aim of filtering images is typically to reduce noise and improve the contrast and quality of the image for its intended purpose. Noise in digital images can come from many different sources [44].

- Capture noise occurs during the image capturing process, from incorrect lighting, dust, improper focus, or other environmental factors at the time of image capture.
- Occlusion noise occurs if an object is partially blocked when the image was taken. This is an issue in visual inspection applications as the inspection system can only inspect what is visible in the image.

- Noise can occur because of limitation during the digitising of analogue images or during compression of these images if unsuitable formats are used.
- Processing noise can be introduced due to rounding errors and approximations.

This next section defines common smoothing spatial filters as described in [45]. Figure 2.25 shows a rolled in debris image which will have the following filters of applied to it to show the different outcomes of the filters.

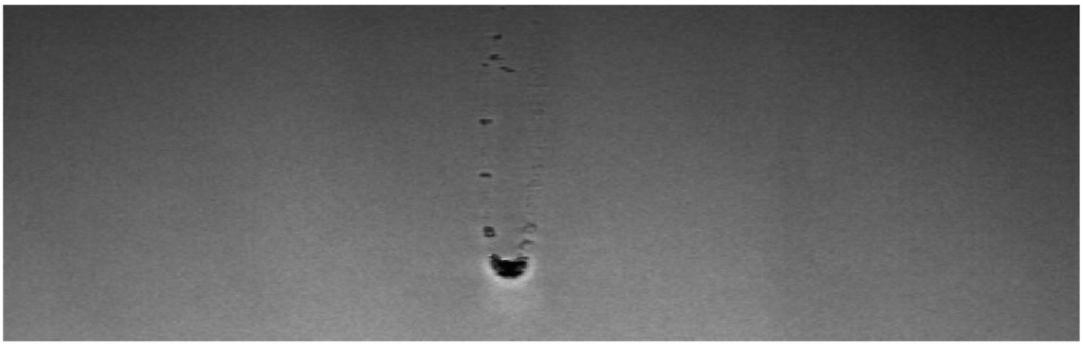


FIGURE 2.25: A greyscale image of rolled in debris.

**Mean Filter** The mean filter calculates the mean of all pixels in a defined neighbourhood, around the central pixel. For example, using a 3x3 neighbourhood, where $[i, j]$ is the pixel in the $i$th row and $j$th column of the image $I$, the filtered image, $I_f[i, j]$, is calculated in equation 2.23. Figure 2.26 shows the original image, Figure 2.25, with the mean filter applied using a 3x3 neighbourhood size.

$$\begin{aligned} I_f[i,j] = \frac{1}{9}(I[i-1,j-1] + I[i-1,j] + I[i-1,j+1] + I[i,j-1] + I[i,j] \\ + I[i,j+1]I[i+1,j-1] + I[i+1,j] + I[i+1,j+1]) \end{aligned} \tag{2.23}$$

**Gaussian filter** The Gaussian filter is a commonly used filter to blur or smooth images. The filter uses a Gaussian function to calculate the transformation applied to each pixel. Equation 2.24 shows the two dimensional equation; where $x$ and $y$ are the distance from the origin in the horizontal and vertical axis and

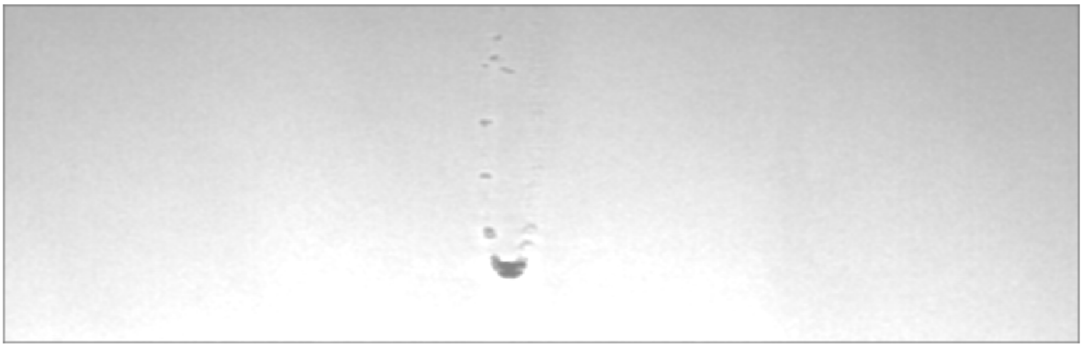


FIGURE 2.26: Mean filter with a 3x3 neighbourhood applied to Figure 2.25.

$\sigma$ is the standard deviation of the Gaussian distribution. Figure 2.27 shows the original image, Figure 2.25, with the Gaussian filter applied when $\sigma = 2$.

$$F_{g_2}(x, y) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.24}$$

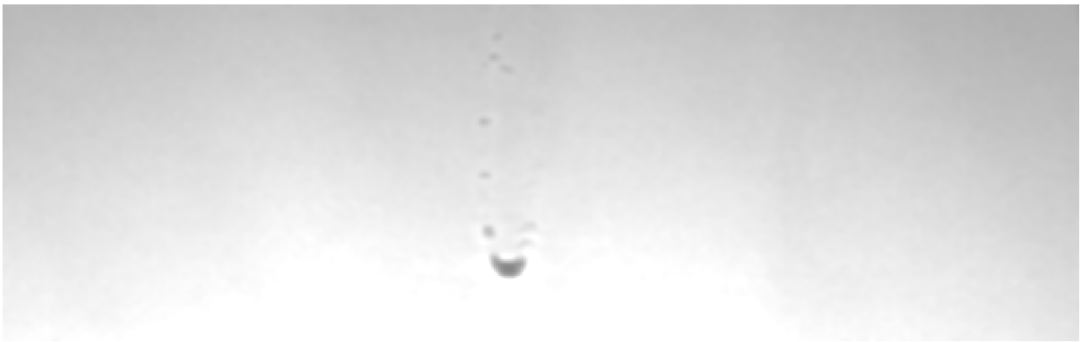


FIGURE 2.27: Gaussian filter where $\sigma = 2$ applied to Figure 2.25.

**Median filter** The median filter calculates the median value of each neighbourhood to compute the each filtered pixel. Figure 2.28 shows the original image, Figure 2.25, with the median filter applied using a 3x3 neighbourhood size.

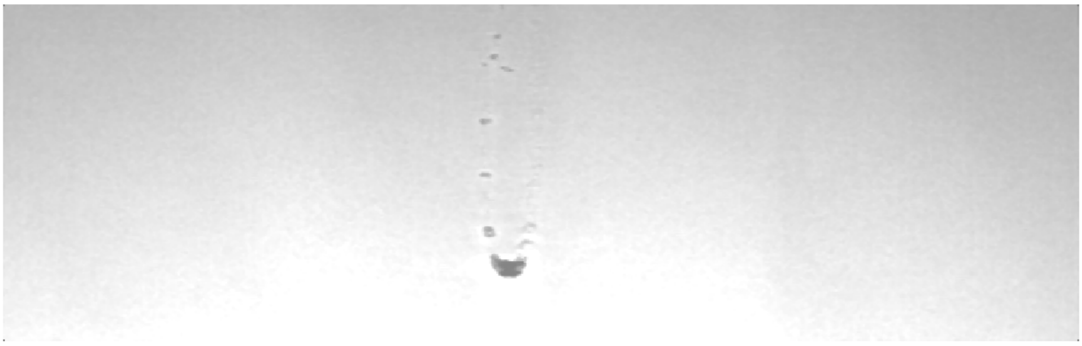


FIGURE 2.28: Median filter with a 3x3 neighbourhood applied to Figure 2.25.

**The minimum and maximum filters** Working in a similar way to the mean filter, the minimum or maximum values from a neighbourhood can be taken which help to get rid of any black and white noise. This type of noise usually results from dead pixels or faulty hot pixels which produce completely black and completely white spots on an image [44]. Examples of these equations can be seen in equations 2.25 and 2.26. Figures 2.29 and 2.30 show the original image, Figure 2.25, with the minimum and maximum filters applied, respectively, both using a 3x3 neighbourhood size.

$$F_{min}[i,j] = min(I[i-1,j-1], I[i-1,j], I[i-1,j+1], I[i,j-1], I[i,j], I[i,j+1], I[i+1,j-1], I[i+1,j], I[i+1,j+1]) \tag{2.25}$$

$$F_{max}[i,j] = max(I[i-1,j-1], I[i-1,j], I[i-1,j+1], I[i,j-1], I[i,j], I[i,j+1], I[i+1,j-1], I[i+1,j], I[i+1,j+1]) \tag{2.26}$$

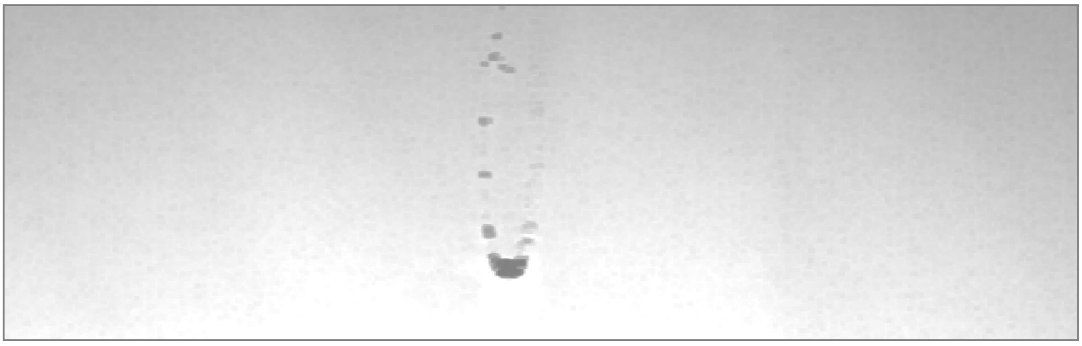


FIGURE 2.29: Minimum filter with a 3x3 neighbourhood applied to Figure 2.25.

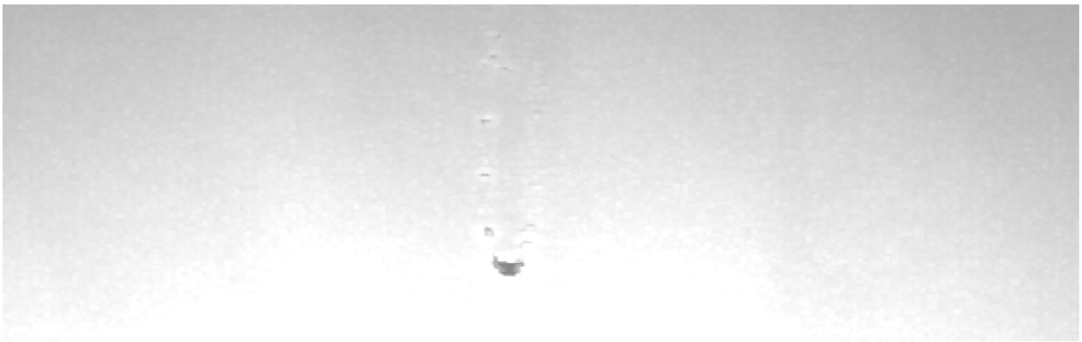


FIGURE 2.30: Maximum filter with a 3x3 neighbourhood applied to Figure 2.25.

**Mid-point filter** The mid-point filter, seen in equation 2.27, takes equations 2.25 and 2.26 and takes the mean of the two values for each pixel. Figure 2.31 shows the original image, Figure 2.25, with the mid-point filter applied.

$$F_{mid}[i,j] = \frac{(F_{min} + F_{max})}{2} \tag{2.27}$$

FIGURE 2.31: Mid-point filter applied to Figure 2.25.

**Gabor filters** Gabor filters are a type of filter widely used for feature extraction, particularly in the fabric industry [46, 47, 48, 49, 50]. The use of Gabor filters in machine vision is motivated by their similarity to the way the human eye works [51, 52]. Gabor filters extract spatial-frequency data and a bank of filters is typically used at multiple frequencies and orientations. Figure 2.32 shows the original image, Figure 2.25, with a Gabor filter of frequency 10 and orientation 45° applied.

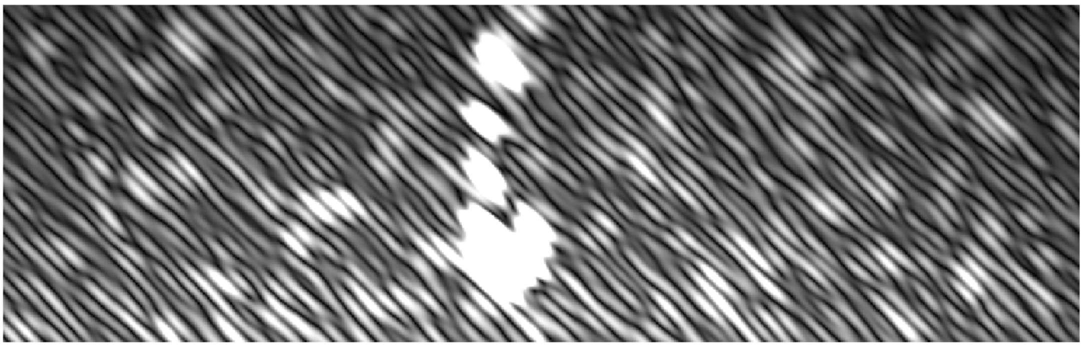


FIGURE 2.32: Gabor filter with frequency 10 and orientation 45° applied to Figure 2.25.

### 2.5.4 Supervised machine learning

The goal of machine learning is to induce a function from a set of given training data and then evaluate the performance of this function by testing it on a set of test data [53]. There are several different types of machine learning problems, the common examples are listed below [53]. There must be a strong relationship between training data and the unseen test data so performance of the algorithm can be accurately measured [53].

- Regression: Trying to predict a real future value based on knowledge of the past.
- Binary Classification: Trying to predict a yes/no response.
- Multiclass Classification: Trying to put data into one of a group of classes. This is the type of learning carried out by TLR and some of the commercial systems used at Tata Steel Europe.
- Ranking: Putting a set of objects in a pre-defined order, e.g. predicting which of a list of items will be most to least expensive.

Machine learning can be split into two main types: supervised and unsupervised learning. Unsupervised learning is where a machine tries to identify patterns and structures within data, without any training input [54]. Unsupervised learning is more similar to human learning; when a person looks at a picture for the first time everything hasn't been pre-labelled, unlike supervised learning [54].

Supervised learning is the type carried out most commonly [54]. Classification and regression are both types of supervised learning. Classification has a finite number of responses whereas regression, although similar, has an output that is continuous [54].

Supervised learning is carried out using a set of labelled data, split into two subsets of training and test data. The training data is used to train whichever learning method is employed and then the algorithm is tested using the new set of test data. This test data must be very similar to the training data to ensure the algorithm is trained properly. A third set of validation data can also be used to Since these sets of data are labelled, we can verify the accuracy of the algorithm on both the training data and the test data.

### Classification methods

This section will look at the common image classification techniques. There are four steps to image classification: Pre-processing the image, training the system to recognise the features of each class, choosing a method to compare the image with the training data and assessing the accuracy of the result [55]. Data is split into different *classes*, which are essentially categories, and each class has a *class name*, which is just a given name - for example, the class name of a group of car images may be ′cars′, or the class name of a group of objects the colour blue may be ′blue things′; it is the labellers choice.

Classification can be supervised or unsupervised. Unsupervised classification is where there is no human input, the computer analyses the data and groups pixels into a set number of classes. This can be less accurate, but is necessary when there is not a large amount of image data available. Supervised classification uses large amounts of training data so it learns to recognise the features of each class which has already been correctly verified by the user.

**Decision Tree** The decision tree classifier compares the image data with a range of selected features. It does this by repeatedly partitioning the data set into subsets in a hierarchical fashion [55]. At each step class labels are either accepted or rejected until terminal nodes are reached with the relevant class label [55]. The use of multiple decision trees is known as a random forest classifier. This method will grow multiple decision trees and then use the most popular classification as the final decision. Random forest classifiers are discussed in further detail in section 3.2.3.

**Support Vector Machines** A support vector machine can distinguish between two classes and locate the optimal boundary by using optimisation algorithms [55]. It uses hyper planes to separate the data samples and the optimal solution is found by the hyperplane which is furthest from the nearest data point [56].

**Fuzzy Set Theory** Typically a data point will be labelled as a class with membership grade 1, or 0 if it is not in that class. If data is not easily separable, i.e. fuzzy, then Fuzzy set theory takes a membership grade as somewhere between 0 and 1 which is defined by the membership function [57]. Using a stochastic approach, different groups of characteristics can be described in a similar manner [55].

**Artificial Neural Networks** Neural networks are based on how the brain functions and have a series of layers, each of which contains a set of neurons with weighted connections to all other neurons [55]. Neural networks are discussed in further detail in section 2.5.4.

**Neural networks** This section discusses the theory behind neural networks and is based on Nielsen's book 'Deep Learning and Neural Networks' [58] unless otherwise stated. An artificial neural network is a learning method modelled on the way the brain works to identify objects and patterns. A neural network is made up of layers of connected neurons where each neuron has an input which is the weighted sum of all of the outputs of the neurons in the previous layer [59]. Each neuron has an activation function, which is fed the inputs and computes the output for the neuron.

**Neuron types** There are two main types of neurons; perceptrons and sigmoid neurons. Figure 2.33 shows the perceptron neuron, which takes binary inputs, $x_1, x_2, ..., x_n$, and gives one binary output, 0 or 1. The binary outcome of this output, as in equation (2.28), is determined by setting a threshold or bias, $b$, for the value of the weighted sum, which has weights, $w_1, ..., w_n$.

$$\text{output} = \begin{cases} 0, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0, \\ 1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0. \end{cases} \tag{2.28}$$

where $b$ is the bias and $\mathbf{w} \cdot \mathbf{x} \equiv \sum_{j=1}^{n} w_j x_j$.

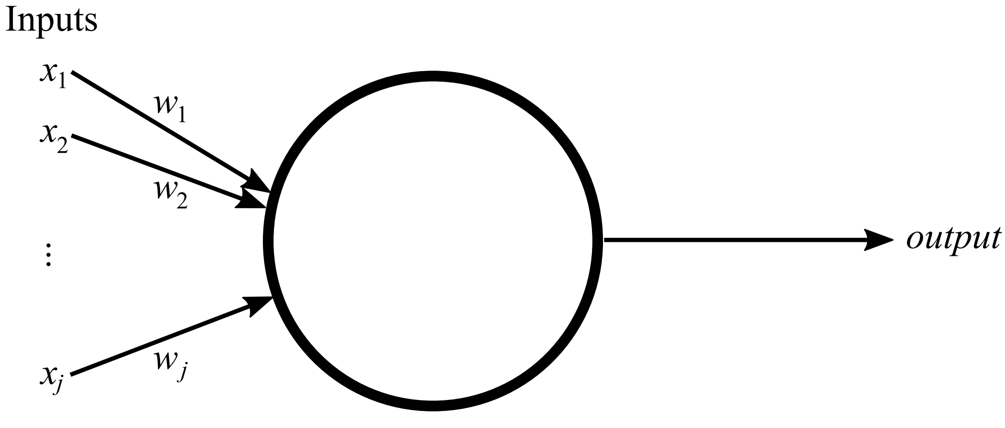


FIGURE 2.33: A perceptron with $n$ binary inputs with weights $\mathbf{w}$ and a single binary output.

Sigmoid neurons are modified perceptrons, so that any small change to their weight or bias leads to a small change in their output; this allows them to learn. The inputs of a sigmoid neuron are not binary meaning they can take on any value between 0 and 1. There are still weights and an overall bias value, but the output of a sigmoid neuron, seen in equation 2.29, is $\sigma_s(\mathbf{w} \cdot \mathbf{x} + b)$, called the sigmoid function.

$$\text{output} = \sigma_s(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + \exp(-\sum_{j=1}^{n} w_j x_j - b)} \tag{2.29}$$

Figure 2.34 shows the difference between a perceptron and a sigmoid neuron. Due to the smoothness of the sigmoid function, it is possible to make very small adjustments to the inputs and output, as seen in equation (2.30).

$$\partial \text{output} \cong \sum_j \frac{\partial \text{output}}{\partial w_j} \partial w_j + \frac{\partial \text{output}}{\partial b} \partial b \tag{2.30}$$

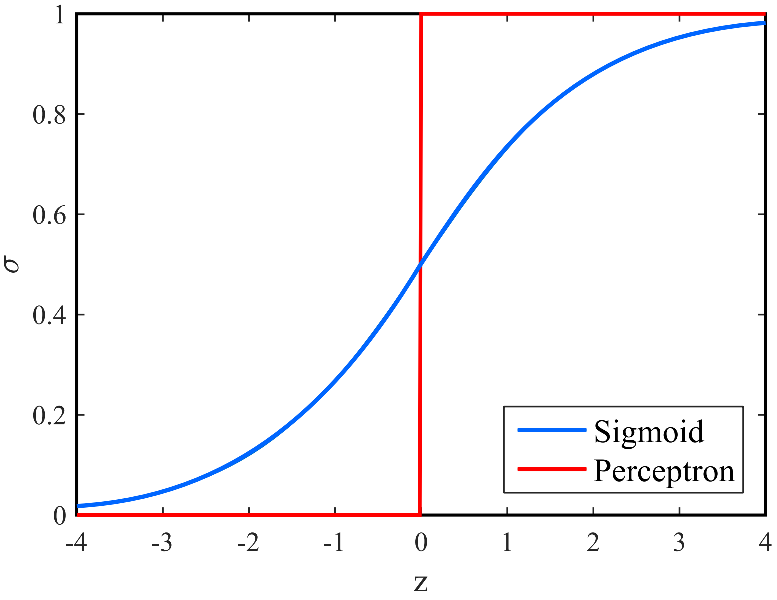


FIGURE 2.34: The sigmoid function is a smooth function, unlike perceptrons which are a stepping function.

There is also the tanh neuron, which uses the hyperbolic tangent function as its activation function, and the rectified linear neuron, which is used in deep learning and defined in equation 2.31. Although it only takes positive inputs, unlike the other neuron types, rectified linear neurons do not experience learning saturation.

$$\text{output} = max(0, \mathbf{w} \cdot \mathbf{x} + b) \tag{2.31}$$

Neural networks contain hidden layers of neurons, the green neurons in Figure 2.35, which simply means that the neurons are neither input nor output neurons. The network seen in Figure 2.35 is a feed-forward network where the output from the previous layer is used as the input for the next layer. A recurrent network would be one which featured feedback loops.

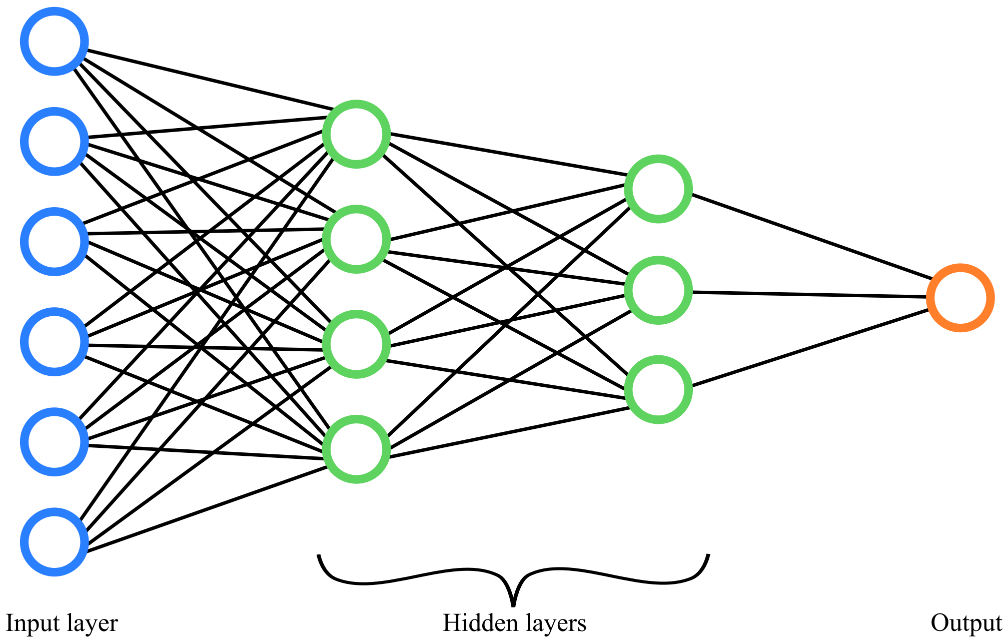


FIGURE 2.35: A four-layered neural network where the blue neurons are the inputs, the green neurons form the two hidden layers and the orange neuron is the ouput.

**Gradient descent** Gradient descent is a method of learning used in neural networks. It works by minimising a cost function, like the quadratic cost function defined in equation 2.32.

$$C_{quad}(w,b) = \frac{1}{2n}\sum_{x}\|a_{aim}(x) - a\|^2 \tag{2.32}$$

where $n$ is the number of training inputs, $w$ is all of the weights in the network, $b$ is the biases of all neurons and $a$ is the vector of outputs, also known as activations, when $x$ is input and the sum is over all inputs, $x$.

The idea is to find the values of $w$ and $b$ which make the network output $a$ as near as possible to $a_{aim}(x)$, our ideal output (or, in most cases, the known answers). For example, say there is a data set of images of dogs and cats and there are two possible output predictions, output $= [\text{cat}, \text{dog}]^{\mathrm{T}}$. Given an input, $x_j$, which is an image of a cat, $a_{aim} = [1,0]^{\mathrm{T}}$ and if the prediction is incorrect then $a = [0,1]^{\mathrm{T}}$ and if the prediction is correct then $a = [1,0]^{\mathrm{T}}$. So as $a_{aim}(x) \rightarrow a \Rightarrow C_{quad} \rightarrow 0$ with a certain combination of weights and biases. This gives the gradient descent iterative rule as described in equation 2.33; this equation has to be iterated through for every input $x_j$.

$$w \rightarrow w' = w - \eta \frac{\partial C_{quad}}{\partial w} \quad b \rightarrow b' = b - \eta \frac{\partial C_{quad}}{\partial b} \tag{2.33}$$

where $\eta$ is the learning rate, $w$ is the weights, and $b$ is the biases.

Due to the fact this equation iterates through every input to make each update, learning can be slow for large inputs, so stochastic gradient descent can be used to speed up the learning process. What makes stochastic gradient descent much faster than gradient descent is that the equation is only iterating through either one input or a small subset of inputs to make each update, instead of the whole input set. The gradient is estimated for a small sample of random training inputs called mini-batches of size $m$. This gives the updated iterative rule, known as stochastic gradient descent, described in equation 2.34.

$$w \rightarrow w' = w - \frac{\eta}{m} \sum_j \frac{\partial C_{x_j}}{\partial w} \quad b \rightarrow b' = b - \frac{\eta}{m} \sum_j \frac{\partial C_{x_j}}{\partial b} \tag{2.34}$$

where $C_{X_j}$ is the chosen cost function calculated for each input $X_j$, $w$ are the weights, $b$ are the biases, $\eta$ is the learning rate, $m$ is the number of images in the mini-batch, and the sums are over all training examples $X_j \in X$, where $X \subset x$.

A further improvement can be made which is called stochastic gradient descent with momentum. The idea is that by adding this momentum hyperparameter to the update equation, this adds the influence of previous iterations to help guide the search in a more optimal direction. Momentum can be any chosen value between 0 and 1, where a momentum of zero is equivalent to stochastic gradient descent without momentum. This is used to add a fraction of the previous gradient to influence the direction of the next iteration. So stochastic gradient descent can be simply described as $x = x - \frac{\eta}{m} f'(x)$ then stochastic gradient descent with momentum is described in equation 2.35 for iteration $t$, where momentum equals 0.2 [60].

$$x^t = x^{t-1} - change(x^t) \text{ where } change(x^t) = \frac{\eta}{m} f'(x^{t-1}) + 0.2 \frac{\eta}{m} change(x^{t-1}) \tag{2.35}$$

The back propagation algorithm is an important algorithm based on four equations and it is used to compute the gradients of a cost function, $C$. Typically a cost function would be combined with the back propagation algorithm. There are many different cost functions which could be used to speed up the learning process besides quadratic, such as the cross-entropy cost function, the log-likelihood cost function and their regularised versions.

**Definition 6.** *The back propagation algorithm:*

1. *Input $x$ - set the corresponding $a^1$ value for the input layer, where $a^l$ denotes the activations for layer $l$ of all $L$ layers*
2. *Feedforward - for $l = 2, 3, ..., L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$*
3. *Output the error $\partial^L = \nabla_a C \odot \sigma'(z^L)$ where $\odot$ is the element-wise product of two vectors*
4. *Backpropagate the error for each $l = L-1, L-2, ..., 2$, computing $\partial^l = ((w^{l+1})^T \partial^{l+1}) \odot \sigma'(z^l)$*
5. *Output - the gradients of the cost function:*

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \partial^l_j \text{ and } \frac{\partial C}{\partial b^l_j} = \partial^l_j$$

*where the matrix $w^l_{jk}$ is the weight for the link between the kth neuron in the $l$-th$-1$ layer and the jth neuron in the $l$-th layer.*

**Overfitting and Regularisation** Overfitting is one of the biggest problems that neural networks face. Overfitting is to model every single detail meaning that any noise present is included [54]. Figure 2.36 shows the test data in black, the validation data in red circles and three polynomials of 2, 3 and 10 degrees [59]. Whilst the polynomial of degree 10, in green, has the smallest training error, it is overfitting. The polynomial of degree 3, in blue, actually has the smallest validation error so this is the model that would be chosen.

One of the best ways to reduce overfitting is to increase the size of your training data set. However there are many techniques that can be used, either where expanding the data set is not possible or to reduce overfitting even further:

- Regularising the cost function by adding on a 'regularisation term', e.g. L1 or L2 regularisation.
- Dropout - this modifies the network itself by temporarily dropping a random half of the neurons then running the algorithm and repeating this

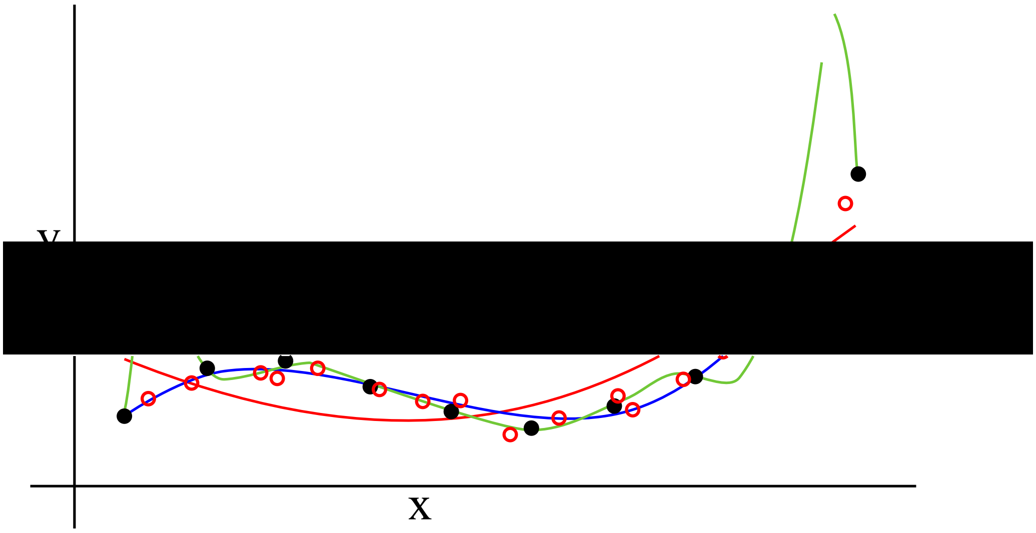

[redacted]

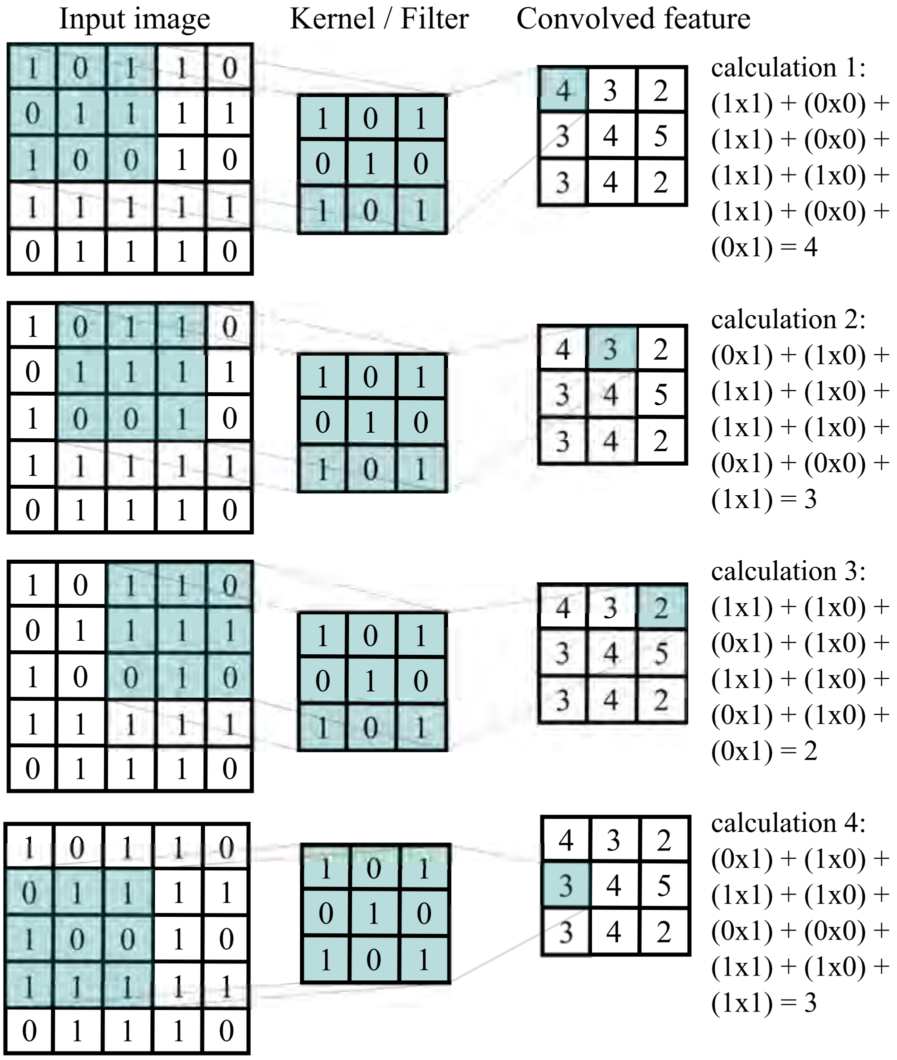
FIGURE 2.36: An example of overfitting with polynomials of degrees 2, 3 and 10 [54].

cycle for different random halves of the network. This ensures the system is robust if there is a loss in data.

- Artificially expanding the training data by making small rotations and alterations.
- Early stopping of training will also stop overfitting.

**Convolutional neural networks (CNNs)** CNNs work specifically with image inputs. Local receptive fields, shared weights and pooling are the three ideas that make up the idea of CNNs. Local receptive fields is the idea that only a small region of the input layer goes to each neuron in the hidden layer; this map is called the feature map. We then use the same weights and biases from its local receptive field for every hidden neuron in the layer. A pooling layer is then typically used after convolutional layers to simplify the outputs by determining whether a feature is in that region of the image. Convolutional layers are not as easily affected by overfitting so their use is one of the many ways to aid training.

Figure 2.37 shows an example of how a convolutional layer moves through an image and shows the first four calculations in a convolution layer. In this example the input image is a 5x5 pixel sized image convolved using the 3x3 kernel specified in the figure, with a stride of 1 pixel, to give the output image of size 3x3 shown in the figure. The kernel is moved across the image, starting at the top left corner of the image and moving to the right by the stride length until it reaches the right side of the image, it then moves back to the left of the image and down by the stride length and carries on moving across the width of the image, etc. until it reaches the bottom right of the image. Calculations are done at each location on the input image that the kernel stops at, to produce the values seen in the output image.



calculation 1:
(1x1) + (0x0) +
(1x1) + (0x0) +
(1x1) + (1x0) +
(1x1) + (0x0) +
(0x1) = 4

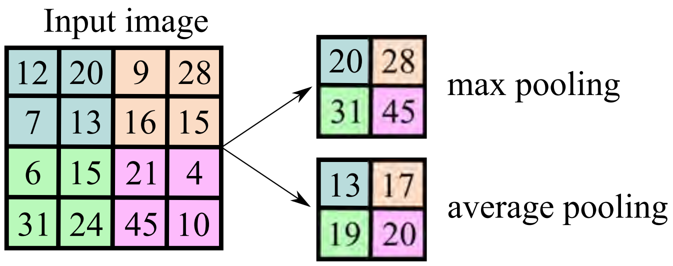
calculation 2:
(0x1) + (1x0) +
(1x1) + (1x0) +
(1x1) + (1x0) +
(0x1) + (0x0) +
(1x1) = 3

calculation 3:
(1x1) + (1x0) +
(0x1) + (1x0) +
(1x1) + (1x0) +
(0x1) + (1x0) +
(0x1) = 2

calculation 4:
(0x1) + (1x0) +
(1x1) + (1x0) +
(0x1) + (0x0) +
(1x1) + (1x0) +
(1x1) = 3

FIGURE 2.37: An example of how convolutions move across an image and how the first four values can be calculated.

Pooling layers use the same method to move across an image, but instead of applying kernels, they apply either the average or maximum value of the kernel. An example of this can be seen in Figure 2.38.



| | | | |
|---|---|---|---|
| 12 | 20 | 9 | 28 |
| 7 | 13 | 16 | 15 |
| 6 | 15 | 21 | 4 |
| 31 | 24 | 45 | 10 |

| | |
|---|---|
| 20 | 28 |
| 31 | 45 |

| | |
|---|---|
| 13 | 17 |
| 19 | 20 |

FIGURE 2.38: An example of max and average pooling.

## 2.6 Visual inspection applications

Automated inspection systems have a wide variety of uses. In manufacturing they are used to detect defects in products [61, 62, 63, 64, 65, 66, 67]; they can be used to recognise faces or body part movement [68, 69, 70, 71, 72] which could be used in games, autonomous driving systems, or law enforcement; they can be used in biometric scanning for identification verification [73, 74, 75]; they can be used for license plate and vehicle detection [76, 77]; in medicine they can be used to detect disease [78, 79, 80].

### 2.6.1 Visual inspection in steel production

This section discusses visual inspection in steel production and existing work that has been carried out in this area. There are two substantial reviews into visual inspection in steel production; one by Neogi et al. [81] in 2014 and another four years later by Sun et al. [82] which describes itself as a supplement to the first. The conclusions from these reviews are as follows.

- The design of lighting and camera systems in steel mills is of great importance due to the environmental challenges faced.
- Steel surface defects are highly irregular in shape and characteristics depending on the mill, the material type, variable lighting, and capture noise.
- Commercially produced automated inspection systems require continuous collaboration between manufacturer and user to adapt the installed system as characteristics change.
- Despite its complex setup, line-scan cameras appear to be the camera of choice as they are faster; although this fact may change as camera technology develops.

- An expectation that LEDs will become the new light of choice.
- An increase in the fusion of multiple technologies to accurately detect unbalanced defect types.
- Typically the detection process is split in two, with feature extraction happening and then classification follows. Using deep learning methods combines these two steps which is simple and produces high accuracy in detection. The biggest disadvantage of deep learning methods is that they require large amounts of data to train on in order to achieve reasonable results. There is an expectation that further work will develop neural networks which achieve good results with significantly less training data.
- It is vital to find the balance between detection accuracy and time taken for detection as in industry detection needs to happen in line with production speeds.
- A standard data set and protocol should be set out to allow fair comparison between methods.

These reviews suggest that using computer vision in steel production is a popular research area and that good success rates are possible. A 2017 paper /citenneogietal achieved a 94% accuracy using an adaptive thresholding method to detect blisters in steel. A paper by Xu et al. [83] achieved a 98.3% accuracy detecting defects on a set of cold-rolled steel images by developing the RNAMlet method which uses wavelet transforms. Hu et al. [84] used an optimized SVM method on 5 defect types and obtained an average accuracy of 95.04%. Whilst steel companies themselves do not publish their methods and accuracy rates, the research suggests that many of these methods may perform well.

### 2.6.2 Tata Logical Release system

Before installation of the TLR, Tata used visual inspection systems which capture images using cameras on the production lines, processed these images and classified any features detected. A person then had to decide whether the feature was acceptable, what should happen to the coil and where it should go next.

The TLR system is a novel programming framework written and updated inhouse over the past several years. It was a system created out of need as no commercial product has the same abilities as the TLR and can be added to and developed in the same manner. The system is designed to work with all of the different process data sources, making inspection more efficient and enabling more efficient communication between processing units. There are currently 11 installations of TLR across Tata in the UK and the system has many functions to aid production:

- The ability for an inspector to add defect markers on to the coil for other departments to see.
- To manage coil standards and customer requirements.
- To create custom reports at set time intervals.
- Create a feature map of the coil with feature locations as well as being able to view the individual feature images.
- Smart rules allow for customer specifications to be adhered to, with TLR giving a decision for each coil, using a grading system, as to whether it fits specification or requires further work.
- A feedforward system which allows correct instructions to be forwarded to delivery units and to customers.
- The ability to track and monitor features through processes.
- The ability to run executable files written in-house from within TLR.
- Storage of all data from the many sources used.
- Allows for composite, split and slit coils to be made by synchronizing their feature information into relevant sections.

An article in IOM3's Material's World June 2020 edition [85] discusses the latest progress made with TLR and the quality inspection department at Tata Steel in South Wales:

> By 2018, a feed-forward system from the hot mill to Trostre pickling line had been established. This feed-forward of coils digital twins allowed the site to optimise its pickle line operations. The engineers processed the hot mill coil data against their own customer and operational requirements to optimally match coils to orders and operation paths. As a second proof-of-concept, feed-forward data could now be used to control production lines.

Further work between surface and process specialists led to the creation of an object-oriented program that combined expert human input with a coil digital twin and a rework optimiser algorithm. This creates an optimised rework path for each coil with surface quality issues between our galvanising and coil inspection lines. This program is capable of reading feature location data and then stopping the production line to within one metre of a feature, so that the attribute is on the inspection bed. This means an inspector can confirm the type and severity, and approve or adapt the rework path suggested for each coil by the program.

Whilst the program currently only communicates and optimises between two production lines, the adaptable way in which it is written opens the doors to feeding data forward from even further back in the production process. The eventual plan is to carry out this real-time, feed-forward and rework optimisation from casting to dispatch.

# Chapter 3

# Steel Defect Detection

## 3.1 Introduction

This chapter discusses existing work using Gabor filters in steel defect detection. It then proposes a method using Gabor filters and histograms for feature extraction and a random forest classifier to detect the presence of steel defects on cold-rolled matte steel coils. The final case-study section considers what combination of these feature extraction steps produces the most suitable method.

Surface quality inspection is a vital part of the steel manufacturing process as damaged steel cannot be sent to customers. Many improvements occur when moving from manual to automated inspection systems. In 2012 a review of manual inspection methods [86] found that human visual inspection typically has a defect detection rate in the range 9 - 100% with most rates in the 50 - 70% range, whereas in the 2016 round of the ImageNet competition for object detection, the winning team had an error rate of just 2.9% [87]. Some defects can also look similar, meaning that two inspectors may not agree on what type a defect is. Although human visual inspection is an option used in the past, modern quality inspection systems are a far more reliable and accurate choice due to the fact they do not suffer from distractions, confirmation bias, or any of the many other reasons that humans make mistakes.

Unlike mass production of materials such as paper, steel production is quite an environmentally challenging process. In certain sections of production, the air around the cameras has water and particulates in it which can make it difficult to capture clean images. Despite the challenging environment, computer vision methods are increasingly popular in the steel industry [81]. There is substantial research in this area which is typically based on variations of thresholding and edge detection methods [88, 62, 89, 90, 91, 92, 93]. The use of texture analysis with co-occurrence matrices and probabilistic neural networks has been suggested in [94], and [95] proposes a similar method which has an overall accuracy of 91.67% when detecting surface defects.

Steel defect detection using Gabor filters and thresholding is a commonly explored method. Work undergone by Yun et al. [88] detects cracks in steel uses Gabor filters and thresholding with some success; using 1328 steel images, they achieved 91.9% accuracy when detecting thin cracks and 93.5% accuracy when detecting corner cracks. Choi et al. [96] uses this method along with morphological features to propose a pinhole detector which was tested on 1764 images, only 116 of which featured the defect, and achieved 87.1% accuracy in correctly classifying the images. Sadeghi et al. [97] use Gabor filters and thresholding to detect defects on steel plate and, whilst very little information is given on the data set, the algorithm achieved 93.14% accuracy on the positive images tested. Whilst thresholding works well in predictable, consistent images, the method described in this section uses a machine learning method to detect gross defects instead of pre-determined thresholding as the algorithm can adapt to any unexpected variation in new images.

Many object detection and classification algorithms perform well on the data they were optimised for, but it is unknown how they perform when applied to identified steel defect data from Tata Steel Europe. There are journal papers testing various methods on specific types of steel defects, where the images come mostly from mills in Asia. However every steel mill has a unique configuration of cameras and equipment meaning inspection system performance will vary.

The research in this chapter takes a similar approach to the Deng at al. paper [98] on using random forest, support vector machine (SVM), and neural network classifiers for eye disease detection. In their study, features were extracted from the image set using a Gabor filter bank and a non-linear energy transformation and then three classifiers were trained using histogram feature descriptors. The random forest classifier was trained with 50 trees and achieved 94.7% accuracy. The SVM and neural network achieved 97.1% and 84.7% accuracy. The retina images, acquired by optical coherence tomography (OCT), used in the study were grayscale images, much like the steel surface images which are used in this research.

There are several purchasable detection systems available, such as ISRA Vision Parsytec [99] who provide surface inspection systems for metal and paper production, Cognex [100] who provide a wide range of automation systems in sectors such as automotive, medical, and food production, and Ametek [101] who provide vision systems for material surface quality inspection. The issue with purchasing a blackbox system is that the methods and finer controls remain a company secret and any work on the product will need to be carried out by their engineers. However creating one's own system within a company has many benefits, such as unlimited scope of where and how these systems could be applied. There is also the added factor of self-reliance if a system has been created internally, although this does come with the cost of hiring and training a specialist team. Any issues can be dealt with instantly, by experienced creators of the system who fully understand the data, without spending resources, whether time or money, on outsourcing a fix.

## 3.2 Review of methods

This section reviews the methods used in this chapter. It discusses Gabor filters, decision trees and random forest classification, and histograms. Quantifying accuracy is discussed and confusion matrices are explained as a measure of accuracy.

### 3.2.1 Gabor filters

This section discusses Gabor filters which are one of the many types of filters used in image processing. First discovered in the 1940s, Gabor filters are generated by multiplying a complex exponential by a Gaussian function to create a special kind of wavelet which can be seen in definition 7 [102]. Since their inception it has been shown that Gabor filters can be used to accurately model the human visual system as the filters capture the same fundamental visual properties [102]. This makes them highly suited to image analysis problems.

In recent years Gabor filters have been used widely in computer vision and texture analysis problems, with applications in object recognition, texture segmentation, and face detection [103, 104, 105, 106, 74, 69, 107, 70, 108, 77, 109]. There has been limited research into steel defect detection using Gabor filters and most of these methods use thresholding as the detection method [97, 89, 90, 96].

**Definition 7.** *A two-dimensional Gabor filter [68] is defined as:*

$$F_{Gabor}(x,y) = \frac{\mathcal{F}^2}{\pi\gamma\zeta}\exp\left(-\left(\alpha^2x'^2+\beta^2y'^2\right)\right)\times\exp(\imath 2\pi\mathcal{F}x') \qquad (3.1)$$

*when $x' = xcos\theta + ysin\theta$ and $y' = -xsin\theta + ycos\theta$, where $\mathcal{F}$ is the central frequency of the sinusoidal plane wave, $\gamma = \frac{\mathcal{F}}{\alpha}$ and $\zeta = \frac{\mathcal{F}}{\beta}$ are the spatial aspect ratios for the major and minor axis, $\theta$ is the anti-clockwise rotation of the Gaussian and the plane wave, $\alpha$ and $\beta$ are the sharpness of the Gaussian along the major and minor axis, parallel and perpendicular, respectively, to the wave.*

As a filter, Gabor filters are a combination of a sinusoidal wave and a Gaussian envelope. Figure 3.1 shows how to visualise the filter in a two-dimensional and three-dimensional space.

A Gabor filter bank is a collection of Gabor filters with different scales and orientations. Unlike other types of filter, a bank of Gabor filters are used to extract

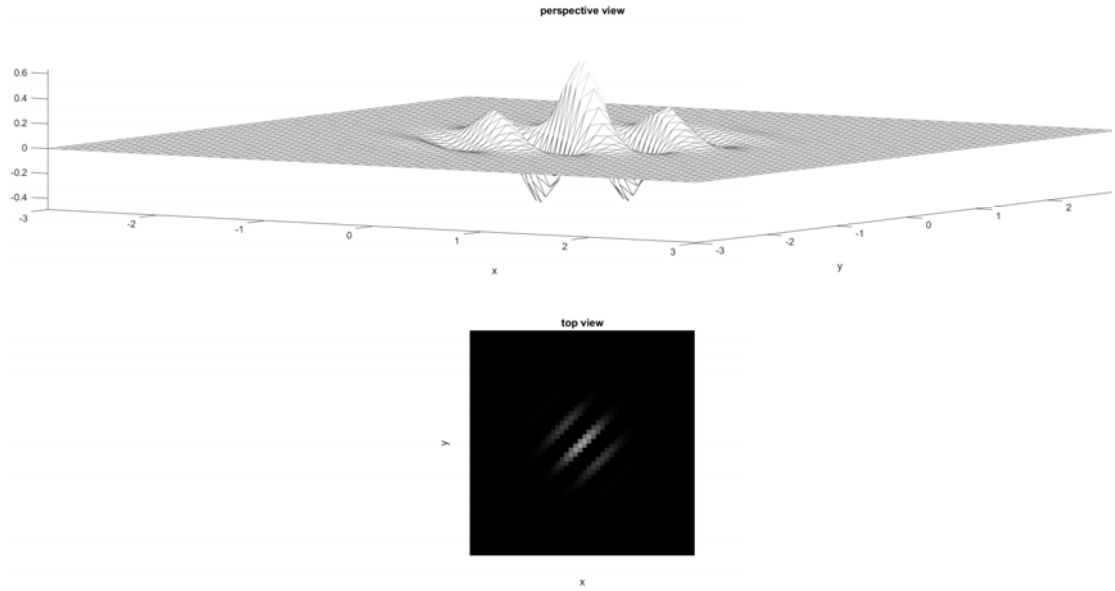


FIGURE 3.1: A visualisation of a Gabor filter in 2-D and 3-D.

both spatial and frequency image information including multiple orientations and frequencies of textures [102]. This means Gabor filters are useful in detecting a wider variation of textures with less computational cost.

Based on the method in J. Deng et al. [98], initial trials used images which were filtered using the Gabor filter code featured in M. Haghighat et al. [110], and then an energy transform and averaging convolution using kernel size $n$ was applied as described in equations 3.2 and 3.3, where $F_{Gabor}$ is the Gabor function, $W_{x,y}$ is the averaging window with size $n \times n$ pixels and centered at image coordinate $(x, y)$.

$$\psi(F_{Gabor}) = \tanh(tF_{Gabor}) = \frac{1 - \exp^{-2tF_{Gabor}}}{1 + \exp^{-2tF_{Gabor}}} \tag{3.2}$$

$$\varepsilon(x, y) = \frac{1}{n^2} \Sigma_{(i,j) \in W_{x,y}} \, |\psi(F_{Gabor}(i, j))| \tag{3.3}$$

This Gabor filter code was later substituted for the Matlab functions `gabor` and `imgaborfilt` which apply the same fundamental methods, but does not apply zero padding around the image. Both of these implementations of filtering produce the feature vectors, of which histograms are taken to run through the classifier for training. Padding is a method which is used to make an image size divisible by the filter or kernel size such that convolving a filter or kernel across an image captures the values of every pixel. This is done by adding the required number of lines of pixels around all sides of the image. In the case of zero padding, it is the number zero which is used as the intensity values in these new lines of pixels.

An example of a Gabor filter bank can be seen in Figure 3.2. To generate a Gabor filter bank in Matlab the `gabor` function is used with inputs 'wavelength' and 'orientation'. The wavelength is specified as pixels per cycle and can be any number greater than 2. In the example filter bank below, 16 filters were generated at four different orientations and four different wavelengths. The code used to generate this filter bank, b, with wavelengths of 5, 10, 20, 40 pixels per cycle and orientations 0°, 45°, 90°, 135° reads:

```
wavelengths = [5, 10, 20, 40];
orientations = [0, 45, 90, 135];
b = gabor(wavelengths, orientations);
```

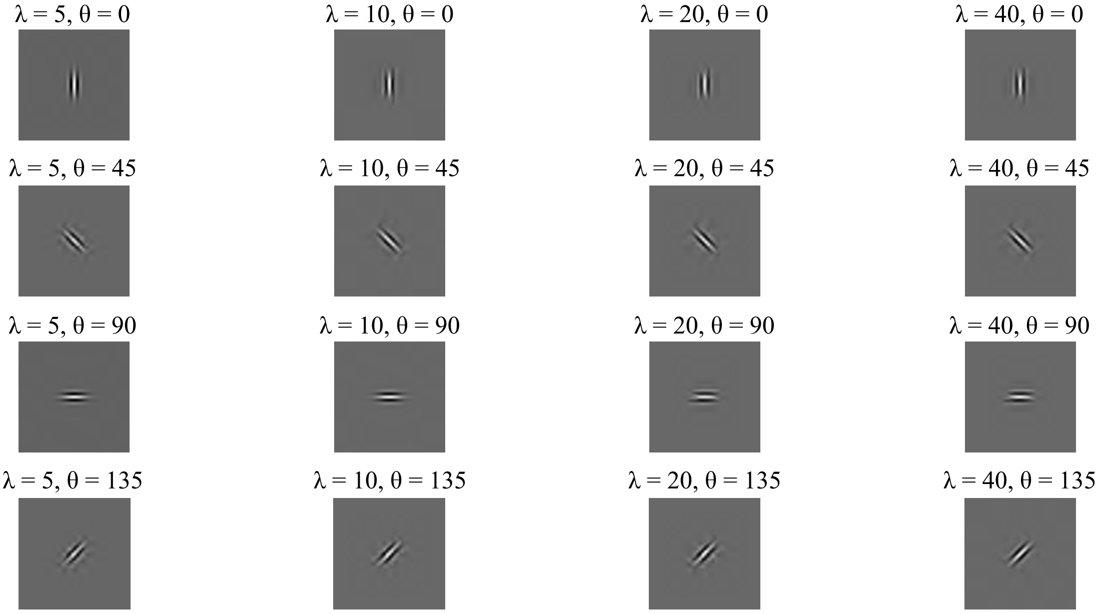


FIGURE 3.2: An example of a Gabor filter bank containing 16 filters with wavelengths [5, 10, 20, 40] at orientations [0°, 45°, 90°, 135°].

The size of the filter increases as the wavelength increases, meaning a smaller wavelength will pick up smaller details and a larger wavelength will miss small details, but pick up larger details. You cannot see this increase in filter size in Figure 3.2 due to how Matlab fills the space in subplot images, however Figure 3.3 shows how this wavelength increase just looks like larger versions of the

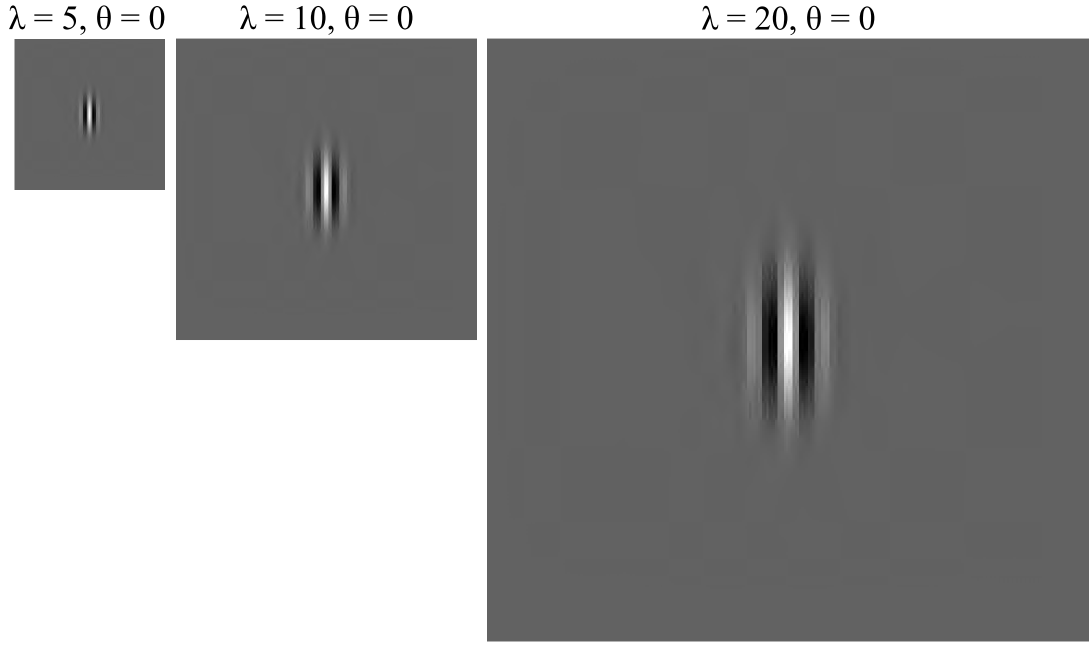


FIGURE 3.3: An example showing how the wavelengths increase in size.

filters. The orientation can be anything between 0 and 180 degrees, so the orientations 0, 45, 90, 135 degrees were selected to filter the image from a selection of angles to minimise information loss. The wavelengths were chosen based on the sizes of the features shown in the next section in Figure 3.11. Analysing the size of the defects showed that most defects were less than 25 pixels in height and width, but also that a significant amount were less than 10 pixels, meaning any wavelengths larger than them would likely miss the feature entirely. It was important to select wavelengths that would capture a broad selection of sizes, so through a process of testing wavelength sizes on some very large and very small defects, to see what could be adequately captured, the wavelengths 4, 6, 12, and 32 pixels per cycle were selected.

The code used apply the Gabor filter bank, b, on an image, I, will produce 16 filtered images, filtered_image_bank, and reads:

```
filtered_image_bank = imgaborfilt(I,b);
```

Figures 3.4 and 3.5 show the original image of a heavy lamination and the subsequent filtered images. The original image was put through a Gabor filter bank with wavelengths 4, 6, 12, and 32 pixels per cycle and orientations 0°, 45°, 90°, 135°.

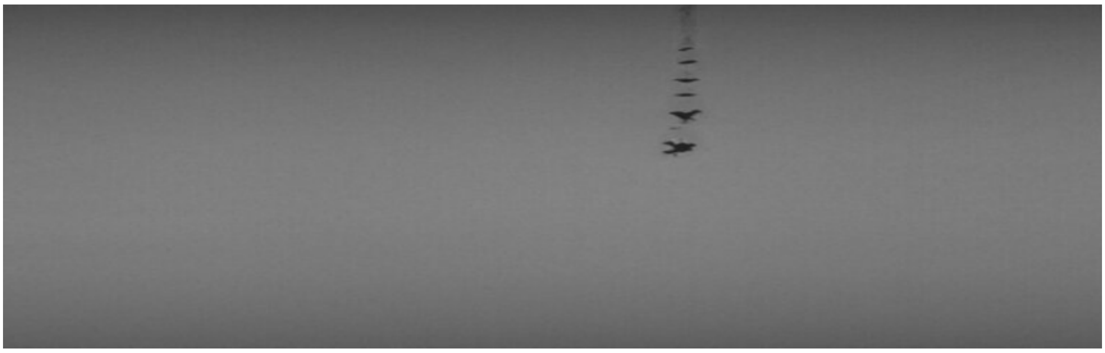


FIGURE 3.4: An image of a heavy lamination.

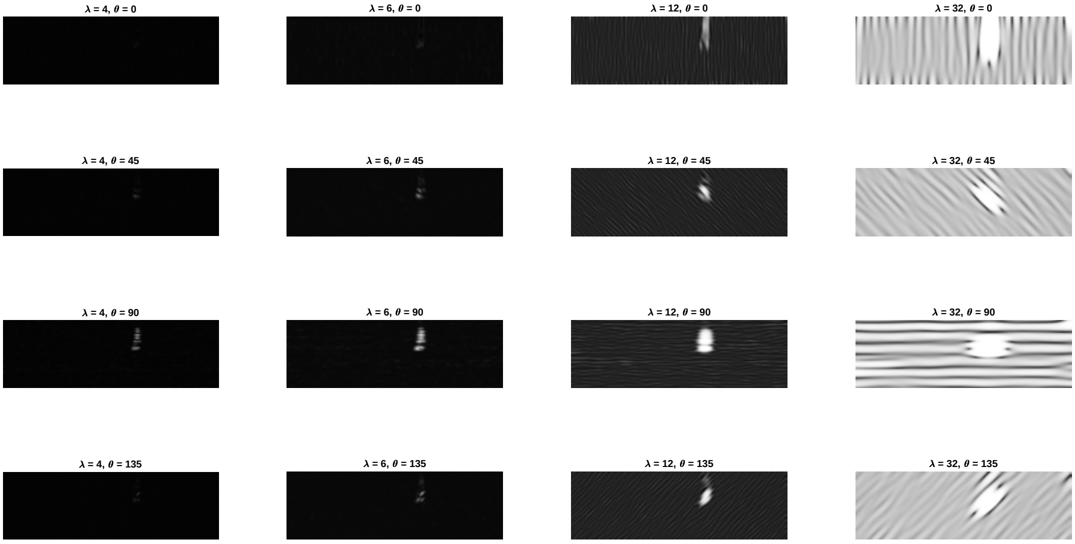


FIGURE 3.5: Figure 3.4 filtered through the Gabor filter bank.

### 3.2.2 Histograms

Histograms are a common tool used to summarise and present data. In the case of feature vectors they provide an excellent method to reduce data size whilst still representing the differences in the data. Using histograms reduces the size of each sample from 26 × 26 × 16 (=10,816) data points to just 1700 data points. This reduction in data size means that training the classifiers is much faster as there is less data to process.

Histograms work by putting data into *bins*; each *bin* has a minimum and maximum value of the data allowed to be in it and each data point which fits in a bin is counted. These maximum and minimum values can either be decided by the software or specified by the user. For example, given the data set $A = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$, bin limits 0 and 10, and the number of bins is set to 3. This produces three bins; bin one has limits $[0, 3.\dot{3}]$, bin two has limits $[3.\dot{3}, 6.\dot{6}]$, and bin three has limits $[6.\dot{6}, 10]$. Distributing the data into the bins gives:

- Bin one contains 4 data points $[0, 1, 2, 3]$
- Bin two contains 3 data points $[4, 5, 6]$
- Bin three contains 4 data points $[7, 8, 9, 10]$

The frequency of the data in the bins is shown in the histogram in Figure 3.6.

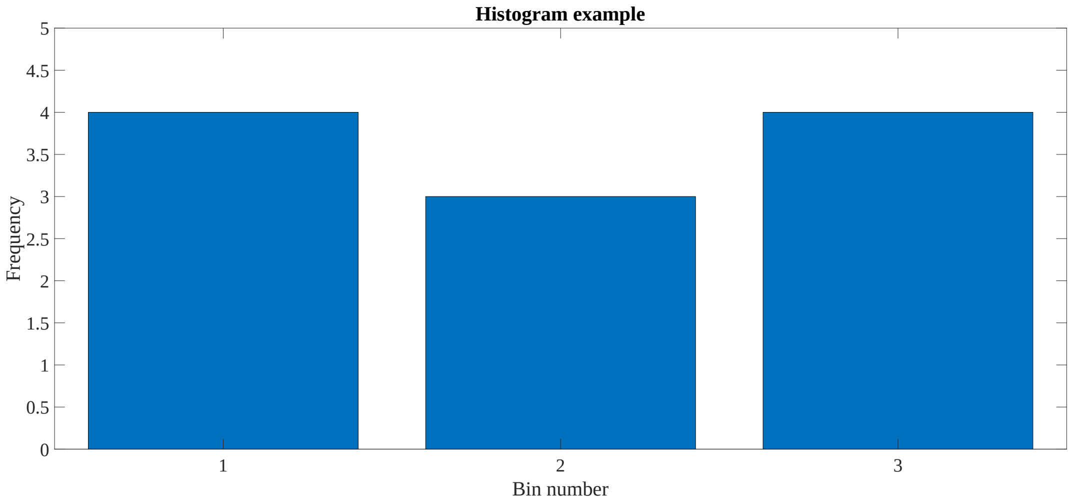


FIGURE 3.6: The histogram of data set A with three bins.

### 3.2.3 Decision trees and random forest classifiers

Decision trees are a method of rule generation which follow a similar concept to a flow chart. Each root node has a hypothesis and the splitting of the branch follows the thresholds to the leaf node which contains the class/output or to the next decision node [111, 112]. Figure 3.7 is an example of a decision tree which is trained with data which has two predictors, x1 and x2, and fits into class A or B. The black circles are leaf nodes and the triangles are decision nodes.

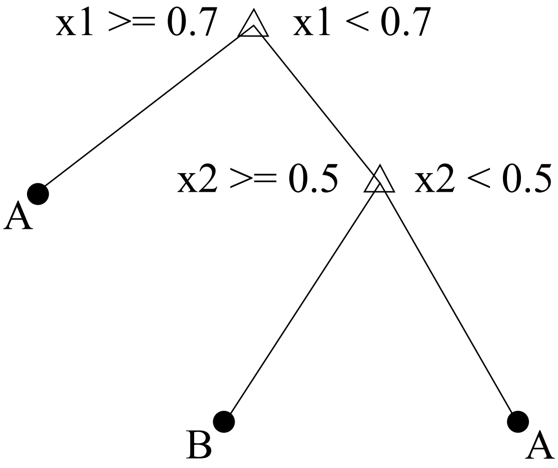


FIGURE 3.7: An example of a decision tree.

As described in the Matlab documentation [113], the algorithm to grow a decision tree is as follows.

1. Each predictor will be examined for all possible binary splits.
2. The split which achieves the most optimal splitting criterion is chosen. The choices of splitting criterion are discussed later in this section.
3. The split is implemented.
4. This is repeated recursively for the two child nodes (these are the left and right nodes coming off each decision node).

So, as an example, if the training data is the 6 lines of data in Table 3.1, this could potentially generate the example decision tree in Figure 3.7.

A random forest classifier is made up of lots of decision trees. Each decision tree is grown by using a subset of data from the training set, which is selected or sampled at random with replacement. This means that regardless of how many times a data point is sampled, it remains in the training data set and has the possibility to be sampled in all subsets of data. These trees will each run

TABLE 3.1: Decision tree example data set.

| **Predictor x1** | **Predictor x2** | **Known class** |
|---|---|---|
| 0.4 | 0.5 | B |
| 0.3 | 0.8 | B |
| 0.7 | 0.1 | A |
| 0.9 | 0.4 | A |
| 0.8 | 0.3 | A |
| 0.6 | 0.6 | B |

through the input and cast a vote on the output class where the most voted output class wins and becomes the final decision [114].

When generating a random forest classifier in Matlab, amongst other conditions, the splitting criterion and minimum leaf size can be specified. The following line of code will generate a collection of 100 decision trees which have learned from the inputs, $X$, and respective outputs, $Y$.

```
randomForest=TreeBagger(100,X,Y,’Method’,’classification’);
```

The method options are ‘classification’ or ‘regression’ so in the case of detecting steel defects ‘classification’ is necessary. The ‘minimum leaf size’ is the minimum number of observations required per leaf, otherwise a leaf will not be generated from that decision node. The default value for this is ‘1’ and increasing it will make a shallower tree.

There are three choices of splitting criterion in Matlab: Maximum deviance reduction, Gini’s diversity index and twoing rule. These criterion split the node based on the node impurity [115]. The term node impurity refers to how many items of training data end up at a leaf node which would be incorrect, so the aim is for nodes to be pure meaning that they have no error.

**Definition 8.** *Maximum deviance reduction [115]:*

$$d = -\sum_{\kappa} p\left(\kappa\right) log\left(p\left(\kappa\right)\right)$$

*where $d \geq 0$ and $p\left(\kappa\right)$ is the proportion of classes with class $\kappa$ that reach the node. A pure node has $d = 0$, otherwise the deviance, $d$, is positive.*

**Definition 9.** *Gini's diversity index [115]:*

$$Gini(T) = \sum p(\kappa)(1 - p(\kappa)) = 1 - \sum_{\kappa=1}^{\mathcal{K}} p_{\kappa}^{2}$$

*where $Gini\,(T) \geq 0$, $T$ is the training set, $p(\kappa)$ is the proportion of class $\kappa$ which occurs at a certain node in the tree and $\mathcal{K}$ is the total number of classes. The Gini index is a measure of node impurity, with a pure node having one class and a Gini index of* 0.

**Definition 10.** *Twoing rule [115]: The fraction of inputs of class $\kappa$ which split to the left child node is denoted by $L(\kappa)$ and to the right child node denoted by $R(\kappa)$. The split criterion is chosen to maximize the following equation.*

$$p(L)p(R)\left(\sum_{\kappa}|L(\kappa) - R(\kappa)|\right)^{2}$$

*where $p(L)$ and $p(R)$ are the fractions of observations that split to the left and right, respectively. The larger the expression, the purer the child nodes have become by the split. The smaller the expression, the more similar the child nodes are to each other and their parent node meaning the split did not increase node purity.*

### 3.2.4 Quantifying accuracy

There are two ways that have been used to observe the overall accuracy and efficacy of the method proposed in this chapter. The first is to visualise the results for each image, an example of which can be seen in Figure 3.8. The red box indicates the known region of interest, where the gross defects are, and the yellow boxes are all of the windows which have been predicted as containing defects.

The second is to generate confusion matrices, which compare the known answers against the predicted answers. Being able to consider both the accuracy statistics and view the images with their ROIs and predicted boxes overlaid, provides a clearer understanding of the results.

#### Confusion matrices

This chapter features many confusion matrices for the different classifiers tested. A confusion matrix is a way of evaluating the performance of a classifier in a

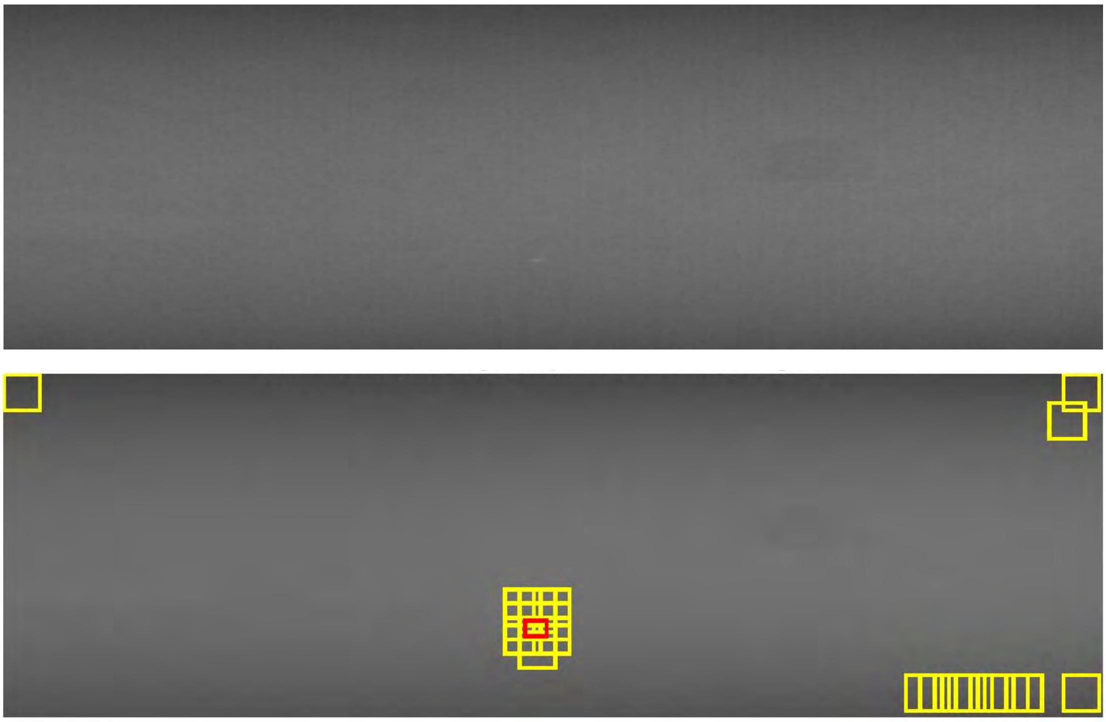


FIGURE 3.8: An example test image with the original image on top and below is the image with the gross defect in the red box and defect predictions in yellow.

meaningful way so that classifiers can be compared; it measures the performance of the classifier when new, unseen inputs are fed into it and outputs are predicted [116]. Using a matrix to display the data helps to visualize the performance of the classifier. An example confusion matrix can be seen in Table 3.2.

TABLE 3.2: An example of a confusion matrix.

| | | | | |
|---|---|---|---|---|
| Predicted Class | 0 | A<br>$a\%$ | B<br>$b\%$ | |
| | 1 | C<br>$c\%$ | D<br>$d\%$ | |
| | | $E\%$<br>$e\%$ | $F\%$<br>$f\%$ | $G\%$<br>$g\%$ |
| | | 0 | 1 | |
| | | Known Class | | |

- *0* is the class of defect-free images and *1* is the class of images with defects
- *Known Class* is the known class of an image
- *Predicted Class* is the predicted class of an image
- *A* is the number of correctly classified negative images, **true negatives**
- *B* is the number of positive images wrongly classified as negative images, **false negatives**
- *C* is the number of negative images wrongly classified as positive images, **false positives**
- *D* is the number of positive images correctly classified as positive images, **true positives**
- $a$, $b$, $c$, and $d$ are $A$, $B$, $C$, and $D$, respectively, as a percentage of all images
- Of all **true negatives**, $(A + C)$, $E\%$ are correctly classified and $e\%$ are wrongly classified
- Of all **true positives**, $(B + D)$, $F$ % are correctly classified and $f$ % are wrongly classified
- $G$ % of all predictions are correct and $g$ % of all predictions are incorrect

## 3.3 Data

This section discusses the data received from Tata Steel UK and all of the analysis and preparation of the data that was undertaken. The data provided for this section of work required some extraction and piecing together to be in a suitable form for machine learning.

### 3.3.1 Data analysis

The data was received in the form of '.jpeg' images and accompanying '.csv' files with defect information. The images received were taken from cameras set to 60mm from the strip surface and set up as either vertical for dark field or 17 degrees off vertical for bright field. Each pixel represents 0.24mm of steel surface and the images are size 768 pixels wide and 240 pixels high. Given that every image in this data set had not been formally checked and had their defect type confirmed, a lot of processing was required to create sets of training and testing images. There were five questions that needed to be answered before any feature generation could begin:

- Which defects occur most frequently?
- How many clear, non-confusing images of each defect type do we have?
- Has each image been classified correctly?
- If not, do any need discarding or putting in the correct class?

The frequency of defects, seen in Figure 3.9, in the whole data set was recorded to generate a preliminary list of defect types. Any defect types which had less than 500 clear, correctly classified images were discarded from the list. Weld holes and seams were also discarded as they are not defects, only markers. This left a list of 23 defect types, three of which were later excluded. The names and definitions for defects can vary according to different companies; the details of the final 20 defect types are detailed in Table 3.3.

Initial testing was carried out using these 23 types of defects, although upon further conversations with colleagues at my sponsor company, the defects which pick up as patterns (such as large stains) and edge images which contain sections of the rollers were excluded from the data sets, as these are treated separately. Section 3.3.2 discusses the issues with edge images and the reasons for excluding them from this trial. This reduced the data set to 2000 training images and 72 test images.

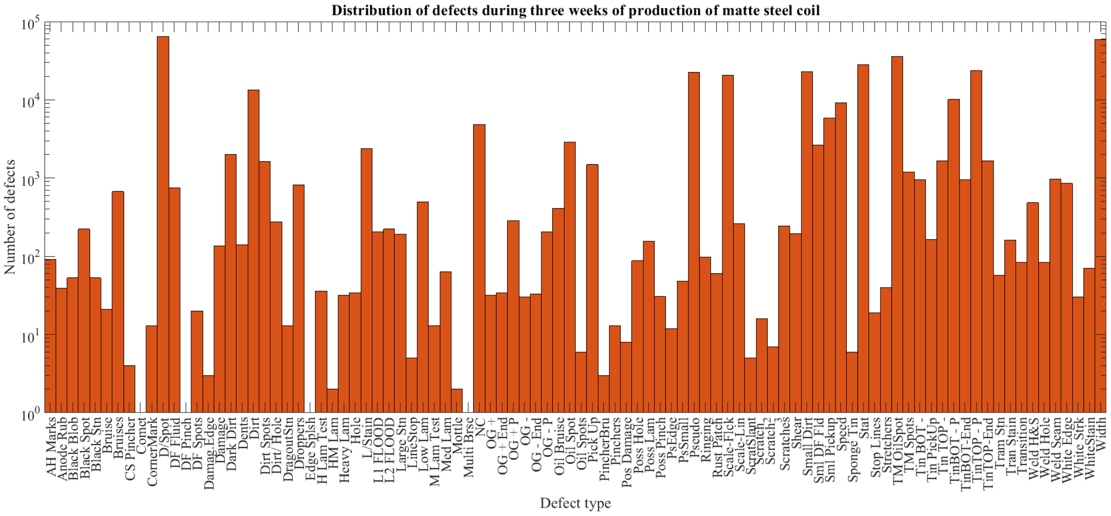


FIGURE 3.9: Frequency of defects which occurred over a three week period of production of matte steel coil (full figure available in appendix B).

Once a definitive list of defects was established, the frequency of these defects was looked at to determine any bias that might need to be considered. Figure 3.10 shows the frequency of the 20 defect types in the data set. It is clear there is a large amount of dark spot detected over any other defect. To ensure there is no bias toward certain types of defects, equal amounts of each class were used in the training and test sets.

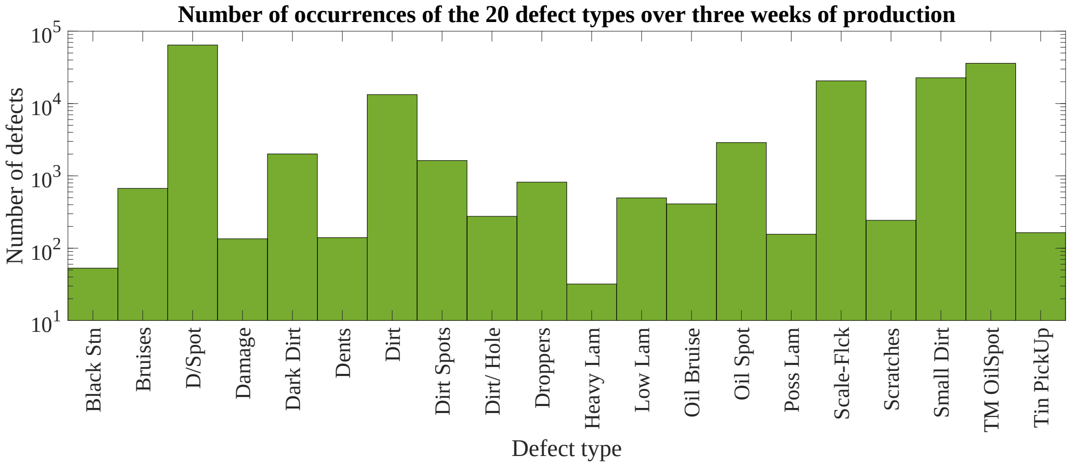


FIGURE 3.10: Frequency of the top 20 defects which occurred over three weeks of production.

TABLE 3.3: Defect types and descriptions of the 20 defect types identified.

| **Defect type** | **Description** |
|---|---|
| Black stain | Stains caused by debris in the coating tanks |
| Bruises | Bruised patches of steel caused by damaged rolls |
| Damage | Damage with an unknown source |
| Dark dirt | Dirt gets onto the strip from many different sources |
| Dents | Caused by crystallised electrolyte from the coating tank sticking to the rolls or roll damage |
| Dirt | Dirt from various sources |
| Dirt/hole | Dirt which could also be a hole |
| Dirt spots | Multiple dirt spots |
| Droppers | A chemical treatment stain caused by a build up of acid which can flake off and destroy the tin coating |
| Dark spot | Possibly dirt or a very small stain |
| Heavy laminations | Laminations that have been detected on both sides of the strip so could possibly have a hole |
| Low laminations | Laminations that don't feature on both sides of the strip or have not broken the surface of the strip |
| Oil bruise | Grease or oil which drops onto the strip, sticks to rollers and rolls into strip  5 cm diameter |
| Oil spot | Grease or oil which drops onto the strip, sticks to rollers and rolls into strip  1 cm diameter |
| Possible lamination | A pseudo class, could also be a number of other defect types |
| Scale | Caused by not washing off properly or by the dropout temperature being too high in the hot mill |
| Scratches | If they appear shiny in the dark field images then they are new, if they appear dark in the bright field images then they are from a previous production stage |
| Small dirt | Small dirt from various sources |
| Tin pickup | Tin deposits from tank debris |
| TM oil spot | The same as oil spots, only from the temper mill |

The window size 26 × 26 pixels was chosen by plotting a histogram of all the defect sizes and determining the most common sizes that occurred. The widths and heights of the regions of interest labelled for each defect are shown in Figure 3.11. Over 90% of the heights were within 0 to 25 pixels and over 90% of the widths were within 0 to 30 pixels. The average width was 20 pixels, so 26 × 26 pixels is an appropriate window size for most defects, without being too large and still being divisible by 2 to allow downsampling if required. For real-time decision making there must be a balance between accuracy and computational speed. Choosing a window size too small would be computationally expensive, but choosing one too large will lower the performance of the classifier on smaller defects.

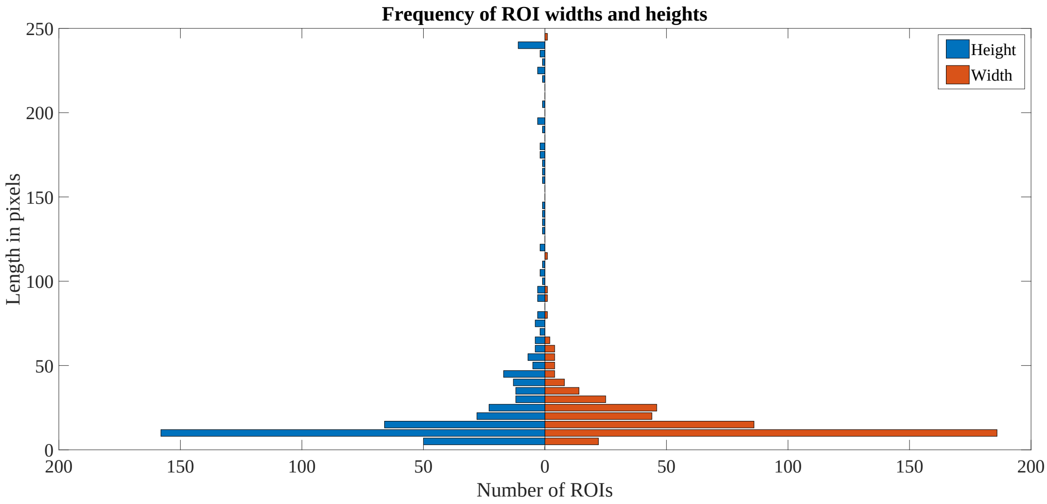


FIGURE 3.11: Histogram of ROI widths and heights.

### Exploration of aspect ratios of regions of interest

Splitting the defects into groups according to their shape and aspect ratio of the image was considered. Three clear types of image were defined; tall and thin, square-ish, and tall and wide. The majority of the defects fit in the square-ish group, with them being split by aspect ratios of range $[0, 0.5]$, $[0.5, 1.5]$ and $[1.5, 16]$. The frequency of these shapes of defects in the whole data set can be seen in Figure 3.12. For all material types there were 21167 tall and thin defects, 103321 square-ish defects, and 42599 short and wide defects. It was decided that whilst this method could be possible with extensive data processing, forcing the defect images to conform to a box could hinder a classifiers ability to accurately detect images in real size.

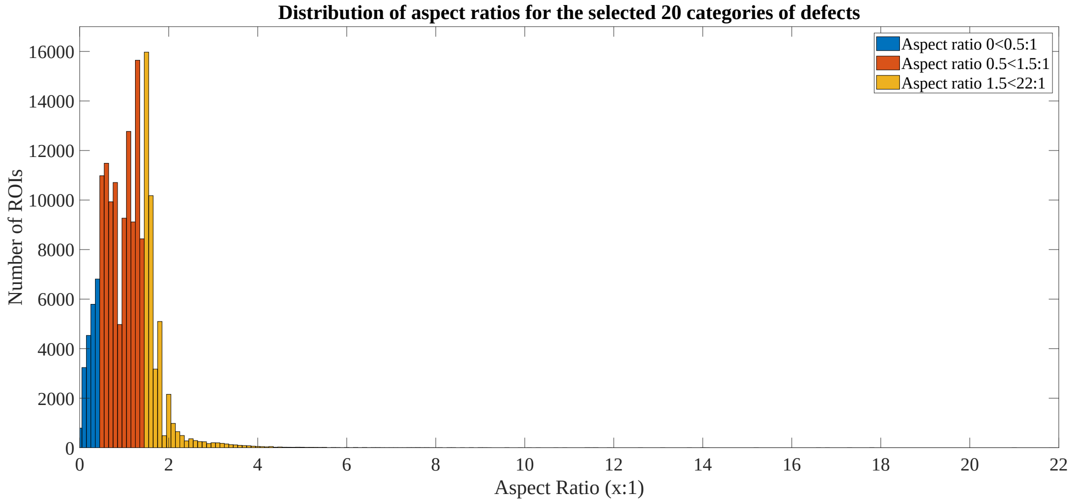


FIGURE 3.12: Distribution of aspect ratios for the 20 categories of defects.

### 3.3.2 Data preparation

The data set consists of 100 images of typical examples of each of the 23 types of defect were hand-selected, making a defect training data set of 2300 images. This data set was later reduced to 20 defect types, making a set of 2000 images. Given more time and data, it would be possible to build on this data set to include more examples of each class and further defect types, but 100 images of each type was considered an acceptable data set size based other research in the area. Table 3 "Comparison of defect detection systems" from the Neogi et al. [81] review of vision-based steel surface inspection systems lists 19 studies and of those which specify their sample size (18 of them) the mean sample size was 2125 images. From the initial set of 2300 images, sample patches of both defect areas and clean steel were selected from the images ensuring an even spread of samples from all over the images and defects. All patches were size 26 × 26 pixels, as discussed in section 3.3.1. For defects larger than the patch size, random areas of the defect were selected for the patches, an example of this can be seen in Figure 3.13. Selecting the clean steel patches required some care as they must be high quality and have no features present. Because of this, there are some images which only had defect samples taken from them, whilst other images had multiple clean steel samples taken from them.

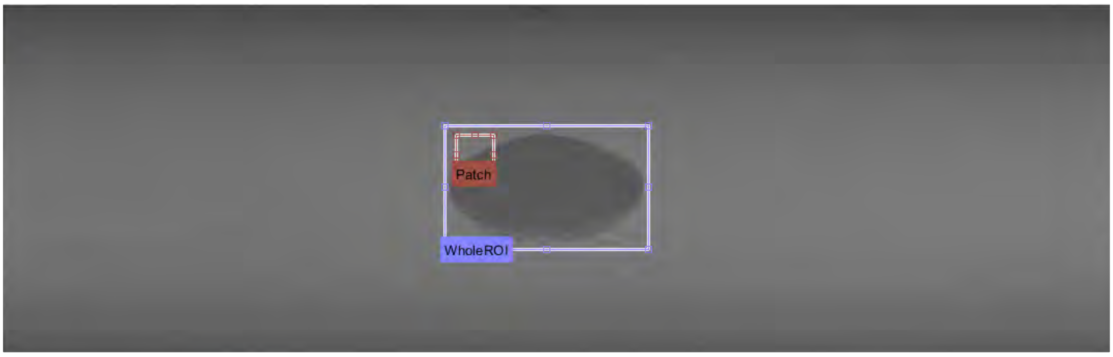
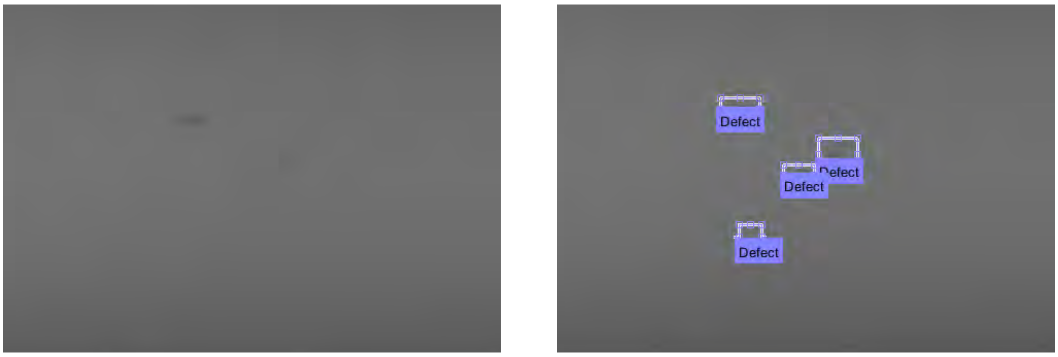


FIGURE 3.13: Labelling of whole ROIs and patches for training images.

The initial test set of images included 5 further images of each defect type, meaning the test set had 115 images in total. After excluding edge images, this test set was reduced to 72 images. All defects within the image were labelled, an example of which can be seen in Figures 3.14a and 3.14b. The labelling of these images has to be thorough as any missed defects in an image will have an effect on the accuracy percentage of the method.



(A) Unlabelled test image.

(B) Labelled test image.

FIGURE 3.14: Labelling all ROIs in test images.

### Issues with edge images

The initial method of generating the training data cropped out the patches and then filtered these patches using the Gabor filters code [110] discussed in section 3.2.1. This method led to very poor training results and the initial results showed that areas of edge images where the rollers were showing were being classed as defects. Edge images refers to an image where the strip edge ends and the rollers begin; an example of an edge image can be seen in Figure 3.15. A small study on what to expect from these dark areas was carried out by visually analysing two dark patches. One was a dark patch taken from the rollers part of an edge image. The other was created using the `ones` command in Matlab [3] which creates a matrix of specified size, with the value 1 in each position. This matrix was then multiplied by 7 to replicate the rollers section of an image; as the intensity values in dark patches are typically uniform and have grey values in the $[5, 10]$ region. These two patches were then filtered and compared.

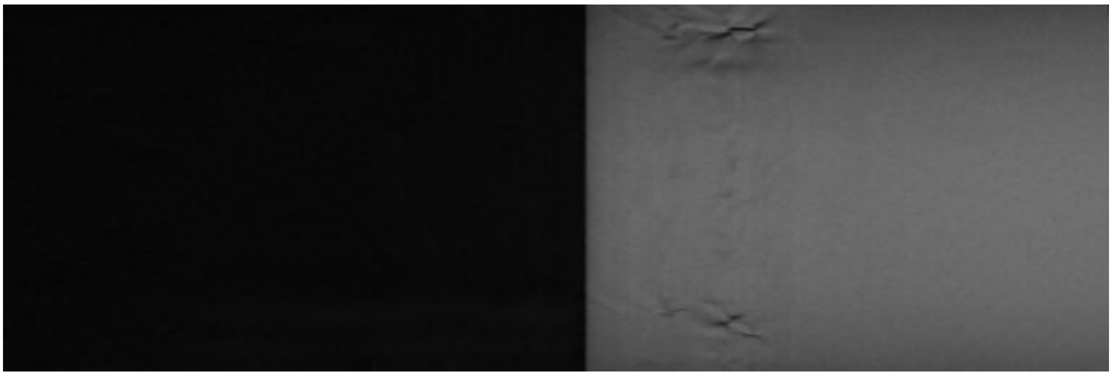


FIGURE 3.15: An example of an edge image.

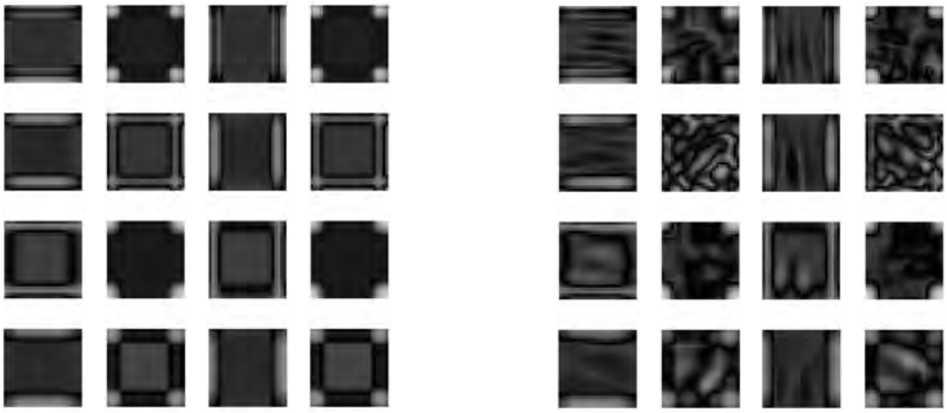
Figures 3.16a and 3.16b are the original patches. There are no obvious differences between them. Figures 3.17a and 3.17b show the responses when put through a Gabor filter bank of 4 scales and 4 orientations with a filter size of $12 \times 12$ pixels, followed by the energy transform. Figure 3.18 is the respective histograms of the intensity values, followed by histograms of each of the filters, for each patch.

(A) Generated patch of intensity value 7.

(B) Cropped patch from image.

FIGURE 3.16: Comparing two patches; one created and one cropped.

From this initial testing it was clear something was not quite right. These responses from each patch should be somewhat similar, if not nearly identical. However, the patch taken from the image is too noisy which adds confusion as it looks like a defect is present. The square borders around Figure 3.17a were also unexpected, and likely due to the zero padding added using this particular Gabor filter code.



(A) Filter responses for created patch.

(B) Filter responses for cropped patch.

FIGURE 3.17: Comparing Gabor filter responses for two patches; one created and one cropped.

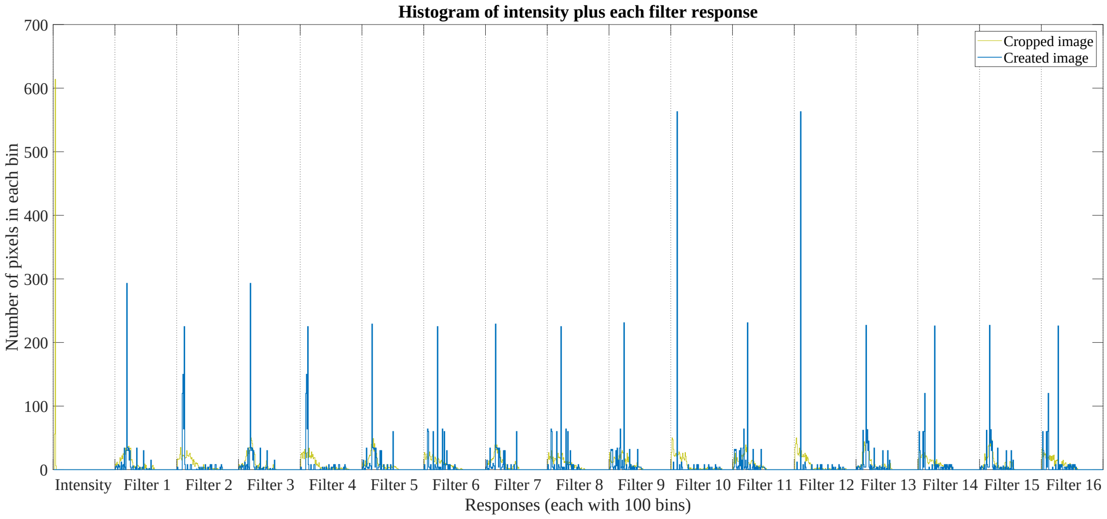


FIGURE 3.18: Histogram of intensity values and filter responses for each patch. Each response is split into 100 bins.

These issues were remedied by doing two things; filtering the images before cropping them and using the Matlab function `gabor` to create the Gabor filter bank and the function `imgaborfilt` to apply the filters. An example of the filter responses for an entire image is shown in Figure 3.19. This image has been filtered through a Gabor bank of filters of size 4, 6, 12, and 32 pixels and orientations 0°, 45°, 90°, and 135°. Figure 3.20 shows the responses for a dark patch cropped out of the filtered images. This is filter response is much closer to what was expected. By cropping *after* filtering and not using zero padding in the Gabor filter method, much clearer samples are produced with much less noise compared to Figure 3.17b.

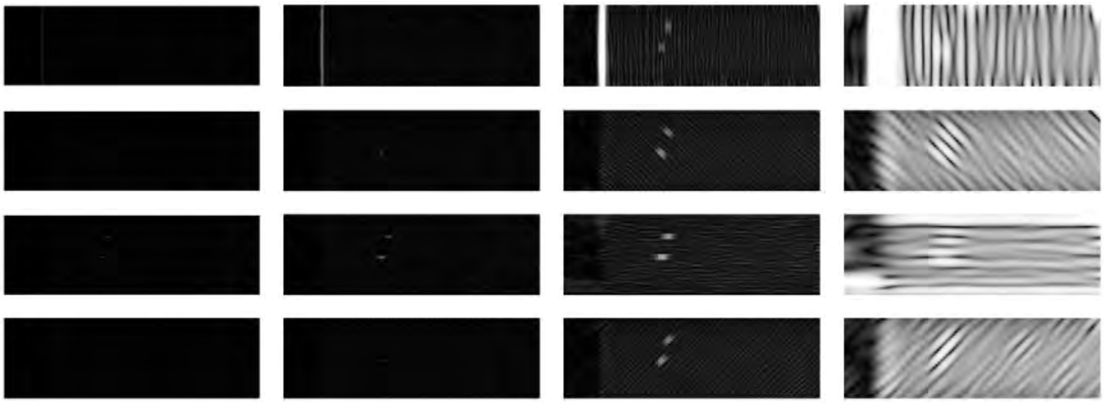


FIGURE 3.19: Gabor filter response for a whole training image.

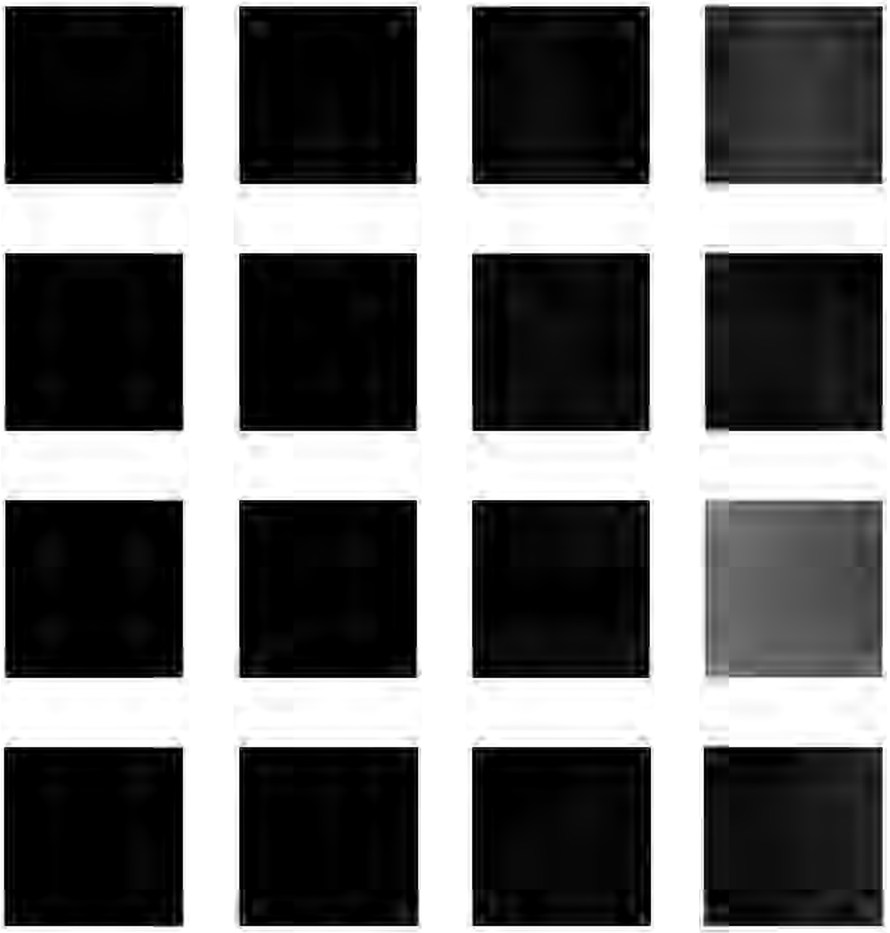


FIGURE 3.20: Gabor filter response for a dark patch cropped out of the filtered images.

It was then decided that, since edge images are treated differently and there is already a method in place to deal with them, they would not be included in this data set. However, implementing these fixes on edge images also improved the quality of the non-edge samples. The use of the Matlab method of Gabor filtering gets rid of zero padding, meaning that there are no unusual borders on samples. Also by filtering the whole image, and then cropping out the ROIs, in the testing phase whole images can be filtered and then cropped which requires less computational work.

### 3.3.3 Image correction

Ideally all of the images used would be flat, without the darkness at the top and bottom of the image. However, due to the set-up of the steel surface, cameras, and lighting there is a distinct lighting gradient on the images. To visualize the gradients of each image, the mean grey value for each of the 240 rows of pixels was taken for the entire test set of images and plotted in Figure 3.21. In this figure, each image is represented by a line and the thick blue line shows the mean of all of the images. There is a difference of 38 between the mean minimum grey value and the mean maximum grey value. The average grey value of each row would ideally form a roughly vertical line, depending on what defect is present, but this curve shows that the grey values are smaller at the top and bottom of the image, meaning that the image is darker on these edges.

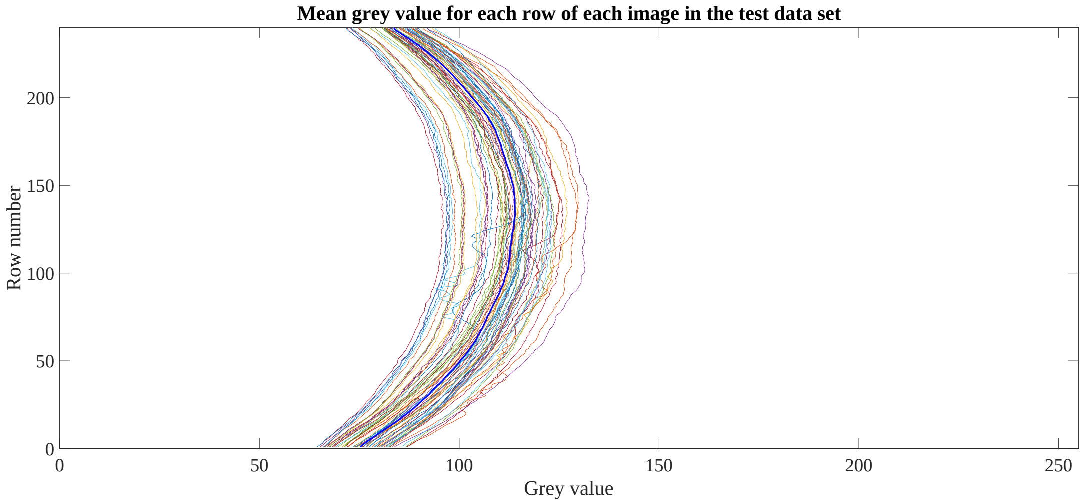


FIGURE 3.21: This plot shows the variation of grey values from top to bottom of each image from the test set where the thick blue line is the mean gradient.

By using the `polyfit` function, polynomials were fitted to the grey values of each image and then the images were corrected by taking off the difference between the maximum and minimum grey values. Figure 3.22 shows an example of the curves used in this process for an image of a heavy lamination. Each value on the $y$-axis represents each row of pixels which make up the image.

The $x$ values are the average grey value of each row of pixels. The blue curve is calculated by taking the median grey value of each row of the image; the red curve is the polynomial fitted to the blue curve, and the green curve is the correction used to produce the final flattened image.

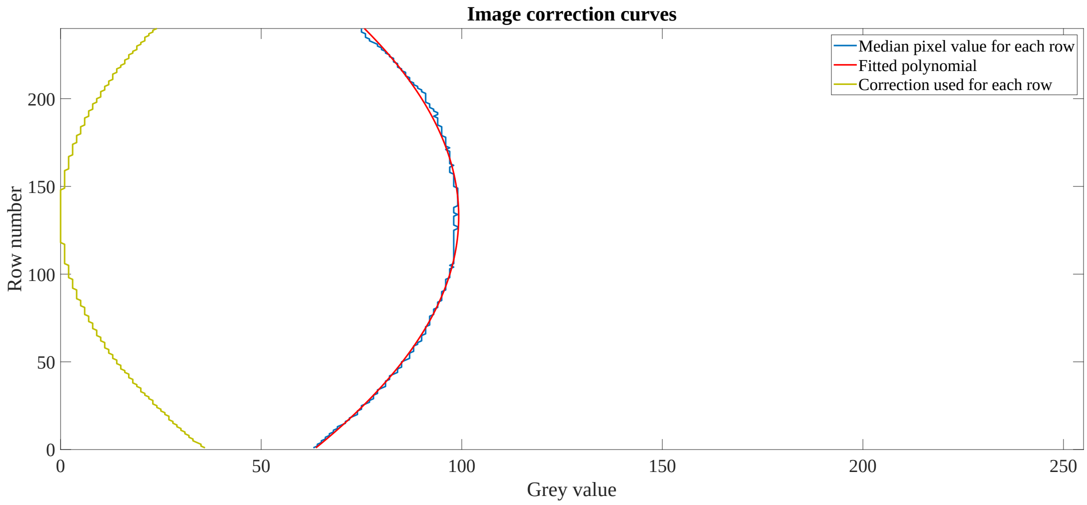


FIGURE 3.22: This plot shows an example of the curves used in image correction.

Figure 3.23 shows the before and after images for the case of a heavy lamination image. The top image shows the original and the lower image is the flattened image. Flattening the image removes any confusion of having darker grey values without a defect being present.

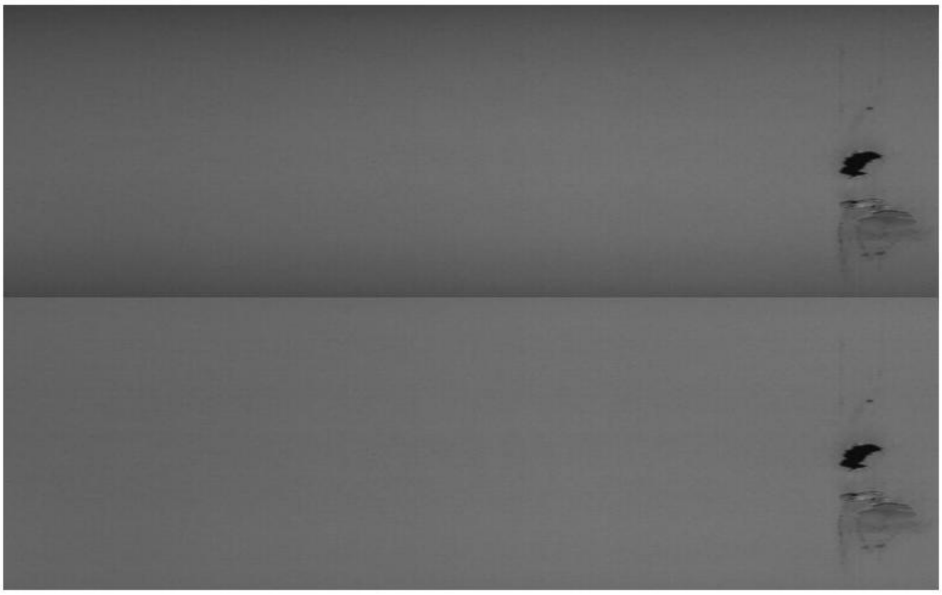


FIGURE 3.23: These images are the before and after images of a heavy lamination image that has been flattened.

When using the Gabor filter method which features zero-padding, after flattening, the images were mirrored around the edges to ensure that the filters will be applied accurately to the edge pixels. Without mirroring there can be some confusion added when Matlab uses zero-padding to make up the 26 × 26 window around the edge pixels. Once the Matlab function for filtering was used, mirroring the edges was no longer required.

### 3.3.4 Training Data

The training data used in this chapter consists of 2000 images of steel defects taken from data produced over a three week period and manually verified on screen by an inspector. Of the 94 defect types which occurred over this time period, many had very few appropriate example images and were excluded from this analysis. Images which contain the edge of the strip and a section of the rollers were excluded as these are treated separately or else they add confusion to the classifier which decreases performance. Large stains were also excluded as these are treated as a separate event. This left only 20 suitable defect types and, due to time constraints, 100 images of each type were hand-labelled with 26 × 26 pixel regions of interest (ROIs) and 26 × 26 pixel regions of defect-free steel. An example of a labelled training image can be seen in Figure 3.25. Defect types included are laminations, damage, dirt, dents, stains, scratches, oil spots, bruises, holes, and scale, all of various grades. Figure 3.24 shows some examples of defects and Figure 3.25 shows an example of a labelled training image.

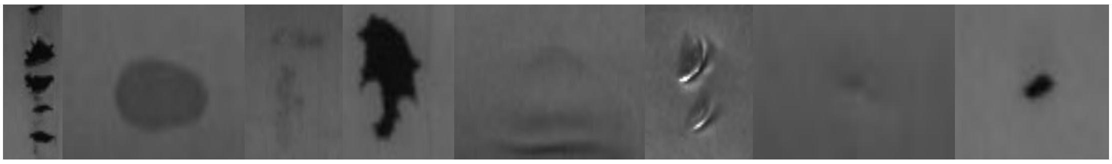


FIGURE 3.24: Examples of defect images; heavy lamination, oil spots, possible laminations, black dirt, bruises, damage, dents, and holes, respectively.

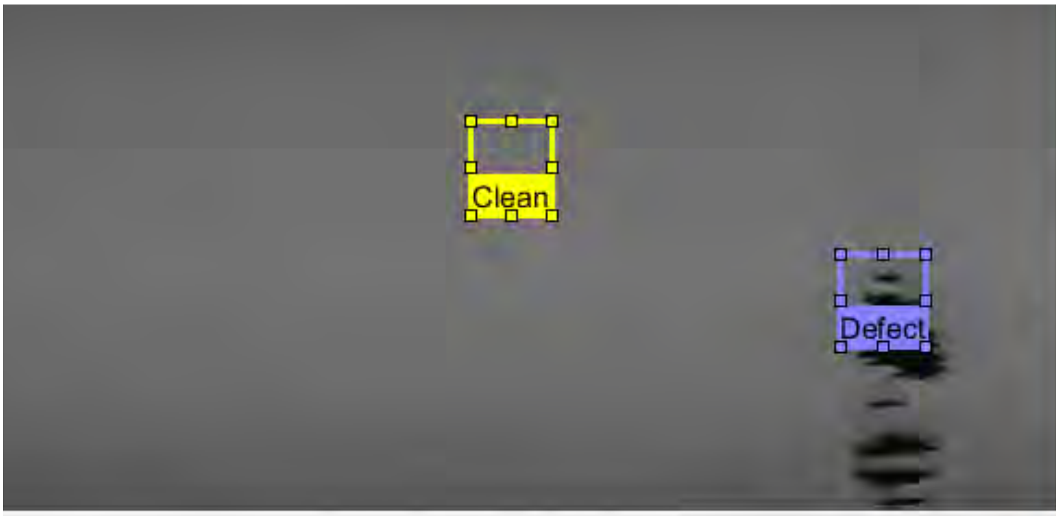


FIGURE 3.25: A labelled training image example.

### 3.3.5 Testing Data

The testing data is made up of 72 images of steel defects, featuring at least two of each of the 20 defect types. Originally five samples of each type were selected, but those which contained the strip edge were later excluded as edge images are subject to different treatment than images of the central strip. An example of a test image can be seen in Figure 3.26.

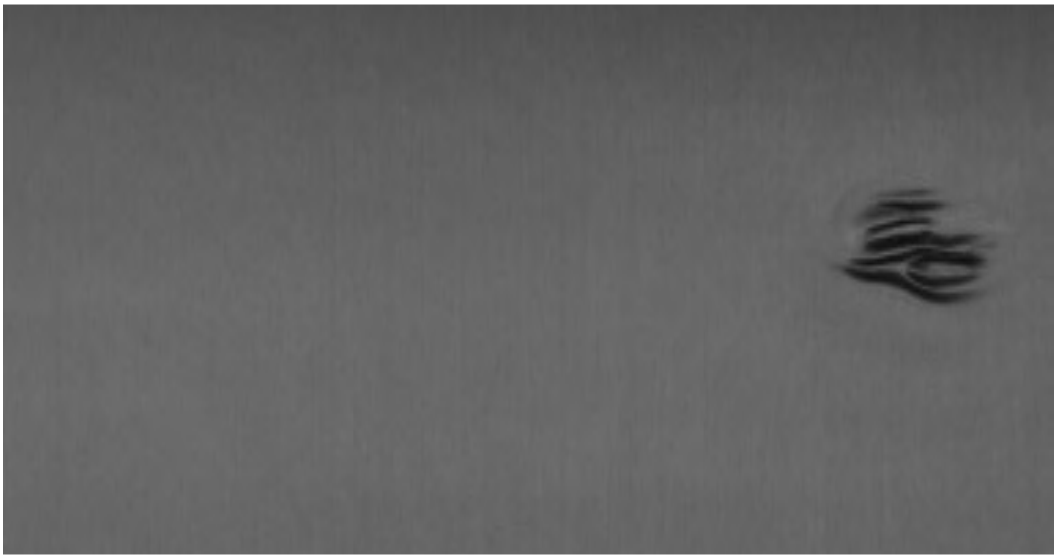


FIGURE 3.26: An example test image.

## 3.4 Using Gabor filters and random forest classifiers in steel defect detection

A basic outline of the proposed method can be seen in Figure 3.27. Generally this method can be considered as three key phases: generating the training data, training the classifier, and testing the classifier. These phases were written as three separate codes in MATLAB 2018a [3].

### 3.4.1 Phase 1: Generating the training data

In the first stage the original images were gathered and labelled using 26x26 pixel boxes for defects and non-defects. It was important to choose the clean steel boxes carefully because, due to the appearance of the steel's surface, some areas were not as uniform which, although not considered a gross defect, would cause confusion in the classifier; and example of two clean steel surfaces can be seen in Figure 3.28a.

Due to the set up of the production line, the images of steel are not a flat surface with equal lighting across the length of the image. The images are rounded, with a darker gradient at the top and bottom of the image. Because of this, the whole images were corrected, to flatten their edges, making the grey background a homogeneous surface. This was achieved by first calculating the median grey value of each row and fitting a polynomial to these grey values. Then the row values are subtracted from the maximum of the fitted polynomial and rounded to the nearest integer, so that the new row medians would form a vertical line at the maximum. These corrected values are then added onto each element in the corresponding row, producing a flat image. Using the maximum of the polynomial, and not a mean, ensures that all grey values remain in the $[0, 255]$ space because correction does not require subtracting from grey values. This method of image correction is explained fully in section 3.3.3.

Gabor filters are a common filter type used in image analysis problems. It has been shown that they can capture the same fundamental visual properties as the human visual system [102]. Because of this, they have a wide variety of suitable applications such as medical image analysis, face detection and object detection [103, 76, 106, 69, 70].

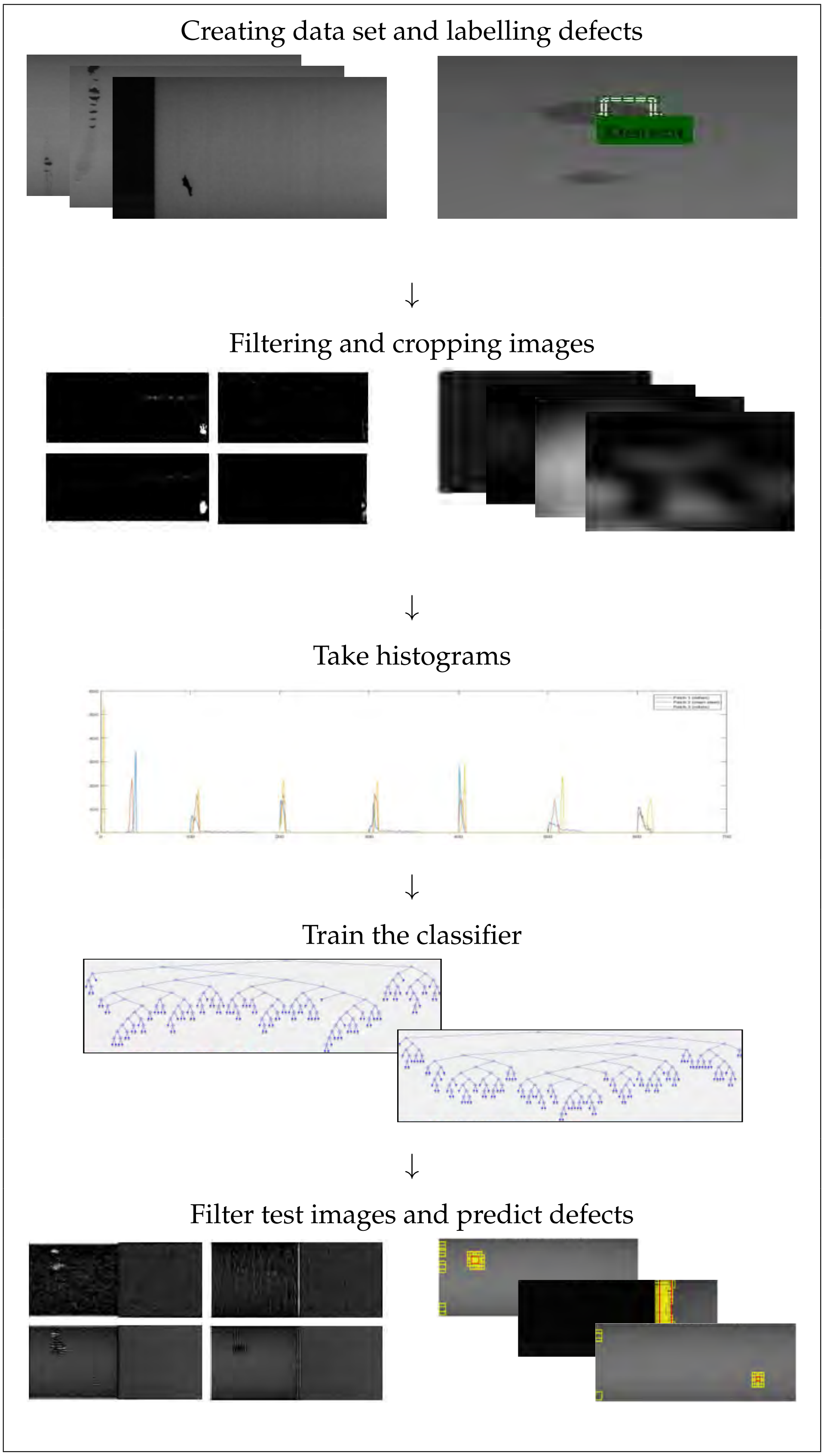
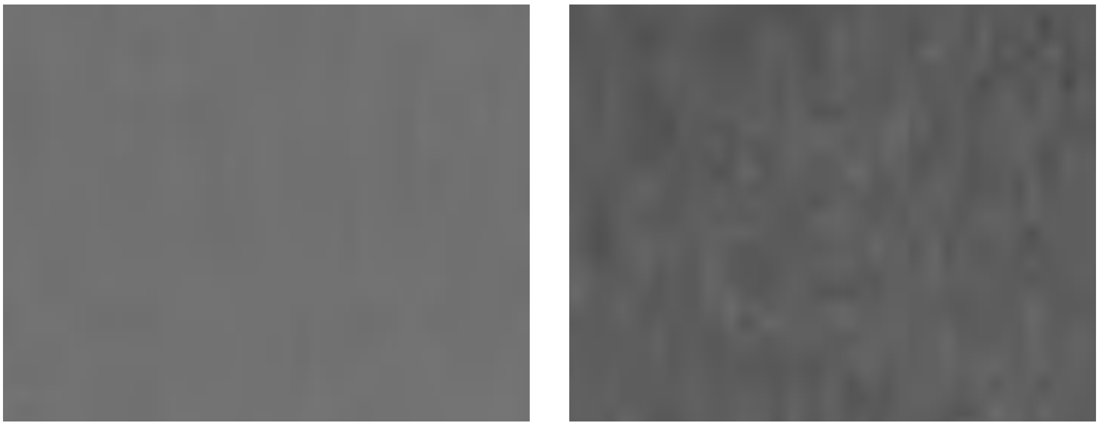

Creating data set and labelling defects
↓
Filtering and cropping images
↓
Take histograms
↓
Train the classifier
↓
Filter test images and predict defects

FIGURE 3.27: Method outline.



(A) A good clean surface.

(B) A bad clean surface.

FIGURE 3.28: Examples of good and bad clean steel surface.

A bank of Gabor filters was created at four different wavelengths and orientations, using the matlab function `gabor`. It was observed that the filter results for orientations 0° to 180° and 180° to 360° were very similar, so the orientations $[0°, 45°, 90°, 135°]$ were selected to minimise computational cost. The wavelength can be in the range $[2, inf)$ [117] so the size of the defects was considered and the wavelengths $[4, 6, 12, 32]$ were selected based on a trial process of which wavelengths visibly showed the most defect information for a set selection of images. Using a filter bank too large can become computationally expensive, but using too few orientations and scales can mean certain types of defects are no longer visible.

Using this Gabor filter bank, the whole images were filtered and then an energy transform was applied, as described by equation 3.4 where $F_{Gabor}$ is the filter responses and $t = 0.25$ is a constant, as it is in [98] and [118].

$$\psi(F_{Gabor}) = \tanh(tF_{Gabor}) = \frac{1 - \exp^{-2tF_{Gabor}}}{1 + \exp^{-2tF_{Gabor}}} \tag{3.4}$$

The regions of interest (a.k.a. positive windows) and defect-free regions (a.k.a. negative windows) were then cropped out of the whole images into a matrix of size [window size, window size, number of Gabor filters, number of training images]. A series of histograms were taken, one for the unfiltered image, and then for each of the 16 filter responses. These histograms have 100 bins, with limits $[0, 1]$ and results in a $1 \times 1700$ feature vector for each $26 \times 26$ region. No normalisation was required for these histograms since all outputs are in the range $[0, 1]$.

The last step before training was to concatenate the positive and negative feature vectors into a matrix and to generate the vector of known classes for this feature vector, where the class 1 means a defect is present and the class 0 means there is no defect.

### 3.4.2 Phase 2: Training the binary classifier

The proposed method uses a random forest classifier to decide the location of defects. Random forest classifiers are made up of numerous trained decision trees. Using the matlab function `TreeBagger` a specified number of trees are generated and trained by sampling a random subset of data from the training set, with replacement. Using a random subset of data to train each tree helps to avoid overfitting. Assessing all of the features and associated class given to it, the decision tree will grow its branches through a process of minimising variance in sub-branches. When predicting new outputs, every tree is given the unseen image and generates a prediction output, and the most voted output class then becomes the final prediction decision [114]. On their own, decision trees have poor prediction rates, but taking the most voted decision from a large number of trees gives greatly improved performance for only a small expense in computational cost.

The inputs used to train the `TreeBagger` [111] function were the number of trees, the $4000 \times 1700$ feature vector derived from the training set, and the $4000 \times 1$ vector of known classes for the features. The method 'classification' was specified and *OOBPredictorImportance* and *OOBPrediction* were selected as 'on'. The *OOBPredictorImportance* and *OOBPrediction* allow the out-of-bag error to be calculated. Given that each decision tree is trained on a random subset of data, for any given tree there is a subset of data which it has not seen yet. This data is referred to as 'out-of-bag' and the fact that this data is unseen means that it can be used to calculate the error of each tree. This is how the out-of-bag error (*OOBerror*) has been calculated and this error has been found to be roughly as accurate as using a new data set of the same size as the training set to test the algorithm accuracy [119]. This enables the process of optimization to be carried out without having to have a large validation set of data since quantity of data is a limitation in this research.

The proposed method uses 200 trees in the random forest classifier, as the *OOBerror*, seen in Figure 3.29, shows that minimum error occurs at tree 131 and so this was rounded up to 200 to allow for any variation that may occur. Adding many more hundreds of trees just adds to computational cost without considerable gain in accuracy.

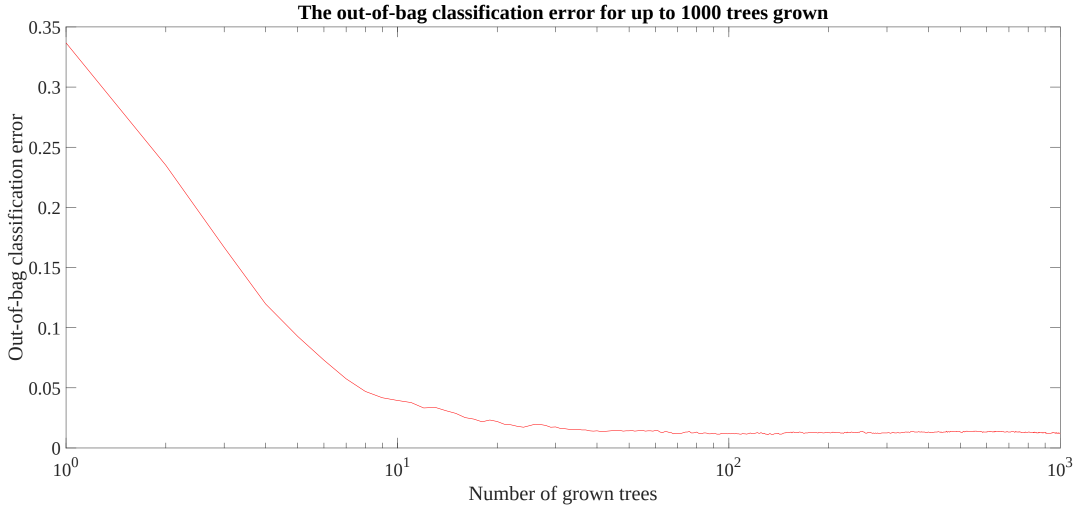


FIGURE 3.29: The out-of-bag error for a random forest binary classifier with 1000 trees grown.

### 3.4.3 Phase 3: Testing the classifier

The test images need to enter the classifier as an $n \times 1700$ vector where $n$ is the number of test images. This process begins with filtering the whole test images in the same manner as the training data. From here a sliding windows function records a $26 \times 26$ window every 10 pixels across and down the whole image. The output of this function is *Windows* which is a matrix of [window contents × window contents × number of windows across the whole image × number of filter responses] and *Locations* which is a matrix of [$x0$ $y0$ width height] corresponding to each window in *Windows*. Each of these windows is then passed through the random forest classifier for prediction. This produces a vector of predicted classes for each window where 0 means there is no defect and 1 means there is a defect. These predictions are then compared to the labelled regions of interest and the regions of interest generated by the currently used commercial system, we will call *system X*, which detected the presence of these defects as the material went through the production line.

The known classes of each window were calculated by assigning a positive class to any window which contains a labelled region in at least 50% of its area. A flow chart to explain this process is in Figure 3.30. This percentage was selected to ensure that all areas labelled as defects definitely contained defects, and were not just the empty corner of an ROI. The first confusion matrix compares these known window classes with the predicted windows.

Next, any positively predicted windows which overlap another window by at least 50% of its area were merged and these merged regions were compared with the known windows. The same flow chart process in Figure 3.30 is applied to the merged predicted regions to determine the prediction of each Window location according to these new merged regions. Comparing these with the known windows produces the second confusion matrix.

The *system X* predictions for each window were also extrapolated from the predicted region of interest in the same fashion; any window whose area was at least 50% filled by the *system X* predicted region of interest was considered to be a positive prediction. The third confusion matrix compares these *system X* predictions with the known window classes.

This first confusion matrix gives the prediction accuracy of the unprocessed results and is seemingly more reflective of the real accuracy of the method. However this second confusion matrix allows fair comparison of the accuracy of the *system X* predictions and the proposed method predictions against the known, labelled regions of interest.

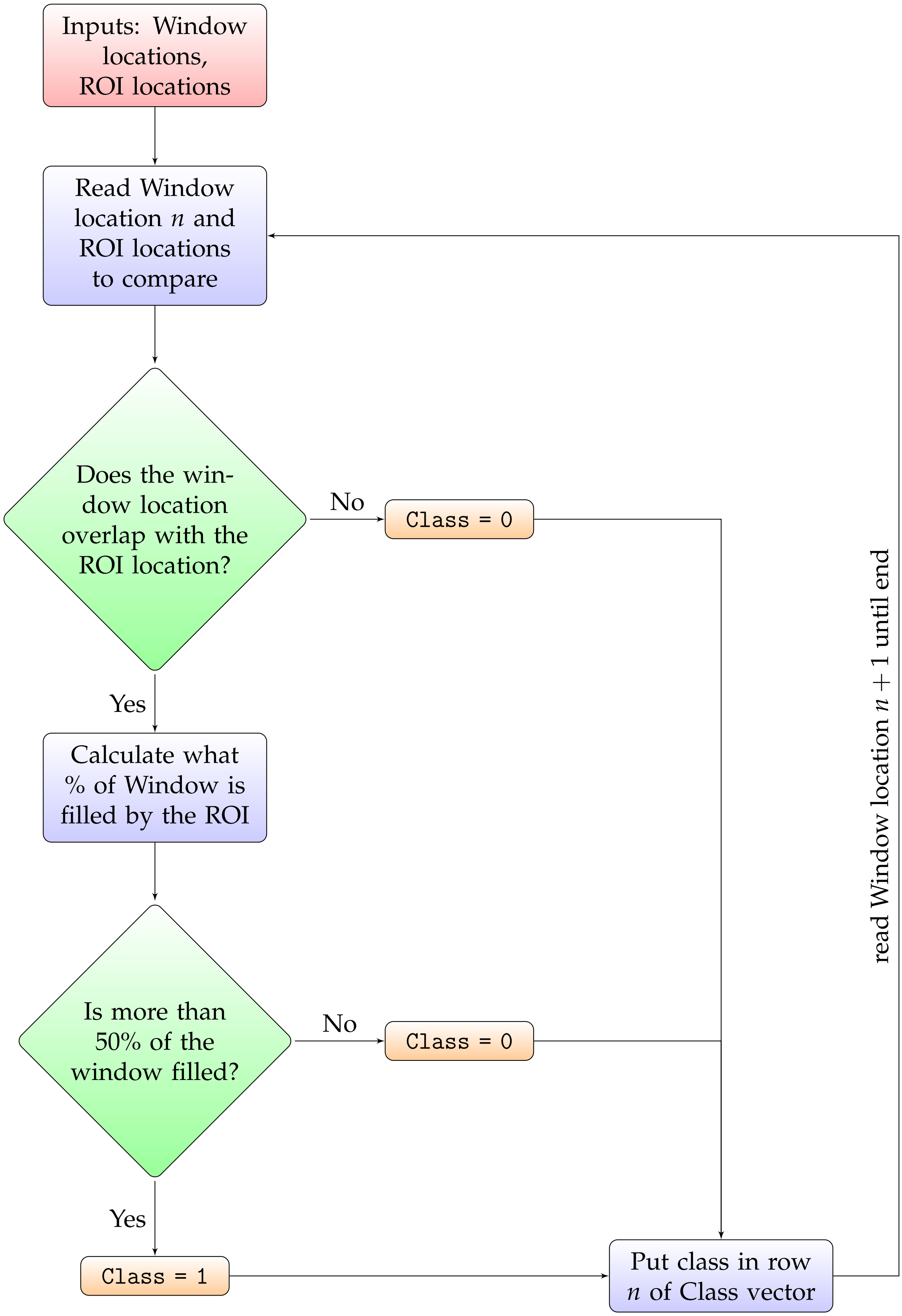

Inputs: Window locations, ROI locations
Read Window location $n$ and ROI locations to compare
Does the window location overlap with the ROI location?
No
Class = 0
Yes
Calculate what % of Window is filled by the ROI
Is more than 50% of the window filled?
No
Class = 0
Yes
Class = 1
Put class in row $n$ of Class vector
read Window location $n+1$ until end

FIGURE 3.30: Flow chart showing the process to determine which Windows featured the labelled ROI.

The measures of comparison are the overall accuracy, the false positive rate, and the false negative rate. The calculations for these percentages are described in equations 3.5, 3.6, and 3.7; where true positives (TP) and true negatives (TN) are the correctly predicted positives and negatives respectively, false positives (FP) are known negatives falsely predicted as positives, false negatives (FN) are known positives falsely predicted as negatives, TNR is the total negative rate, TPR is the total positive rate, and TNS is the total number of samples.

$$Overall\ Accuracy\ \% = \frac{(TP + TN)}{TNS} \times 100 \tag{3.5}$$

$$\begin{aligned} False\ Positive\ Rate\ (FPR)\ \% =& \frac{FP}{(TN + FP)} \times 100 \\ =& 1 - TNR \\ =& 1 - \frac{TN}{TN + FP} \end{aligned} \tag{3.6}$$

$$\begin{aligned} False\ Negative\ Rate\ (FNR)\ \% =& \frac{FN}{(TP + FN)} \times 100 \\ =& 1 - TPR \\ =& 1 - \frac{TP}{TP + FN} \end{aligned} \tag{3.7}$$

The aims for these measures are that the *overall accuracy* should be as high as possible. The *false positive rate* should be minimised where possible to reduce unnecessary time spent by inspectors checking these, however not at the cost of the false negative rate. The *false negative rate* needs to be as low as possible and should be the measure considered most important because there can be large costs associated with undetected defects, such as broken machinery for both the steel producers and their customers, which can have dire consequences. One could calculate the F1 measure to compare methods and this could be done in further work. However the F1 measure gives equal weighting to both the false negative rate and the false positive rate so is not as applicable here given the higher importance of the false negative rate due to the consequences of unknown defects.

### 3.4.4 Results

Tables 3.4, 3.5, and 3.6 show the overall accuracy, false positive rate and false negative rate of the *system* X predictions and the two methods for quantifying accuracy previously discussed in section 3.2.4. These tables consist of the red and green cells, and the six calculated percentages underneath. The upper left and lower right green cells are the number of correctly predicted negative, and positive, samples and what percentage of all predictions they constitute, respectively. It is expected that there are far more negative samples than positive samples. The lower left and upper right red cells are the number of incorrectly predicted positive, and negative, samples and what percentage of all predictions they constitute, respectively. The text in blue is the false positive rate, the text in orange is the false negative rate, and the text in green is the overall accuracy. For example, in Table 3.4, 112128 samples were correctly predicted to be negative samples, which is 94.4% of all samples tested; 115 defect samples were incorrectly predicted to have no defects present, which is 0.1% of all samples tested; 5226 clean samples were incorrectly predicted to have defects present, which is 4.4% of all samples tested; 1331 defect samples were correctly predicted to be positive samples, which is 1.1% of all samples tested; the false positive rate is 4.5%, the false negative rate is 8.0%, and the overall accuracy is 95.5%.

The *system X* predictions for this data set have a high overall accuracy of 98%, with a low false positive rate of 1.8% and a high false negative rate of 15%. The reason for this high false negative rate is that *system X* will select the worst features it sees in an image and not every single detection that is found.

TABLE 3.4: Results for proposed method *before processing*.

| | | | | |
|---|---|---|---|---|
| Predicted Class | 0 | 112128<br>94.4% | 115<br>0.1% | |
| | 1 | 5226<br>4.4% | 1331<br>1.1% | |
| | | 95.5%<br>4.5% | 92.0%<br>8.0% | 95.5%<br>4.5% |
| | | 0 | 1 | |
| | | Known Class | | |

TABLE 3.5: Results for proposed method *after processing*.

| | | | | |
|---|---|---|---|---|
| Predicted Class | 0 | 103429<br>87.1% | 17<br>0.0% | |
| | 1 | 13925<br>11.7% | 1429<br>1.2% | |
| | | 88.1%<br>11.9% | 98.8%<br>1.2% | 88.3%<br>11.7% |
| | | 0 | 1 | |
| | | Known Class | | |

TABLE 3.6: Results for *system X*.

| | | | | |
|---|---|---|---|---|
| Predicted Class | 0 | 115246<br>97.0% | 217<br>0.2% | |
| | 1 | 2108<br>1.8% | 1229<br>1.0% | |
| | | 98.2%<br>1.8% | 85.0%<br>15.0% | 98.0%<br>2.0% |
| | | 0 | 1 | |
| | | Known Class | | |

The unprocessed predictions for the proposed method show a good all-round performance, with an overall accuracy of 95.5%, a fairly low false positive rate of 4.5%, and a slightly too high false negative rate of 8.0%. The processed predictions give a much lower accuracy of 88.3%, a high false positive rate of 11.9%, and a low false negative rate of 1.2%.

Taking the image results with the unprocessed predictions and known regions of interest drawn onto the image, the areas where false negatives occur are generally inside very large dark spots where the entire window is one dark, uniform colour. The reason the false negative rate drops between unprocessed and processed data is because by merging all of the windows around the edge of the spot, the centre of the spot is then included in the region of interest. This is the same reason we see an increase in the false positive rate. Whilst the false positive rate is fairly low using the unprocessed data, once windows have been merged into a rectangle, there will be windows encompassed within that rectangle which do not feature a defect, but have to make up part of the rectangle because the defect is oriented diagonally. More windows become positive predictions when using the 50+% criteria than when using the unprocessed data.

Comparing *system X* and the processed predictions, *system X* performs better overall, with a 10% higher accuracy than this method. However the false negative rate for the proposed method is significantly better than *system X*. Since *system X* could be missing up to 15% of defects and due to the importance of the false negative rate in industry, the proposed method could be an appropriate alternative to the current system.

The unprocessed results present a good middle ground between the processed results and the *system X* results. The false negative rate is 7% less than that of *system X* and The overall accuracy is 95.5% which is far more than the 88.3% achieved in the processed results, and only just behind the *system X* results of 98%.

The images in Figure 3.31 shows a selection of test responses. Some of the responses have large amounts of false positive regions which in no way overlap the labelled regions of interest, such as the second image in Figure 3.31. However, most responses only have one or two small false positive regions and these are commonly around the edge of the image.

There are two distinct limitations with this experiment. The first is the lack of labelled and verified data. The data provided has to be manually labelled, which in itself is not always accurate. For some defect types the only way to confirm what they are and their true boundary is by metallurgical testing and the work required to create a data set of 100,000 or even 1 million samples would be significant. Whilst it is not completely impossible to create a data set of this magnitude, it would require considerable investment of resources. However, a larger data set to train the classifier would produce a more robust classifier, but the level of increase in data required to see improvement in results is currently unknown.

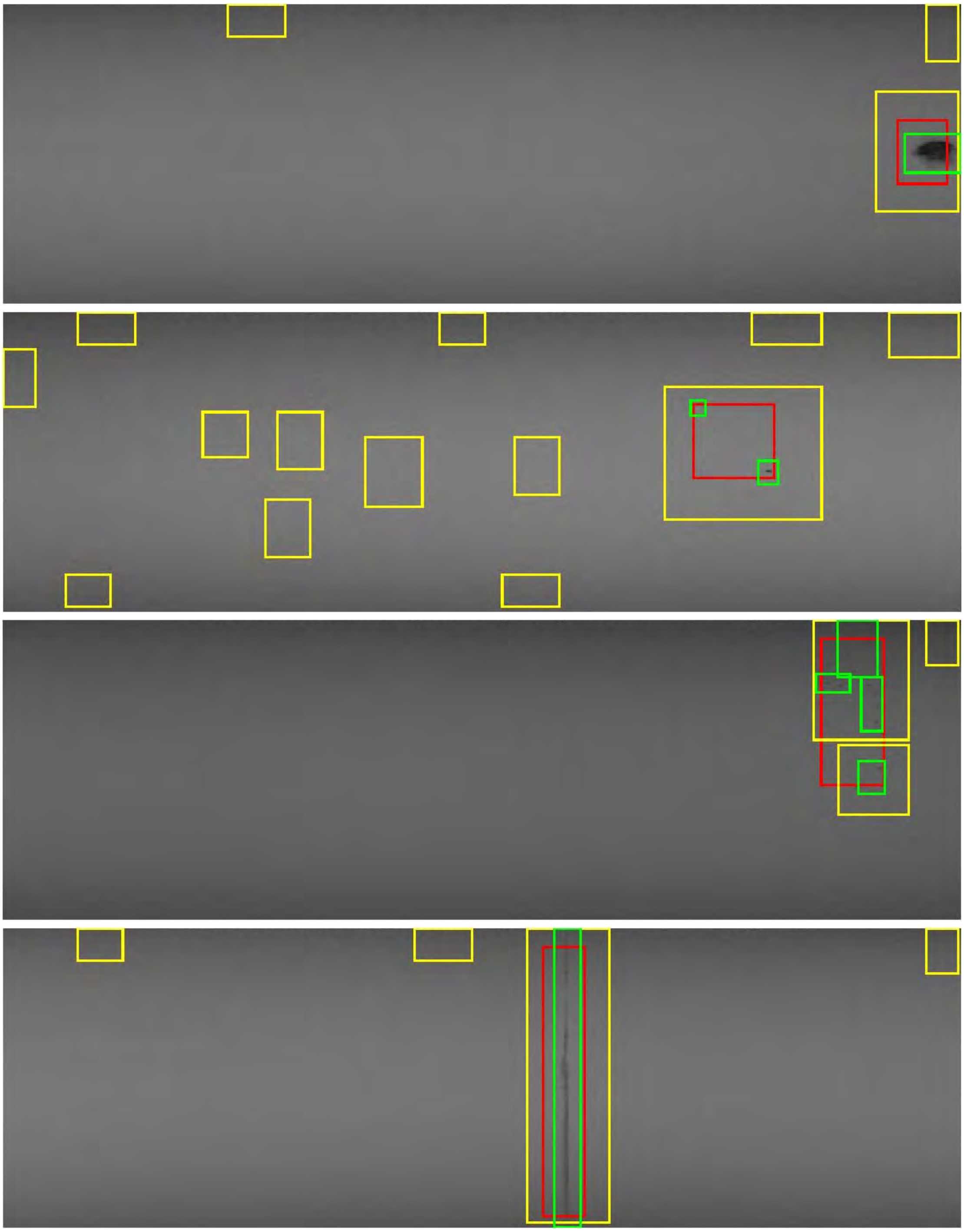


FIGURE 3.31: An example of the processed predictions (yellow), the *system X* prediction (red), and the known, labelled ROIs (green).

The second limitation comes in the ability to confidently quantify accuracy. Taking the proposed method's direct predictions shows high accuracy of 95.5%, with a lower false positive rate of 4.5% and a slightly higher false negative rate of 8%. Merging the unprocessed predictions and using the criteria of 50+% of each window filled by the merged ROI to generate a positive result gives a lower overall accuracy of 88.3%, a high false positive rate of 11.9%, and the lowest false negative rate of all three methods at 1.2%. This discrepancy of boundaries can be clearly seen in Figure 3.32, where the green boxes are the hand-labelled ROIs, the red box is the *system X* prediction for this image and the yellow boxes are the processed results for the proposed method. Given that the proposed method 2 draws a wider boundary around the ROI than the labelled ROIs, a higher false positive rate is expected however this does skew the overall accuracy.

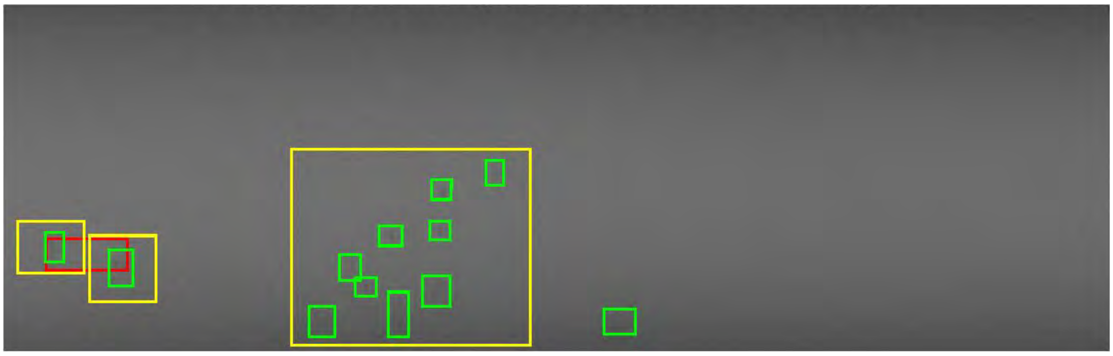


FIGURE 3.32: An example of the processed predictions (yellow), the *system X* prediction (red), and the known, labelled ROIs (green).

To measure this increase in false positive area, the areas of any predicted positive regions which overlapped a labelled ROI were calculated and a percentage increase of the labelled ROI was calculated. On average, the area of positive predictions was 197% larger than the area of the labelled ROI area it overlapped, meaning the predicted positive regions are approximately three times the size of the associated labelled ROIs. One possible cause of this increase in region size could be down to the filter selection. Figure 3.33 shows the 0° filter responses for a long thin defect. In the fourth filter, the one with the largest scale, the response to the defect fills the whole red box. So as this filter response is used in training and testing, it makes sense that the detected boxes may overestimate the size of the defect. If this scale size were excluded, or a smaller scale selected, the increase in region size may be smaller. Further work could be done on which scales and orientations of filters may lead to improvements. A balance needs to be struck to make sure both small and large defects can be detected.

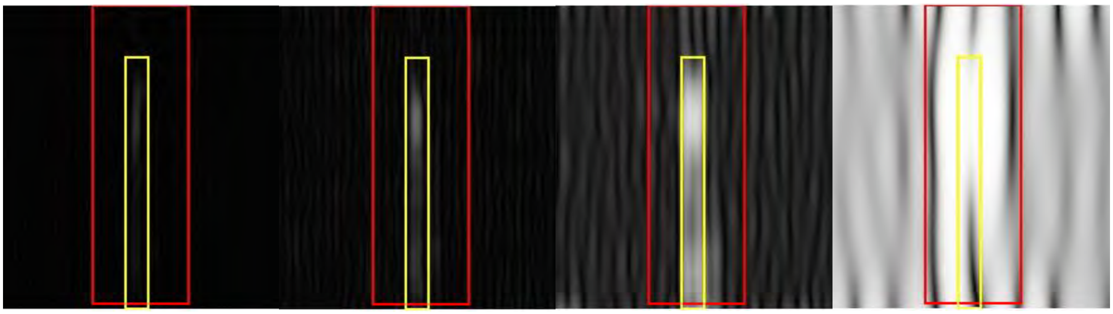


FIGURE 3.33: An example of the 0° orientation filters at all four scales. The processed predictions (red), and the known, labelled ROIs (yellow) are laid over the image to identify where the edges of the filter response are.

### 3.4.5 Conclusions

This section presents a defect detection method applied to strip steel defect images. The proposed method could be an alternative approach to *system X* currently used within the company, given a larger data set and further defect analysis. However, further functionality would need to be added, such as a user interface and a method to export the data to the current systems.

Using four wavelengths and orientations of Gabor filters gives adequate feature information for the classifier to distinguish between defects and non-defects. Since there is no proven best suggestion of how to decide which filter to use, further exploration could be done into the optimal wavelength and orientation of filters for steel images or even specific types or sizes of steel defects.

The distribution of shape and type of defects is important to consider to avoid bias. Excluding edge images from the data ensures there isn't unnecessary confusion between what is and isn't a defect and excluding large stains ensures the program isn't flooded by defects which would be treated as individual events within production.

Whilst the existing settings of the Treebagger function provide a fairly accurate detection method, if there had been a larger data set labelled and verified then a validation set could have been made, to further optimise the function. A larger data set would also be beneficial to train a more robust classifier. Further analysis of a larger number of defect shapes and sizes could lead to a multi-classifier algorithm resulting in higher computational cost, but improved results.

This piece of work highlights the importance of accurately quantifying the accuracy in this case, as different measures can lead to quite different appearances in results. Where calculating accuracy is automated, it is important that the actual images are checked for clarity as the numbers do not always show the full picture.

The method proposed in this section gives a notable increase in the false negative rate for detecting the presence of steel defects with only 2000 training images. This improvement means that more surface defects are detected which could lead to an increase in customer satisfaction and a decrease in production issues further down the line.

## 3.5 Case Study: Use of histograms and Gabor filters in detecting steel defects - are the extra steps necessary?

### 3.5.1 Introduction

This case study considers which combination of Gabor filters, histograms and random forest classifiers produces the best results for steel defect detection application. Any detection method used on the production line needs to run in real-time or else it will delay the production process unnecessarily. Therefore any decrease in computational cost, which is not at the expense of any form of accuracy, would be beneficial.

### 3.5.2 Method

The code for the method in section 3.4 was modified to test two further variations of feature vector, each with the same corresponding class vector. The class vector has the known class of each sample on its respective row. In this case the class vector is $4000 \times 1$ and the first 2000 rows feature defects (class 1), the last 2000 rows feature clean steel (class 0). The three variations are listed below.

1. Variation 1: Corrected $26 \times 26$ samples
2. Variation 2: Corrected $26 \times 26$ samples put into a histogram with 100 bins and limits $[0, 1]$
3. Variation 3: The final method outlined in section 3.4; corrected $26 \times 26$ samples put through a Gabor filter bank, energy transform, and then into a histogram with 100 bins and limits $[0, 1]$

Each variation of the feature vector follows a similar method of generating features from the same labelled images, training the classifier, and then testing the classifier on the same set of unseen samples. This testing phase produces two types of measures for comparison; the first is to produce images with the detected regions overlaid to visualise the predicted and known results for each test image. The second measure involves creating two confusion matrices.

The testing phase takes each test image and uses a sliding window function to divide the image into $26 \times 26$ windows at 10 pixel increments. Each window is tested and a prediction of whether or not a defect is detected is recorded. Then the known region of interest (ROI) for each test image is compared to the window locations to determine which window locations feature the known ROI. The positively predicted windows are then merged where possible and these merged windows are then compared against the known ROIs in the same fashion as described in section 3.4.3. This produces two confusion matrices, one before and one after this merging process.

#### Variation 1

The method for variation 1 is simple; the labelled defect and clean samples are cropped out of their original images, reshaped into a $1 \times 676$ vector and concatenated together into a $4000 \times 676$ feature vector. This feature vector and its associated vector of classes are fed into the random forest classifier with 200 trees grown and the rest of the settings left as default values. The test images were then split up into windows using a sliding windows function with a 10 pixel increment. These windows were then passed through the classifier and a confusion matrix was formed.

#### Variation 2

The method for variation 2 involves cropping the samples from the original images, taking histograms of each sample with limits $[0, 1]$ and 100 bins. The limits are pre-defined as the images can only possibly have values between 0 and 1, so by pre-determining the bin limits and number of bins, it ensures that all histograms are directly comparable. This aids the classifier in using this data to distinguish the differences between defects and non-defect samples. Once histograms have been taken, this produces a $4000 \times 100$ feature vector to train the classifier on. This should be quicker to train, given the difference in data size. The same test data set is then run through the classifier to get predictions and calculate a confusion matrix.

### 3.5.3 Results

The results for comparing the unprocessed and processed window predictions against the known window labels for variations 1, 2, and 3 are shown in Tables 3.7, 3.8, and 3.9, respectively. Unprocessed variation 1 initially appears to be very successful, with a 99% overall accuracy rate with 117671 out of 118800 windows correctly predicted and only a 0.6% false positive rate. This means only 732 out of 117354 windows featuring no defects were misclassified as having a defect present. However the false negative rate is 27.5%, meaning 397 out of 1446 defect windows were missed which is too many. Missing more than a quarter of defects would be unacceptable in steel production. The processed results for variation 1 are a marked improvement over the unprocessed results. A significant decrease in the false negative rate was achieved with only a small decrease in the overall accuracy and small increase in false positive rate, making it a more viable method. Unprocessed variation 2 has a slightly lower overall accuracy at 98.3% with 2023 out of 118800 windows detected incorrectly. The false positive rate of 1.6% is nearly 3 times that of unprocessed variation 1, but less than the 2.8% false positive rate seen in the processed variation 1 results. The false negative rate is greatly improved at 9.6%, but this is still quite high as 139 out of 1446 defect windows were not detected as defects. Processed variation 2 follows the trend and has a slightly decreased overall accuracy of 95.2%, a slightly increased false positive rate of 4.8% and a significantly smaller false negative rate of 2.1%. The results for variation 3 are featured in the previous section. The unprocessed results for variation 3 have a slightly better overall accuracy than processed variation 2 at 95.5% compared to their 95.2%; the false positive rate is slightly lower than that of processed variation 2, but still higher than the other variations at 4.5%; the false negative rate is almost four times that of processed variation 2. Processed variation 3 has both the lowest overall accuracy at 88.3% with 13942 incorrectly classified windows and the highest false positive rate at 11.9%; however the false negative rate is the best at only 1.2% with only 17 out of 1446 defect windows misclassified.

TABLE 3.7: Results for variation 1 before (left) and after (right) processing.

| | | | | |
|---|---|---|---|---|
| Predicted Class | 0 | 116622<br>98.2% | 397<br>0.3% | |
| | 1 | 732<br>0.6% | 1049<br>0.9% | |
| | | 99.4%<br>0.6% | 72.5%<br>27.5% | 99.0%<br>1.0% |
| | | 0 | 1 | |
| | | Known Class | | |

| | | | | |
|---|---|---|---|---|
| Predicted Class | 0 | 114059<br>96% | 117<br>0.1% | |
| | 1 | 3295<br>2.8% | 1329<br>1.1% | |
| | | 97.2%<br>2.8% | 91.9%<br>8.1% | 97.1%<br>2.9% |
| | | 0 | 1 | |
| | | Known Class | | |

TABLE 3.8: Results for variation 2 before (left) and after (right) processing.

|  |  | 0 | 1 |  |
|---|---|---|---|---|
| Predicted Class | 0 | 115470<br>97.2% | 139<br>0.1% |  |
|  | 1 | 1884<br>1.6% | 1307<br>1.1% |  |
|  |  | 98.4%<br>1.6% | 90.4%<br>9.6% | 98.3%<br>1.7% |

Known Class

|  |  | 0 | 1 |  |
|---|---|---|---|---|
| Predicted Class | 0 | 111708<br>94% | 31<br>0% |  |
|  | 1 | 5646<br>4.8% | 1415<br>1.2% |  |
|  |  | 95.2%<br>4.8% | 97.9%<br>2.1% | 95.2%<br>4.8% |

Known Class

TABLE 3.9: Results for variation 3 before (left) and after (right) processing.

|  |  | 0 | 1 |  |
|---|---|---|---|---|
| Predicted Class | 0 | 112128<br>94.4% | 115<br>0.1% |  |
|  | 1 | 5226<br>4.4% | 1331<br>1.1% |  |
|  |  | 95.5%<br>4.5% | 92.0%<br>8.0% | 95.5%<br>4.5% |

Known Class

|  |  | 0 | 1 |  |
|---|---|---|---|---|
| Predicted Class | 0 | 103429<br>87.1% | 17<br>0% |  |
|  | 1 | 13925<br>11.7% | 1429<br>1.2% |  |
|  |  | 88.1%<br>11.9% | 98.8%<br>1.2% | 88.3%<br>11.7% |

Known Class

Figures 3.34, 3.35, and 3.36 show an example test image for each variation. The test images are representative of these findings. Figure 3.36 misses many of the labelled defects, where as Figure 3.35 detects the majority, but with minimal false positive areas and Figure 3.34 detects all of the defects in the test image, but has a lot more positively detected areas which do not actually feature a defect.

The time taken to train the classifier was measured and the results shown in Table 3.10; these results also show the size of the feature vector used in training. Variation 1 was fairly fast, taking 5 minutes 32 seconds to train the classifier. Variation 2 was the fastest at 3 minutes 58 seconds, it is also the smallest feature vector used. Variation 3 took 6 minutes and 9 seconds and was the largest feature vector. These speeds were measured on a desktop computer, specifications listed in Appendix A, and not implemented on the servers in plant. So whilst

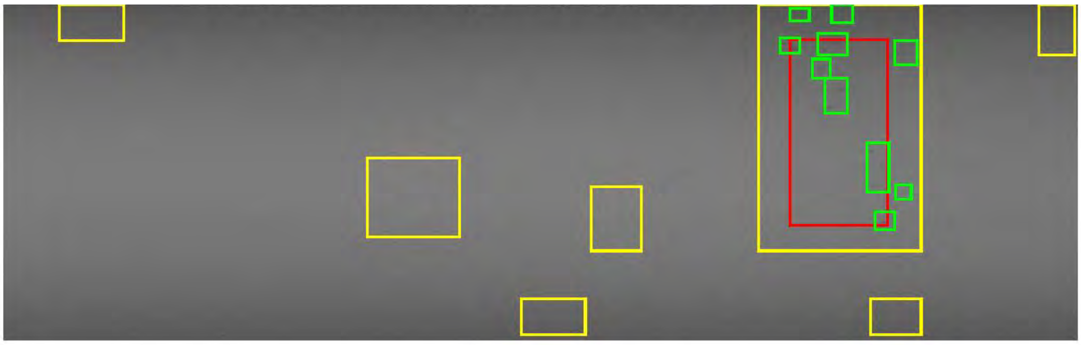


FIGURE 3.34: Variation 3 results for image 26 where the boxes are the processed predictions (yellow), the *system X* prediction (red), and the known, labelled ROIs (green).

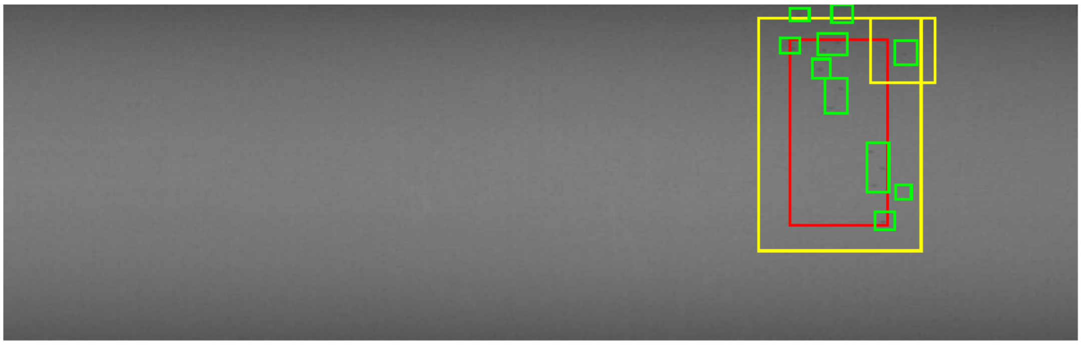


FIGURE 3.35: Variation 2 results for image 26 where the boxes are the processed predictions (yellow), the *system X* prediction (red), and the known, labelled ROIs (green).

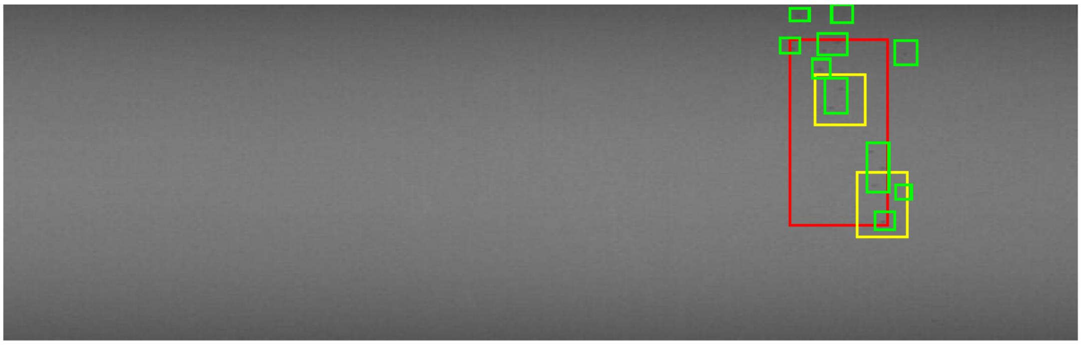


FIGURE 3.36: Variation 1 results for image 26 where the boxes are the processed predictions (yellow), the *system X* prediction (red), and the known, labelled ROIs (green).

the actual numbers are not relevant, it is worth noting that Variation 2 is significantly faster than the other two methods. When it comes to implementation, the accuracy must be balanced with the speed, as a method which is accurate, but cannot keep up in real-time is not going to be implementable.

Unprocessed variation 1 may have the highest overall accuracy, but it actually performs the worst in regards to what is most useful during steel production; a method which misses more than a quarter of its defects is practically useless.

TABLE 3.10: Time taken to train the classifier for each method.

| Method | Speed [seconds] | Feature vector size |
|---|---|---|
| Variation 1 | 332 | $4000 \times 676$ |
| Variation 2 | 238 | $4000 \times 100$ |
| Variation 3 | 369 | $4000 \times 1700$ |

Both variations 1 and unprocessed variation 2 have much lower false positive rates than processed variation 2 and both variation 3, but 3-10% extra false defect processing is not a huge cost to pay for the lower false negative rates. Processed variation 3 has the lowest false negative rate, which is a key driver in steel surface inspection. So if one was evalutating these methods based purely on their ability to not miss defects, then variation 3 would be the best option.

However, given the consideration of speed, and the 0.9% higher false negative rate, processed variation 2 could potentially be a more viable method than processed variation 3. It strikes a good balance between having a low false positive rate, fairly high overall accuracy, fairly low false positive rate, and the fastest speed.

One of the conclusions from the review paper by Sun et al. is the suggestion of a rise in combining methods to increase accuracy in defect classification [82]. Using just corrected images and the random forest classifier produces poor results in comparison to the combined use of random forest with Gabor filters and/or histograms to extract features.

### 3.5.4 Case-Study Conclusions

It is important that detection methods in steel production can run in real-time. The results show that combining these methods does not increase the overall accuracy of results. However, the false negative rate improves from an unacceptably high rate, to a more acceptable rate. Whilst unprocessed variation 1 has the highest overall accuracy rate, it may not be the best option for use in steel production as the false negative rate is too high. Processed variation 1 and unprocessed variations 2 and 3 share similar results, with slightly lower overall accuracy results, but more acceptable false negative rates. On the basis of false negative rate alone, processed variation 3 would likely be the best method to use in steel defect detection; however its overall accuracy is significantly lower than the other methods. Processed variation 2 presents as the best method when considering false negative and overall accuracy.

On the contrary, accuracy is not the only factor that should be considered. Ideally any method used in steel detection should be able to run at real-time production speeds. Taking into consideration the speed it took to train the classifiers and the size of the feature vectors, processed variation 2 would also be the more suitable variation as it uses a considerably smaller feature vector than variations 2 and 3 which requires less processing time.

Further work on this case study would be to look into implementation on a pilot line or alongside existing systems on the production line to test the methods in real-time. It was not possible in this instance due to time restraints.

The data suggests that these extra steps may be necessary to achieving better results and speeds. However the choice of which steps depends on what balance is required between accuracy and speed. A very complex algorithm may get higher accuracy results, but would likely run at a slower speed. In the case of steel defect detection, using either histograms or Gabor filters and histograms would offer the best results, depending on what speeds would be required.

## 3.6 Conclusions

Surface quality inspection is a pivotal part of steel production. Defects can damage machinery so it is imperative that none are missed. Given that any implemented steel detection methods are privately owned by companies, producing and testing our own methods with our own data is imperative for any future possibilities of expanding our in-house capabilities.

This section considered a steel defect detection method using a combination of methods including histograms, Gabor filters, and random forest classifiers. The combination of all three methods performs well, but with respect to steel production and real-time defect detection, the exclusion of Gabor filter may be more suitable if computational cost becomes a constraint. With the continuous improvement in hardware, software can achieve more in a shorter space of time, so testing the methods in real-time on a pilot line would be the next step to determine which combination of methods is optimal for real-time defect detection.

The training speed results in Table 3.10 shows that the feature vector size was not drastically affected by the feature size at this level. So since none of the training and testing episodes took more than a few minutes each, 1700 data points for each image was considered a reasonable size. However, further work could include testing different dimensional reduction methods to see which works best to reduce noise as well as speed up the process without a reduction in accuracy.

## Chapter 4

# Using R-CNNs and Faster R-CNNs to determine the quality of weld holes

## 4.1 Introduction

This chapter investigates the application of a selection of state of the art methods in classifying a particular steel defect. The aim was to create a weld hole quality classifier using image data supplied by Tata Steel UK. Three different methods were tested and compared to determine which methods, if any, were suitable.

Computer vision and classification problems research has been in and out of popularity since the creation of modern computers, but the latest wave of popularity in neural networks and deep learning research started around 2006 [120]. Around this time a group of researchers in the USA began the creation of the ImageNet data set [121]. ImageNet is a data set of over 14 million images with over 20,000 categories which is used by researchers across the world to benchmark advances in image classification [122].

In 2010 the ImageNet data set was used in the first ImageNet Large Scale Visual Recognition Challenge; a competition available for any researchers to submit their latest advancements in machine learning. Between 2010 and 2017 the competition ran annually and was entered by many major research groups. In this short period of time there were significant increases in the accuracy of entrants methods [123]. In 2014 a PhD student at Stanford University competed by classifying images himself, to get an idea of what accuracy humans would be able to achieve compared to computer vision methods. In his blog post about the challenge [124], he describes how those unfamiliar with the ImageNet data had up to a 15% error rate, whereas his error rate was 1.2% less than the winning computer vision method that year. By the following year there were methods which outperformed him. The results of the winning entry for each year, plus the results of the only human to attempt the competition, can be seen in Figure 4.1.

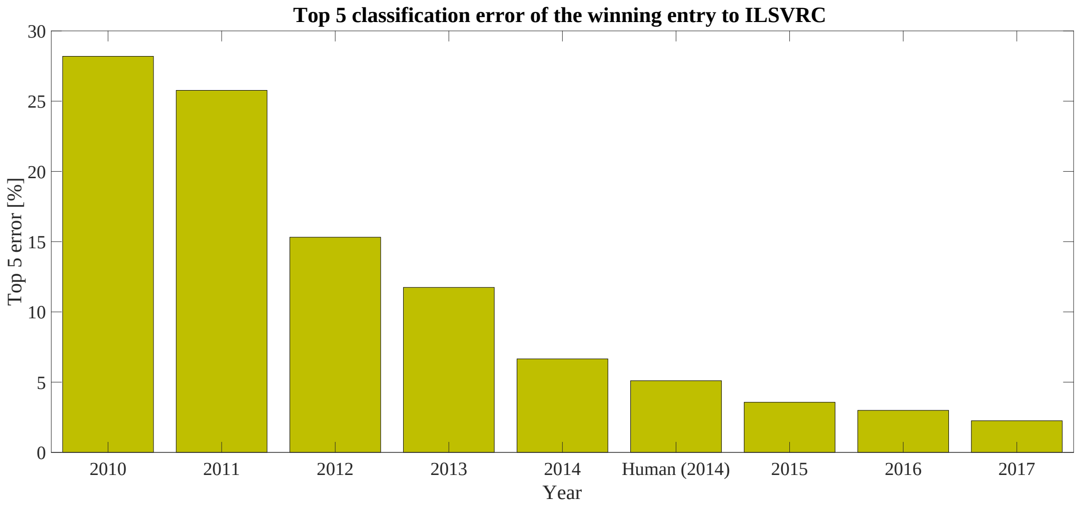


FIGURE 4.1: The top 5 classification error of the winner of each year of the ImageNet Large Scale Visual Recognition Challenge, including the result of the only human to complete the challenge.

The significant decrease in error since 2012 was down to the use of CNNs. The winner in 2012 was a network called AlexNet, the first to use CNNs, and every winner since 2012 has been a network based on CNNs [125]. A convolutional neural network (CNN) is a type of deep neural network designed for image classification specifically. The networks adapted in this chapter, GoogleNet and ResNet, were the winners of the ImageNet Challenge in 2014 and 2015.

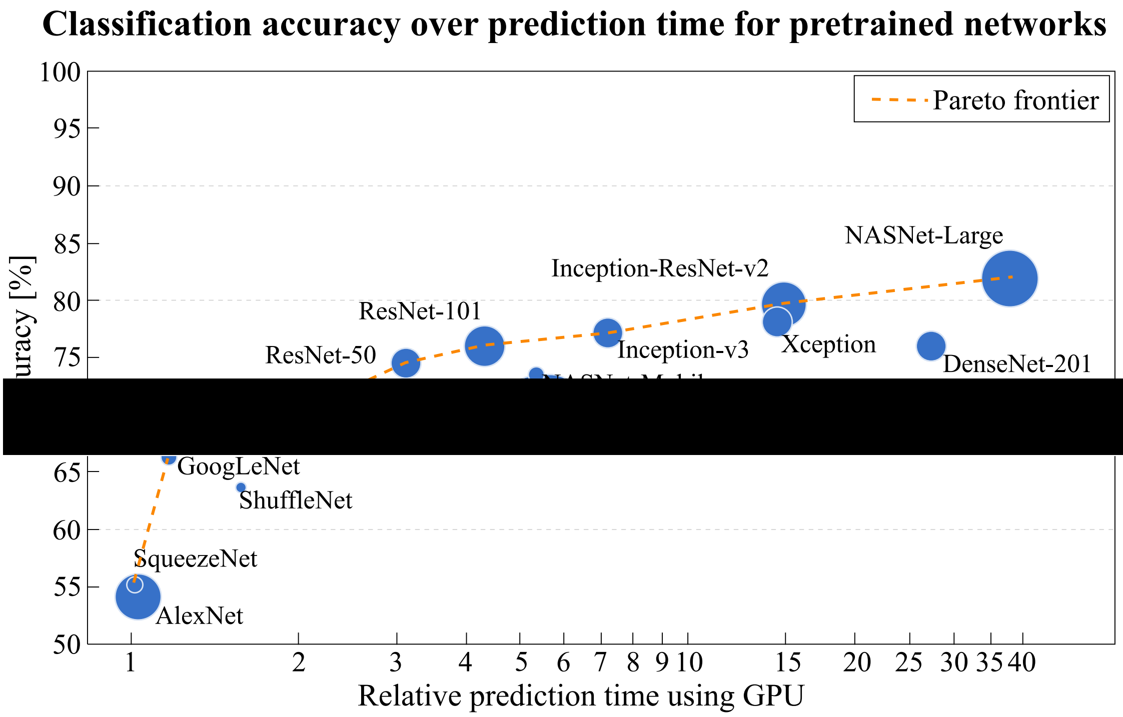
In this chapter these networks are used to classify three grades of weld holes. A weld hole classifier was selected for multiple reasons. Firstly, it is a defect type in which there can be 100% confidence in what it is by looking at an image of it. For most defect types a sample of the defect must be observed under a microscope to be certain of its classification. Given that this has not occurred with the data received from Tata, working on a classification problem using a data set of images and defect types which was predicted by a computer would not be accurate.

Secondly, a weld hole is a very important marker in production and secondary use of material. Weld holes are a deliberate feature in steel, and not a defect. They are a hole that is punched into the steel next to the seam of two coils which have been welded together. When weld seams pass through certain machines, the speed at which it goes through needs to be reduced to minimise risk of breakage or damage. Tracking a weld hole enables tracking of where in the coil a weld seam is. This enables the calculation of when the weld seam will go through a certain part of machinery, meaning that the run speed can be reduced at the right time and for as short a period as possible so as not to waste production time. This tracking also enables coils to be split into shorter coils by cutting either side of each weld line for minimal waste.

Over time the punch which makes the weld holes degrades and is eventually replaced. This degradation leads to poor quality weld holes. The punch can stick to the steel if its surface is degraded, or the circle which gets punched out may not be fully punched out, and can remain stuck to the steel surface, or get folded over and rolled into the steel surface. All of these issues can make the weld hole confusing for classifiers to detect, meaning that the weld hole could be mis-classified and the tracking that a weld hole provides could be lost.

## 4.2 Method

The general method is to select networks, train them using a subset of the data set and then test them on the remaining data. Certain elements of the networks could be adjusted, so variations in these elements were trained and tested. Which networks to test was decided by which networks were available. Figure 4.2 is a graph of the classification accuracy and speed of prediction of the pretrained networks available in Matlab in 2019. The dataset used in all tests was the ImageNet data set and the GPU is an NVIDIA Titan Xp. The area of each blue marker is approximately proportional to the network's disk size [126]. Whilst one cannot expect the exact same results when these networks are applied to other tasks, this graph gives an indication of what these methods are capable of when deciding which to select.



[redacted]

FIGURE 4.2: The ImageNet validation accuracy of various pretrained networks over prediction speed. The area of the markers represent the network's disk size [126].

The ImageNet data set is used as a benchmark to enable fairer comparison between classification methods. The methods on the pareto frontier are ones which have no superior method in all of the observed measures, in this case prediction speed and accuracy. The methods which are not on the pareto frontier are AlexNet, ShuffleNet, NASNet-Mobile, VGG-16, VGG-19, Xception, and

DenseNet-201. GoogLeNet, ResNet-50, and ResNet101 were the methods selected to test.

The data set is made up of 764x240 pixel images, so the minimum image size required for the network needed to be less than 240 pixels to ensure our data would fit. Methods ResNet-50, ResNet-101, and GoogLeNet were selected, as their image input sizes are all 224x224 pixels. All three of these networks are on the pareto frontier, but their accuracies and relative prediction times vary. GoogLeNet is fast, but has relatively low accuracy, and ResNet-50 is in the middle, with higher accuracy, but slower prediction time, and ResNet-101 is the slowest, but most accurate.

After selecting the networks, they were adapted to become faster R-CNNs. This was successful for the ResNet networks, but the GoogLeNet network failed to function when transformed, so it was only trained as an R-CNN, and not a Faster R-CNN.

Five sets of tests were carried out, each repeated three times. The data set used were all split into 80% training data, 20% test data with a similar proportion of each class type in the training and test sets. Initial testing was carried out on the Faster R-CNN ResNet-50 network, to get an idea of what effect different parameters have. Then Faster R-CNN ResNet-101 and R-CNN GoogLeNet were tested.

### 4.2.1 The Networks

**ResNet-50** ResNet-50 is a convolutional neural network which has 50 layers [127]. In Matlab its feature extraction layer is called 'activation_40_relu' and its ROI pooling layer has output size [14 14] and is inserted after the feature extraction layer.

**ResNet-101** ResNet-101 is a convolutional neural network which has 101 layers [128]. In Matlab its feature extraction layer is called 'res4b22_relu' and its ROI pooling layer has output size [14 14] and is inserted after the feature extraction layer.

**GoogLeNet** GoogLeNet is a convolutional neural network which has 22 layers [129]. This method was not successfully transformed from its pre-trained network into a Faster R-CNN, so was trained as an R-CNN.

The GoogleNet network was first proposed by a collaboration of researchers from Google Inc., University of North Carolina, and University of Michigan and won the ImageNet competition in 2014 [130]. The network has the same initial layers as a traditional CNN, but then stacks specifically designed modules of layers, with max-pooling layers every so often to halve the resolution for efficiency purposes. These modules include a direct connection between nodes and some deeper branches, as can be seen in Figure 4.3.

The ResNet networks were first proposed in 2016 by a research team at Microsoft [131]. The aim of Res-Nets was to create a residual learning framework to make training deep networks easier. Microsoft's paper highlights that training error degrades as more layers are added to a network and, like the GoogLeNet network, makes use of stacks of layers which include a direct connection to solve this issue. By including the ability to essentially skip a layer or group of layers, it means that adding these extra stacks of layers doesn't add further error to the network. Networks of 50, 101, and 152 layers were proposed, and all were based on a 34 layer network which then had extra stacks of layers added. Testing all their networks on the ImageNet dataset, the top-1 error was decreased from 28.5% to 22.8%, 21.7%, and 21.4%, respectively. An ensemble of the 152 layer network won the 2015 ImageNet competition.

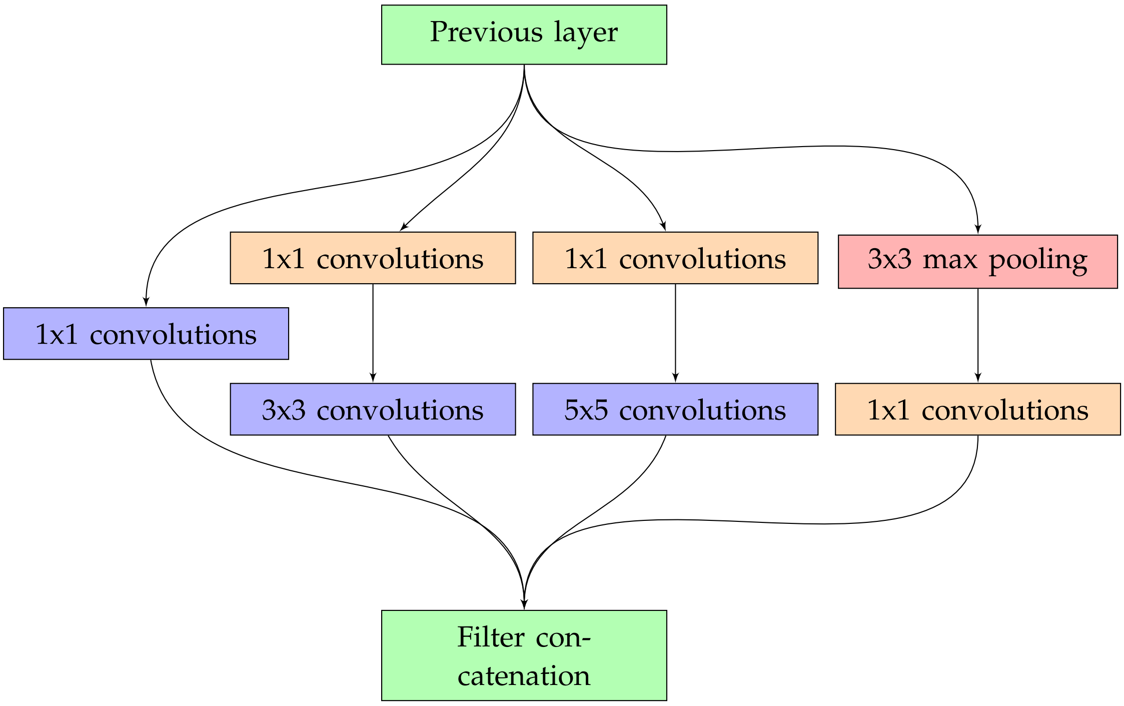
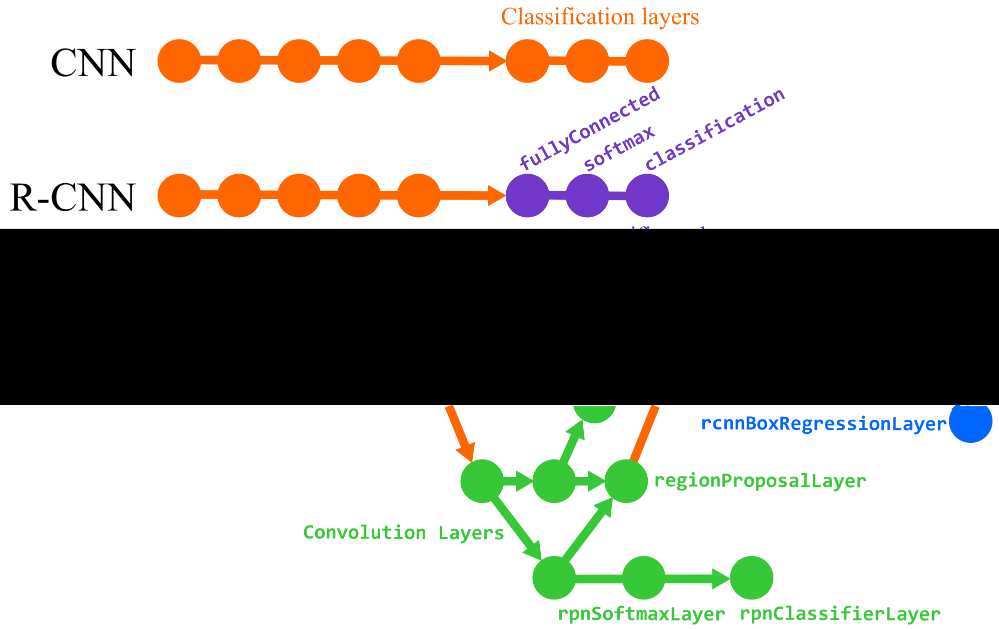


FIGURE 4.3: The modules stacked in the GoogLeNet network.

### 4.2.2 Adapting the CNNs to Faster R-CNNs

This section is based on the MathWorks documentation [132, 133] for creating customisable Faster R-CNNs, unless otherwise stated. Figure 4.4 shows the stages of adapting CNNs into R-CNNs and then Faster R-CNNs, where circles represent layers.



[redacted]

FIGURE 4.4: The layers changed to adapt a C-NN to an R-CNN and a Faster R-CNN [132].

CNNs are convolutional neural networks, which are neural networks specifically designed to work with images by using feature specific image filters on its nodes to create a feature map. These can be transformed into region-based CNNs (a.k.a. R-CNNs) with some small changes to the architecture of the network. R-CNNs generate region proposals using an edge boxes algorithm. These proposed regions are cropped out and resized. The CNN then classifies the cropped and resized regions. The region proposal boxes are then refined by an SVM which is trained using the CNN features. Since a pre-trained network is used, the classification layers are taken out and replaced with three layers called `fullyConnected`, `softmax`, and `classification` which will be specific to the new data the network will be trained on.

A fully connected layer is a layer where all neurons are connected to all neurons from the previous layer. The layer is used to multiply its input by a weight matrix and add a bias vector. This combines all of the features from the previous layers to identify patterns in the image [134].

A softmax layer applies the softmax function to the previous fully connected layer. The softmax function, seen in equation 4.1 [135], is used in a multi-class classification problem to give the probability of $x$ belonging to the $\kappa$-th class, $\mathcal{C}_\kappa$, where $p(x|\mathcal{C}_\kappa)$ is the conditional probability of the input $x$ given the $\kappa$-th class and $p(\mathcal{C}_\kappa)$ is the class prior probability [136].

$$P(\mathcal{C}_\kappa|\mathbf{x}) = \frac{p(x|\mathcal{C}_\kappa)p(\mathcal{C}_\kappa)}{\sum_j p(x|\mathcal{C}_j)p(\mathcal{C}_j)} = \frac{\exp(a_\kappa)}{\sum_j \exp(a_j)} \tag{4.1}$$

A classification layer calculates the cross entropy loss for each mutually exclusive class in a multi-class classification problem [137]. The classification layer follows the softmax layer, using the size of the softmax layer to infer the amount of classes it needs to calculate loss for.

From R-CNNs, these networks can then be adapted to become Faster R-CNNs. The idea behind this is that the fastest possible method is required when these weld holes are tested in real-time, so it makes sense to test the fastest possible versions of these networks. Convolution layers are added in and then input into a box regression layer, `rcnnBoxRegressionLayer`, which redefines the bounding boxes using a smooth L1 loss function [138]. These convolution layers are also input into a softmax, `rpnSoftmaxLayer`, and classification layer, `rpnClassifierLayer` to determine whether the input image regions are background or objects. The outputs of the box regression layer and the classifier layer are input into a region proposal layer, `regionProposalLayer`, to output a list of bounding boxes around potential objects. This list of bounding boxes, along with the feature map created in the feature extraction layer, go into a max pooling layer, `roiMaxPooling2dLayer`, to create feature maps for each ROI. These ROIs are further redefined in the box regression layer before the final classification is carried out.

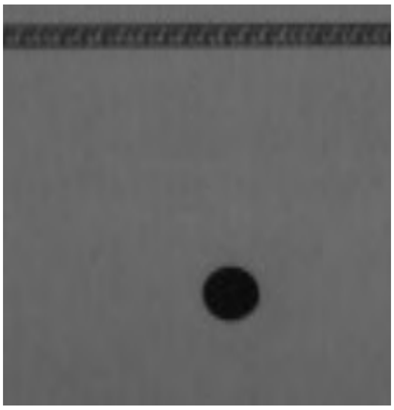
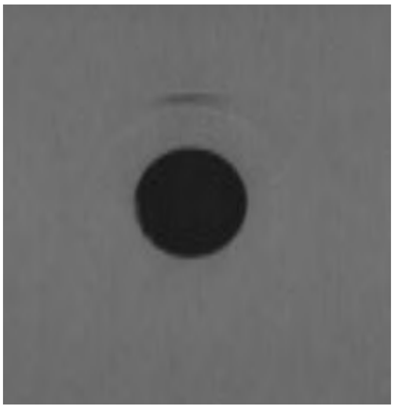
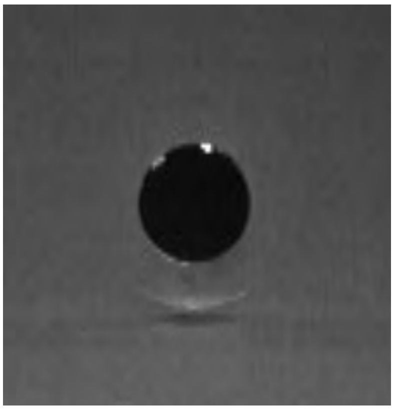
The positive and negative overlap are defined using the overlap ratio of a bounding box of a sample image over the bounding box of the ROI which is defined in equation 4.2 where $A$ and $B$ are bounding boxes. The positive overlap ratio defines by how much a sample must overlap the ROI to be used as a positive training sample. The negative overlap ratio defines how much a sample must not overlap the ROI to be used as a negative training sample.

$$\frac{area(A \cap B)}{area(A \cup B)} \tag{4.2}$$

### 4.2.3 Data

A set of 254 weld hole images, some of which feature two weld holes, was gathered from the data provided by Tata Steel Europe. These were than separated into three grades, descriptions and examples of which can be seen in Table 4.1.

TABLE 4.1: Weld hole grades.

| Grade # | Number of samples | Description | Example |
|---|---|---|---|
| 1 | 71 | A good quality weld hole with clearly defined edges and no surrounding markings |  |
| 2 | 39 | The hole has a clearly defined edge, but the punch has stuck to the surface leaving an imprint and a wrinkle above and/or below the hole. Indicates degradation of the punch |  |
| 3 | 145 | The hole has poorly defined edges, possibly including bits of cut-out left behind, and the punch has left wrinkling/imprints on the surface. Indicates punch requires replacement. |  |

There were 71 images of grade 1 weld holes which are a good quality weld hole with no surrounding markings and well-defined edges. There were 39 images of a grade 2 weld hole which is where the hole is clearly defined, but there are edge markings surrounding the hole. There were 145 grade 3 weld holes which have debris left around the holes or unclear edges, and marks around the hole. This slight imbalance in number of samples may lead to confusion in training so both equal and unequal numbers of samples were tested.

Augmenting the data to increase the size of the data set was considered. However, given the fact that steel production lines move in one direction, the appearance of its weld holes, and defects, are all directionally dependent. The cameras are also a fixed distance away from the steel, so an image of a weld hole is always going to be approximately the same size and in a handful of set locations, depending on which cameras have picked it up. This means that rotation and scaling of images would produce training data that was nothing like the type of test images that the classifier would encounter in the real world. So any results achieved on an augmented data set would not necessarily be replicable when the same method was applied live to the production line. This is why further work on this application would include gathering a much larger data set of weld hole images to carry out further testing.

Table 4.2, Figure 4.5, and Figure 4.6 shows some analysis of the ROI bounding boxes to enable the selection of initial anchor boxes for the region proposal network in the Faster R-CNNs tested. Anchor boxes are predetermined boxes which are selected to capture the scale and aspect ratio of specific object classes that we want to detect. The use of anchor boxes enables a network to detect multiple objects of different sizes and scales by calculating the probability of objects being present in each anchor box as it is tiled across the image. This method enables all predictions to be evaluated at once, instead of having to use the slower sliding window method to check for each type of object in each window of the whole image. [139].

Bounding boxes typically have an aspect ratio of 1.6:1 in width:height and their areas are approximately either 2000 pixels, or 5000 pixels. Based on the analysis of the ROIs, the anchor box widths and heights were selected, as seen in Table 4.3.

TABLE 4.2: Analysis of ROI bounding boxes.

| Method | Width [pixels] | Height [pixels] |
|---|---|---|
| Range | 62 | 48 |
| Minimum | 44 | 20 |
| Maximum | 106 | 68 |
| Mean | 74.82 | 43.36 |
| Mode | 52 | 33 |
| Median | 84 | 39 |

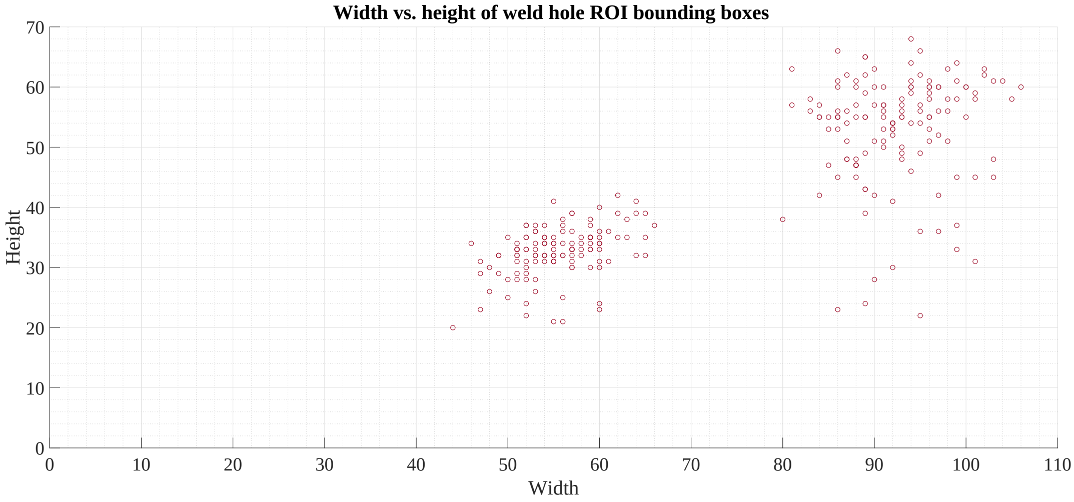


FIGURE 4.5: The width vs. the height for the weld hole sample images.

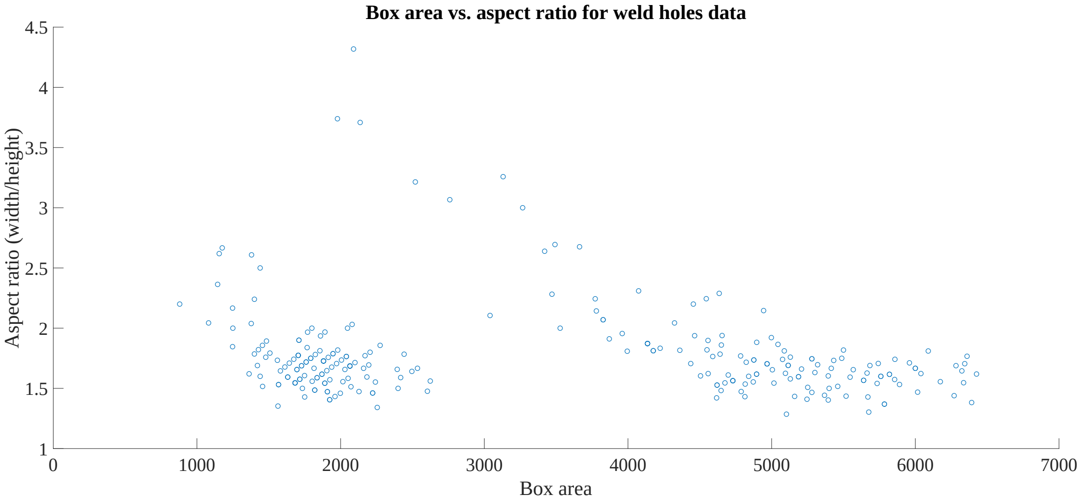


FIGURE 4.6: The aspect ratios and areas for the weld hole sample images.

TABLE 4.3: Selected anchor boxes.

| **Width [pixels]** | **Height [pixels]** |
|---|---|
| 52 | 32 |
| 60 | 40 |
| 94 | 60 |
| 92 | 30 |
| 50 | 24 |
| 86 | 54 |

### 4.2.4 Training the networks

Training the networks was carried out using the `trainRCNNObjectDetector` and `trainFasterRCNNObjectDetector` functions. The solver used was stochastic gradient descent with momentum, the initial learning rate was set to 0.001, and maximum number of epochs was set to 7, as is shown in this example [140].

**Set A** Set A uses the Faster R-CNN ResNet-50 network with 203 training images and 51 test images. The negative overlap was set to $[0, 0.1]$ and positive overlap set to $[0.8, 1]$.

**Set B** Set B uses the Faster R-CNN ResNet-50 network with 203 training images and 51 test images. The negative overlap was set to $[0, 0.05]$ and positive overlap set to $[0.95, 1]$.

**Set C** Set C uses the Faster R-CNN ResNet-50 network with the data cut down so that there is an equal number of images, 38, in each class type. This gave only 114 images, which were split into 91 training images and 23 test images. The negative overlap was set to $[0, 0.1]$ and positive overlap set to $[0.8, 1]$.

**Set D** Set D uses the Faster R-CNN ResNet-101 network with 203 training images and 51 test images. The negative overlap was set to $[0, 0.1]$ and positive overlap set to $[0.8, 1]$.

**Set E** Set E uses the R-CNN GoogLeNet network with 203 training images and 51 test images. The negative overlap was set to $[0, 0.3]$ and positive overlap set to $[0.7, 1]$.

### 4.2.5 Quantifying accuracy

Three measures were calculated to measure accuracy; The percentage of top label correct was calculated in two forms, using either the mode (most frequent) label which occurred in each test image or selecting the single label with the max score. The two most frequent labels were also considered as another measure.

Taking the top-2 labels will generally give an equal or higher accuracy, however given that in our case there are only three classes, taking two labels would only be meaningful if we could check that the defect was predicted either class 1 and 2 (mostly good quality, but beginning to mark around edge of hole), or class 2 and 3 (degrading quality). Although a defect cannot be accurately predicted as class 1 and 3 due to their definitions.

## 4.3 Results

Table 4.4 has the full results from each run. On initial observation, it is surprising just how different the results can be from two versions of the same network. Some difference was expected, due to differing initial weights, but in some cases the results of one network can be twice that of another with the same parameters. There are unfortunately no convergence results for these test runs as further investigation found that convergence data was not automatically collected. After re-training one run from each set to get convergence information, sets B, C, and D did not converge, so for the runs which performed very poorly, it is likely that the training process never converged in these instances either. Sets A and E convergence had both somewhat levelled out, but may have benefited from further training.

Out of the top-1 label results, the prediction with the maximum score has the highest accuracy in most runs tested. For each set of runs, the mean of the top-1 maximum score has either equal or highest accuracy. This is likely because the maximum score represents the prediction with the highest confidence, whereas using the mode takes most frequent prediction, but their confidence could be low. There are instances where top-1 and top-2 measures have equal accuracy. However, the top-2 most frequent labels are observed to always contain the highest accuracy of all three measures, both for individual runs and for the mean accuracy for each network set. This makes sense, given that there is only 3 possible outcomes and the top-2 score considers two labels instead of one label.

It would seem logical that negative samples would want to overlap the ROI as little as possible to ensure none of the weld hole is contained and avoid confusion; and the positive samples would want to overlap the ROI by a good amount to make sure enough of the defect is actually in the positive sample. However, setting these values to the extreme possibly means that the network cannot find enough positive and negative samples within the images. This would explain why the results for test set B are so poor, and somewhat random.

The top-1 results from set C, using an equal amount of each data type, are worse than that of set A which uses the same configuration, but a larger, biased set of images. Set C has mean prediction accuracy of 50.73% and 52.17% for its top-1 mode and max score results, respectively, whereas set A has 60.13% and 60.79%.

TABLE 4.4: Results for each run.

| **Set** | **Run #** | **Top-1 label** | | **Top-2 [%]** |
|---|---|---|---|---|
| | | **mode [%]** | **max score [%]** | |
| A | 1 | 52.94 | 52.94 | 52.94 |
| | 2 | 60.78 | 64.71 | 76.47 |
| | 3 | 66.67 | 64.71 | 80.39 |
| | Mean | 60.13 | 60.79 | 69.93 |
| B | 4 | 7.84 | 7.84 | 7.84 |
| | 5 | 37.25 | 37.25 | 37.25 |
| | 6 | 1.96 | 1.96 | 1.96 |
| | Mean | 15.68 | 15.68 | 15.68 |
| C | 7 | 43.48 | 47.83 | 60.87 |
| | 8 | 65.22 | 73.91 | 86.96 |
| | 9 | 43.48 | 34.78 | 56.52 |
| | Mean | 50.73 | 52.17 | 68.12 |
| D | 10 | 45.1 | 43.14 | 45.1 |
| | 11 | 62.75 | 64.71 | 66.67 |
| | 12 | 68.63 | 76.47 | 80.39 |
| | Mean | 58.83 | 61.44 | 64.05 |
| E | 13 | 62.75 | 76.47 | 80.39 |
| | 14 | 68.63 | 80.39 | 84.31 |
| | 15 | 64.71 | 80.39 | 86.27 |
| | Mean | 65.36 | 79.08 | 83.66 |

However the top-2 results are very similar at 68.12% and 69.93% for set A and C, respectively. There is a trade-off between data set size and bias; in this instance, working with such a small data set, having a larger number of samples is more beneficial than having un-biased samples.

Set D, using ResNet-101, has a small accuracy increase in the mean max score measure over set A and, individually, the highest increase comes from run 12 which has a max score accuracy 12% more than run 3, which has the highest max score accuracy for set A. However, the top-1 mode and the top-2 label accuracies are, for the most part, fairly similar, as runs 1-3 and 10-12 have quite wide spread of accuracies across the runs.

Set E, using GoogLeNet, has the highest mean accuracies compared to all other sets; the set also has a smaller range of accuracies in each measure compared to the other sets. Whilst some individual runs from other sets have a higher top-2 and top-1 mode accuracies than runs 13 to 15, runs 14 and 15 have the highest top-1 maximum score accuracy compared to all other runs. Based on these 15 trials, the GoogLeNet network appears to consistently produce the highest accuracy.

Given the small size and fast training time of the network used in set E, this classifier was able to be retrained with 100 epochs to give an idea of how close to convergence the set E classifiers were and how this extra training would increase accuracy. The re-trained network converged and achieved a top-1 max score accuracy of 82% and a top-1 mode accuracy of 88%. This is an improvement on the average of 65% and 79% achieved in Set E, but still not a high enough accuracy to be considered for reliable industry use.

Figure 4.7 shows a selection of results images. The red boxes are the predicted boxes and the blue boxes are the labelled ROIs. In general, a grade 3, poor quality, weld hole was predicted most often, but this is likely down to the sample set having more grade 3 weld holes than any other type. The size and number of anchor boxes appears to have been adequate as the anchor boxes suitably cover the weld holes detected.

Having gone through all of the test images for each run, like those shown in Figure 4.7, one noteworthy result is that the network configurations used for sets A, D, and E were almost flawless at detecting the location of the weld holes. Every possible detection that was predicted overlapped the labelled ROI by at least 50%, regardless of how small the probability score was. So whilst none of these methods have a high enough accuracy when classifying the type of weld hole, they are suitably accurate at detecting its location, which could still be useful in future work.

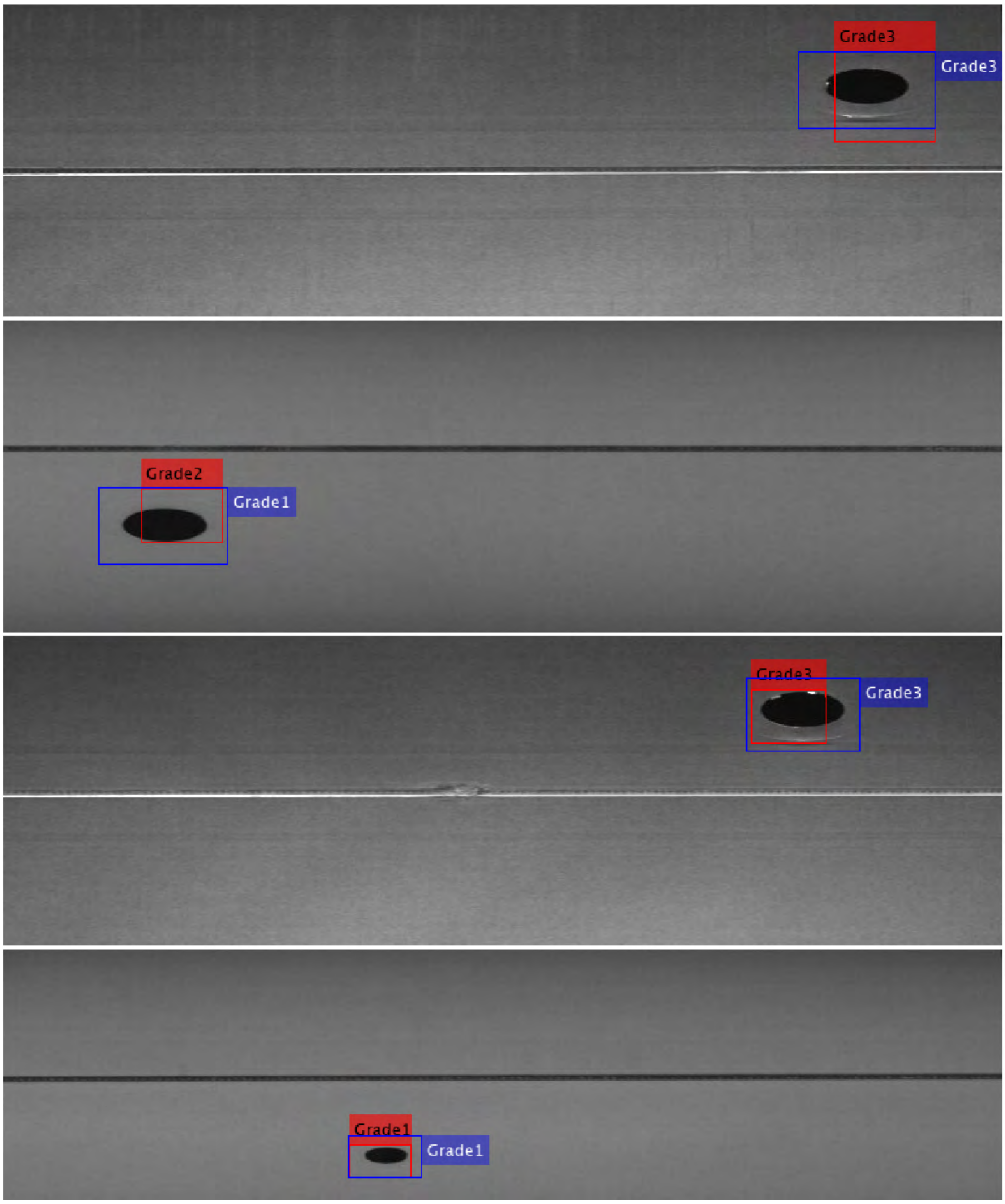


FIGURE 4.7: Results images for weld hole grading.

## 4.4 Conclusions

Accurate weld hole quality classification would create a method of predictive maintenance of punch quality. This piece of work attempts to apply some state-of-the-art methods to classify the quality of a weld hole. Using Faster R-CNN ResNet-50, Faster R-CNN ResNet-101 and R-CNN GoogLeNet networks did not yield the results hoped. The R-CNN GoogLeNet performed best out of the three networks, but not to the sufficient accuracy required for industrial use. However these tests do highlight some important points.

The size and spread of the data set was a limitation in this experiment. Given more images and labelling time, a larger data set could be made which may produce high accuracies, but it is not possible to predict how much improvement this would yield. The spread of the data over the different classes should ideally be equal. However this was not the case for the data set available and this could impact the results, but, again, it is not possible to predict the extent of its effect.

The positive and negative overlap determine what is considered a positive or negative sample within the images. The values set for these overlaps have a significant effect on the accuracy of the networks and should be chosen carefully, with consideration for how much of the labelled ROIs actually contains the weld hole.

The thought behind adapting these existing networks into R-CNNs and Faster-RCNNs is that these methods were, at the time of writing, some of the newest deep learning object detection methods available in Matlab. There exist other methods and combinations of methods which could be considered in further work with this application. However this chapter is designed to be a proof of concept and is not a recommendation that these are the 'best' methods for this application.

Further work in this chapter would include investigation into a two-step method which uses a Faster R-CNN to detect the weld hole and then a second classifier to differentiate between quality of weld hole. It would also include further investigation into the convergence of the networks and testing different learning rates to aid convergence.

# Chapter 5

# Colour and its effects on defect image contrast

## 5.1 Introduction

The use of computer vision in the steel industry faces some unique environmental obstacles. Any machine learning method can only learn well if it is given appropriate data. This means that a visual inspection system monitoring steel defects will need to be trained and testing images of steel which have very little noise, to adequately differentiate between high quality steel surface and a defect. Steel is not produced slowly in a cold, clean room with plenty of space for optimal lighting and camera angles. Steel mills can be fast-paced, hot, dirty, and visual inspection systems are added in to what space is available. These environmental challenges mean that taking an optimal image of the steel surface is difficult.

This chapter of research into the effects of colour was based on an idea proposed by an LED manufacturer suggesting that green light could highlight oil spots [141]. If this suggestion was true, there were many possibilities. So the impact of colours, light and their effects on contrast between defects and clean steel surface was explored, in pursuit of an optimal image for machine learning in defect detection.

There is limited research into the area of greyscale methods and their effects. Since common usage of them is for photographic effect and not for a practical gain, it is understandable that this is the case. A paper highlighting the differences in outputs of different colour-to-greyscale methods by Kanan and Cottrell [142] asks ′does it matter?′ and concludes that colour-to-greyscale methods do influence machine learning performance. So one of the aims of this research is to determine if any method(s) offers a better defect/non-defect contrast than others.

This chapter looks at the use of colour cameras and filters to gain an image of high contrast between defects and the rest of the steel surface. The four questions investigated in this chapter are:

1. Does the greyscale method used affect contrast, and if so which method gives the highest contrast between defects and clean steel?
2. Does colour affect contrast, and if so which colour gives highest contrast between defects and clean steel?
3. Is the optimal greyscale method from question 1 still optimal when colour filters are used?
4. Does green filtering of images improve contrast of oil spots?

### 5.1.1 Colour

This section discusses the physical way that humans see colour and how we translate that to a computer screen, based on [143, 144, 145] unless otherwise stated. The human eye does not measure colour directly, but instead constructs colour based on processing photoreceptor signals. The photoreceptors in the eye are of two types; rods and cones. Cones are sensitive to colour and rods are sensitive to light intensity. There are then three types of cones, short, medium, and long; that is not to say these terms describe the size of the cones, but rather the size of the wavelength of light they are sensitive to.

Light is defined as the visible range of the electromagnetic spectrum; its wavelength range is approximately 360-780nm. Different lengths of wavelength are a different colour; for example red has the longest wavelength at around 700nm and violet has the shortest wavelength at around 380nm. 'Long' cones have maximum absorption at 563nm and make up approximately 64% of all cones in the eye; 'medium' cones have maximum absorption at 534nm and make up 32% of all cones and only about 4% are 'short' cones which have maximum absorption at 420nm [146]. Figure 5.1 shows the normalised cone responses to the range of visible wavelengths of light and the approximate colour that is produced at that wavelength. The values for these functions are obtained from a website maintained by the Colour and Vision Research Laboratory, in the Institute of Ophthalmology at University College London, called *Cone Fundamentals* [147] which reproduces this graph, and others, in various forms based on existing research into cone sensitivity levels [148, 149, 150].

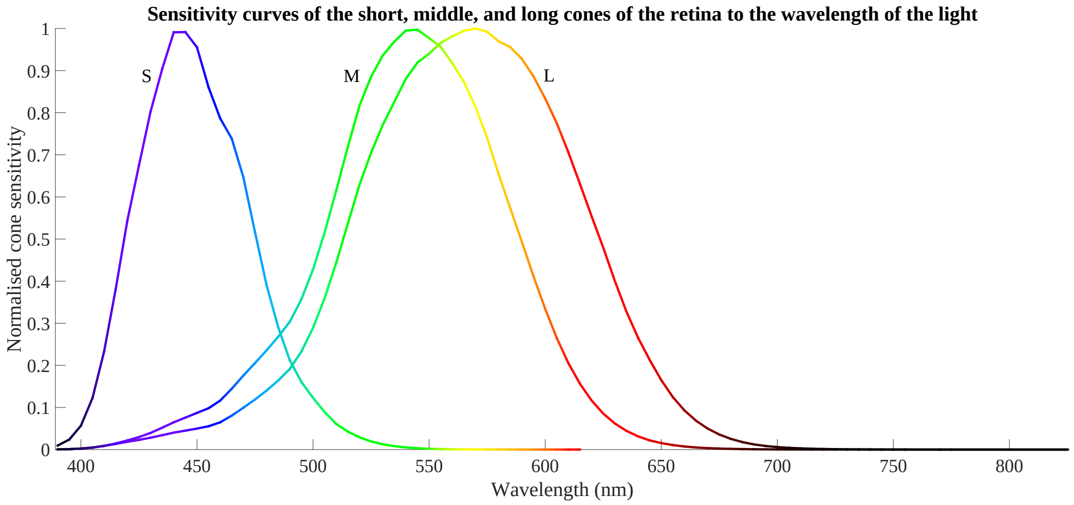


FIGURE 5.1: Graph showing the short (S), medium (M), and long (L) cone sensitivities to the wavelengths of visible light.

There are several different colour spaces which have been studied and suggested during the last century. In 1931 the Committe International de l'Eclairage constructed what is now an international standard, the CIE chromaticity diagram in Figure 5.2, which links the wavelength of light to perceived colours. The CIE diagram projects the XYZ 3-D colour space to 2-D via the coordinates calculated in equations 5.1, 5.2, and 5.3. In these equations $C(\lambda)$ is the amount of photons at wavelength $\lambda$ and the three functions $\mu_X(\lambda)$, $\mu_Y(\lambda)$, and $\mu_Z(\lambda)$ are given in Figure 5.3. Since $C$ and $\mu$ are positive, the scalars $X$, $Y$, and $Z$ are real and non-positive.

$$X = \int C(\lambda)\mu_X(\lambda)d\lambda \tag{5.1}$$

$$Y = \int C(\lambda)\mu_Y(\lambda)d\lambda \tag{5.2}$$

$$Z = \int C(\lambda)\mu_Z(\lambda)d\lambda \tag{5.3}$$

By calculating these scalars, the coordinates of the observed light can be calculated in the normalised CIE-XYZ chromaticity diagram by using equations 5.4, 5.5, and 5.6 where $x + y + z = 1$. Typically $z$ represents the luminosity, so a colour has the same x and y coordinates on Figure 5.2 even if the luminosity is different.

$$x = \frac{X}{X+Y+Z} \tag{5.4}$$

$$y = \frac{Y}{X+Y+Z} \tag{5.5}$$

$$z = \frac{Z}{X+Y+Z} \tag{5.6}$$

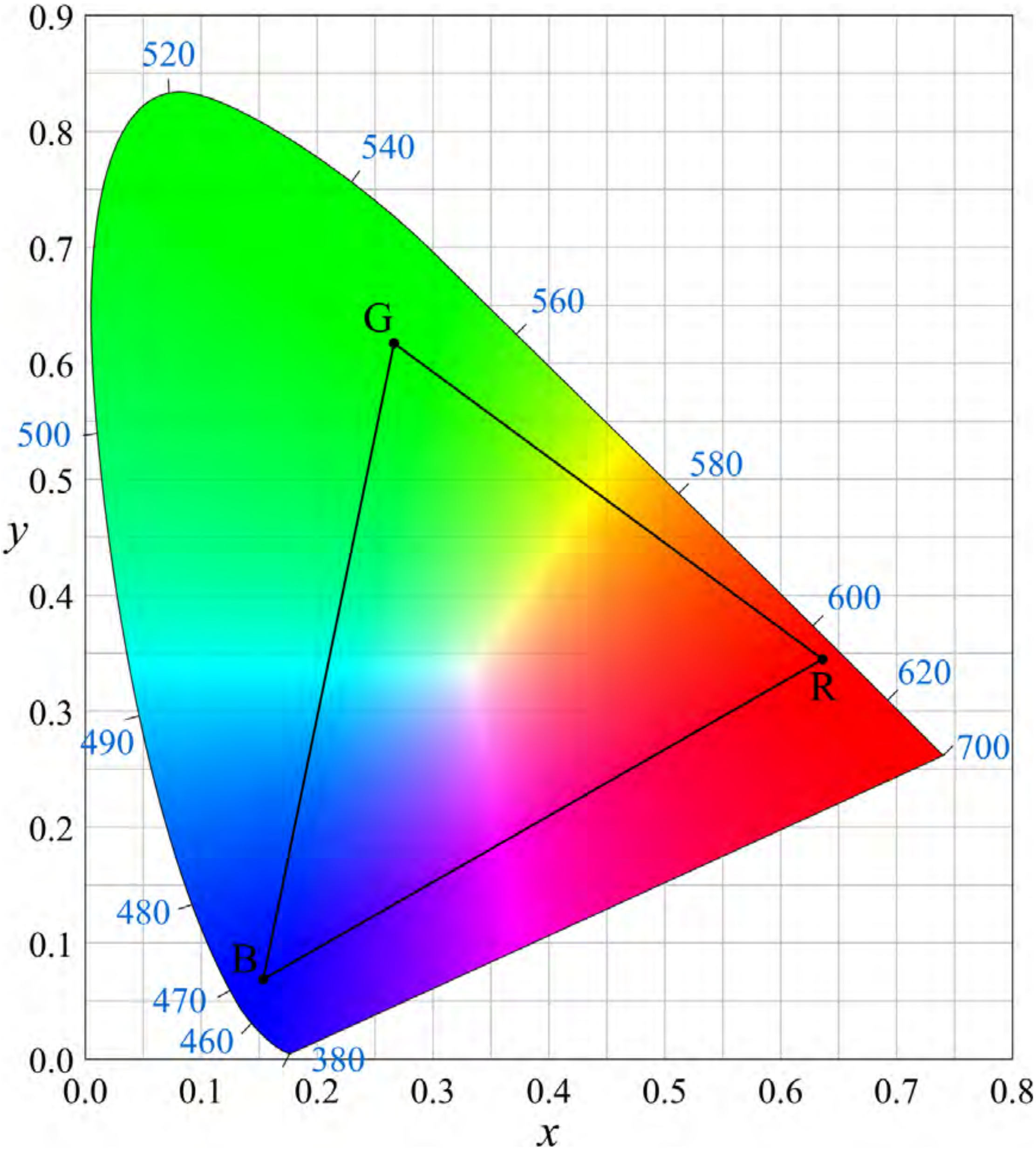


FIGURE 5.2: The 1931 CIE Chromaticity diagram with the RGB colour space highlighted [151].

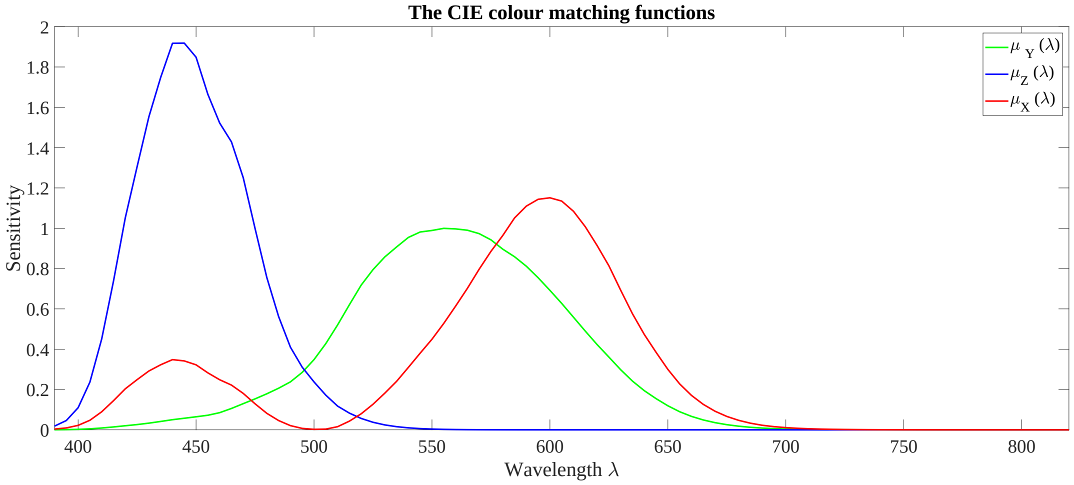


FIGURE 5.3: The colour matching functions used in the CIE diagram.

The RGB colour space is mapped out on the CIE diagram. Different colour spaces are represented by taking different triangles of the CIE diagram. Another mapping of the RGB colour space can be seen in Figure 5.4. Here the triple coordinate represents the intensity in the R, G, or B channel and the black to white line represents the luminosity which has equal parts of red, blue, and green.

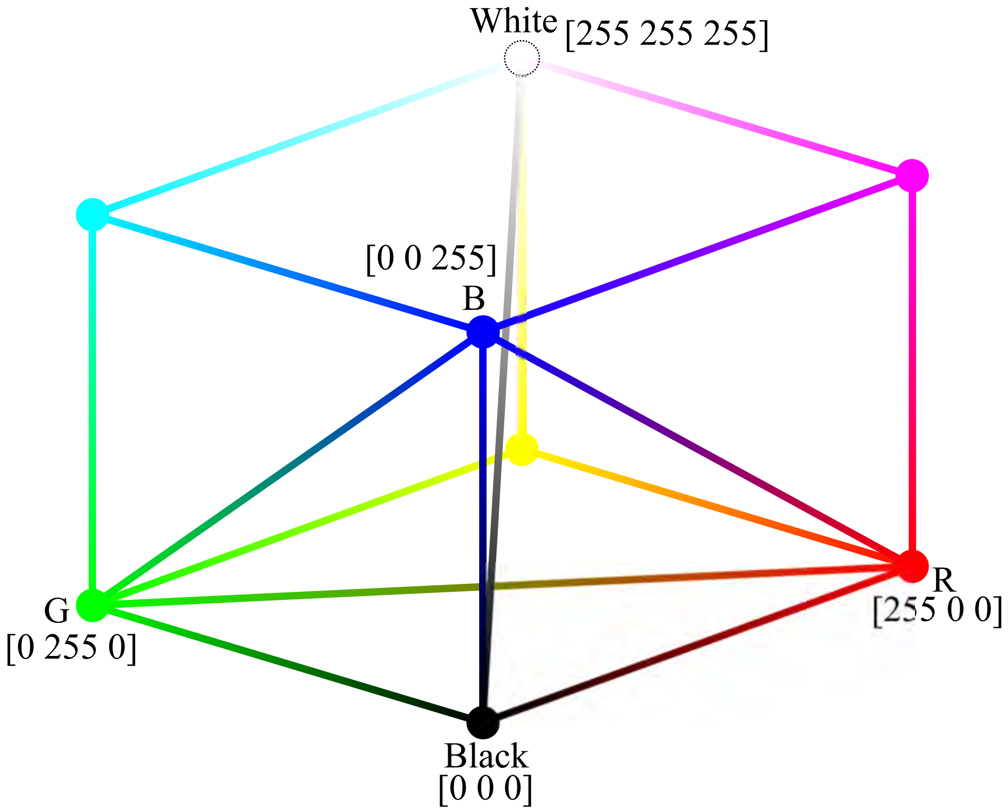


FIGURE 5.4: The RGB colour space represented as a cube.

Unlike the greyscale images dealt with in the previous chapter, colour images use a combination of multiple colour channels to form one image. Mathematically, a greyscale image is a singular 2-D array of pixel locations, each location has an intensity value. A colour image which uses the RGB (red, blue, green) colour space is represented by three 2-D arrays, where each array is the intensity values from the red, blue, and green colour channels, respectively. So every pixel of an RGB image is made up of a combination of three intensity values [152].

RGB is a commonly used colour space that is optimized for screens such as televisions and computers. The RGB colour space can be calculated from the CIE-XYZ coordinates by the transformation in equation 5.7.

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 2.36461 & -0.89654 & -0.46807 \\ -0.51517 & 1.42641 & 0.08876 \\ 0.00520 & -0.01441 & 1.00920 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{5.7}$$

Many other colour spaces were suggested between 1940 and 1976 which were based on transformations of the XYZ space presented in 1931. In 1976 the CIE presented a colour space called CIE-L*a*b* which was intended to match colours. The transformation from RGB to CIE-L*a*b* is given in equations 5.8, 5.9, and 5.10 [153]. CIE-L*a*b* gives three values; L* is the lightness measured from 0 to 100 given a defined white value, a* is where on the scale from red to green the colour is, and b* is where on the scale from blue to yellow the colour is.

$$X = 0.412453R + 0.357580G + 0.180423B \tag{5.8}$$

$$Y = 0.212671R + 0.715160G + 0.072169B \tag{5.9}$$

$$Z = 0.019334R + 0.119193G + 0.950227B \tag{5.10}$$

Based on this transformation the L*a*b* values are defined in equations 5.11, 5.12, and 5.13 where $X_n$, $Y_n$, and $Z_n$ represent the defined white and are obtained by setting $R = G = B = 100$.

$$\mathrm{L}^* = 116 f(Y/Y_n) - 16 \tag{5.11}$$

$$\text{a*} = 500[f(X/X_n) - f(Y/Y_n)] \tag{5.12}$$

$$\text{b*} = 200[f(Y/Y_n) - f(Z/Z_n)] \tag{5.13}$$

where

$$f(q) = \begin{cases} q^{1/3} & \text{if } q > 0.008856 \\ 7.787q + 16/116 & \text{otherwise.} \end{cases}$$

### 5.1.2 Greyscale methods

Colour to greyscale methods are used widely in research and industrial applications with very little thought to the impact of the methods. When processing individual images, one can choose whichever 'looks best', but, for large scale mass image processing, one should consider whether the chosen method looks best for most of the images in the set.

Typically image processing is carried out on greyscale images because of the reduced dimensions, meaning less computational requirements. There are many different linear and non-linear greyscale methods; the most popularly used of which are summed up by Kanan and Cottrell [142], and explained below. The following eight methods are the ones that were tested in this chapter.

*Intensity* is probably the simplest, being just the mean of the RGB channels, seen in equation 5.14.

$$\mathcal{G}_{Intensity} = \frac{1}{3}(R + G + B) \tag{5.14}$$

*Gleam* is the gamma corrected form of *Intensity*, seen in equation 5.15. Gamma correction is a non-linear power law which translates between a camera's light sensitivity and that of our eyes [154]. As an example of typical gamma correction of the red intensity value, $R$, $R'$ represents the gamma corrected $R$ value which is calculated by $R' = R^{1/2.2}$; several of the greyscale methods use the gamma corrected RGB values.

$$\mathcal{G}_{Gleam} = \frac{1}{3}\left(R' + G' + B'\right) \tag{5.15}$$

*Luminance* is a weighted combination of the RGB channels, seen in equation 5.16. *Luminance* is the equation used by Matlab's `rbg2gray` function [155].

$$\mathcal{G}_{Luminance} = 0.3R + 0.59G + 0.11B \tag{5.16}$$

*Luma* is a different weighted combination of the RGB channels, known as sRGB, which also features gamma correction. *Luma* is frequently used as the common standard on the internet and in most screens [156], and can be seen in equation 5.17.

$$\mathcal{G}_{Luma} = 0.2126R' + 0.7152G' + 0.0722B' \tag{5.17}$$

*Lightness* is the intensity part of the CIE-L*a*b* colour space and is calculated in equation 5.18 [153]. In the CIE-L*a*b* colour space the intensity values are in the range 0 to 100, so *Lightness* is normalised to fit the $[0, 1]$ range.

$$\mathcal{G}_{Lightness} = \frac{1}{100}(116f(K) - 16) \tag{5.18}$$

where $K = 0.2126R + 0.7152G + 0.0722B$, and

$$f(K) = \begin{cases} K^{1/3} & \text{if } K > (6/29)^3 \\ \frac{1}{3}\left(\frac{29}{6}\right)^2 K + \frac{4}{29} & \text{otherwise.} \end{cases} \tag{5.19}$$

As an example, to calculate Lightness for colour $RGB = [0.1804, 0.2078, 0.2471]$, let $K = 0.0.2126R + 0.7152G + 0.0722B \implies K = 0.20481222$ .
Since $K > \left(\frac{6}{29}\right)^{1/3} \implies f(K) = K^{1/3}$ from equation 5.19, the greyscale value is calculated as

$$\mathcal{G}_{Lightness} = \frac{1}{100}\left(116f(K) - 16\right) = 0.5237698453 \tag{5.20}$$

*Value* is the absolute brightness information from the HSV colour space and is calculated using the maximum value from the RGB channels, as seen in equation 5.21.

$$\mathcal{G}_{Value} = max(R, G, B) \tag{5.21}$$

*Luster* is actually the L channel from the HLS colour space, but we will keep the name change from Kanan and Cottrell's paper [142] to avoid confusion of multiple methods with the same name. *Luster* is calculated by taking the maximum and minimum RGB values and computing the mean values, as seen in equation 5.22.

$$\mathcal{G}_{Luster} = \frac{1}{2}\left(max(R, G, B) + min(R, G, B)\right) \tag{5.22}$$

*Decolorize* is an algorithm designed to enhance contrast in colour to greyscale transformation [142]. Decolorize converts the image to the YPQ colour space then samples the colour differences using Gaussian pairing, then combines luminance values with the predominant axis of these differences, and uses the saturation to calculate the dynamic range of the final image [157]. Figure 5.5 shows three images put through all of the greyscale methods: one colour chart consisting of red, blue, green, cyan, fuschia, yellow, and a scale from blue to cyan; an image of some flowers; and an image of some scenery in the Welsh valleys.

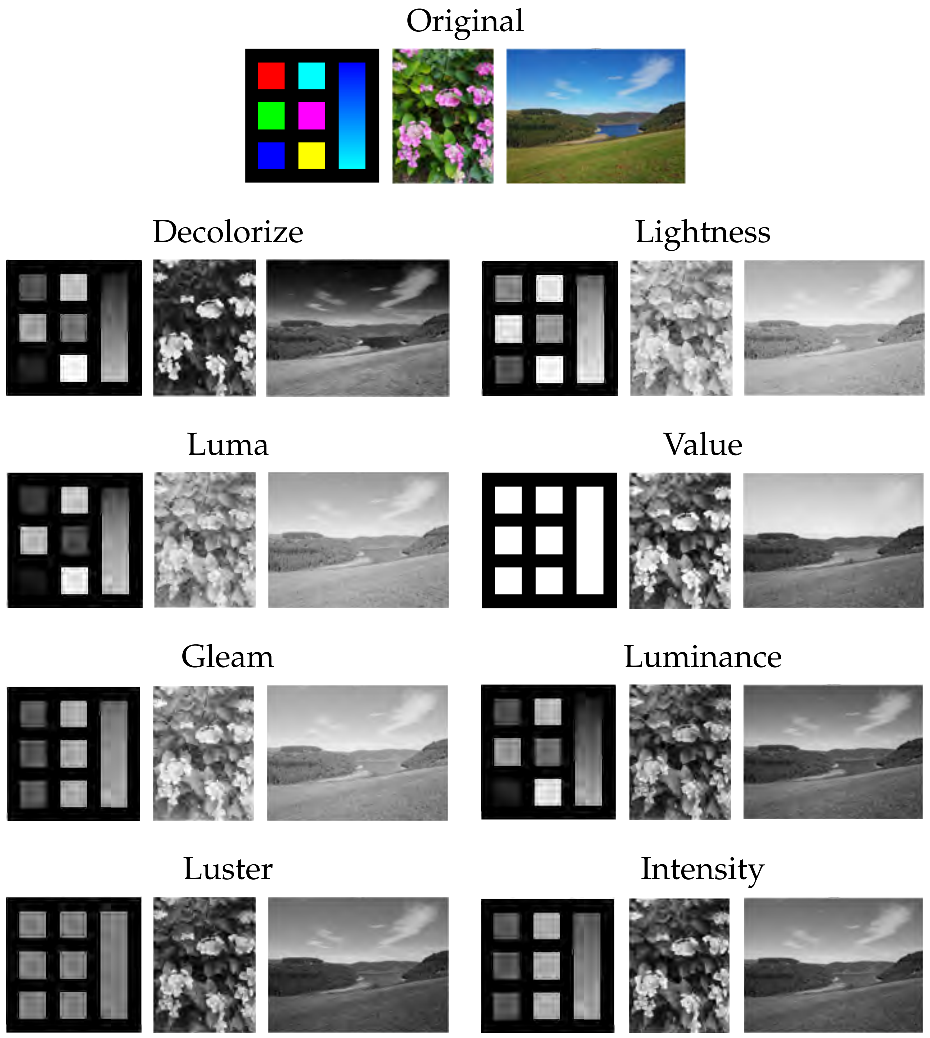


FIGURE 5.5: Three example images to show how the different greyscale methods have an effect on image contrast.

### 5.1.3 Filters

Initially four filters were tested. These were primary red, primary blue, primary green, and yellow. Initial images taken suggested that some of these would not let enough light through, leading to the testing of a light blue, a light green, a pink and an orange filter. The data sheets provided with the filters give the RGB values, a graph of the light transmission across the wavelengths and a Y% transmission value which is the percent of visible light the filter transmits. The higher the Y percentage, the more light passes through the filter and the less saturated the colour effect is. Figure 5.6 shows the colours the filters produced when scanned with nothing between the filter and the scanner lid.

The filters purchased were either made by Rosco Laboratories Inc. [158] or LEE Filters [159] depending on availability. Both manufacturers adhere to the European standard for filter numbering, meaning that a filter of the same filter number from either brand should let the same amount of light through. Whilst that testing was not carried out, the manufacturer's data sheets from both brands were compared for all of the filters used and they do appear to match.

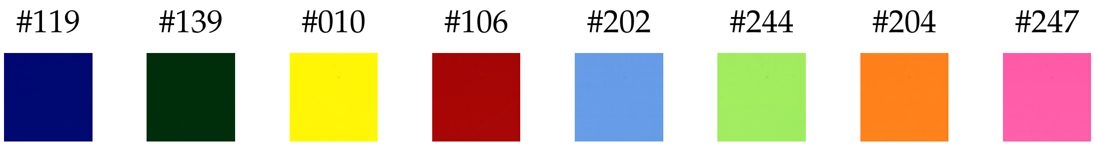


FIGURE 5.6: The colour of the filters when scanned alone.

**Primary blue filter** The first blue filter tested was called #119 Dark Blue by Rosco E-Colour+ as initial investigation suggested that this blue colour filter was the one included in filter packs providing the primary colours. Its RGB values from the data sheet were [0 101 173] and its Y transmission was about 3% which, in hindsight, was far too low. Figure 5.7 shows the transmission of light over the visible wavelengths.

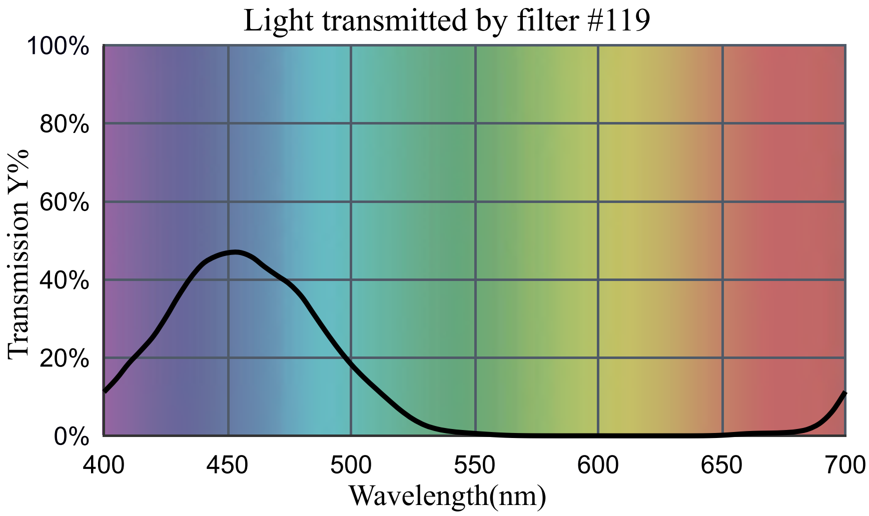


FIGURE 5.7: The transmission of primary blue filter #119.

**Primary green filter** The first green filter tested was called #139 Primary Green by Rosco E-Colour+. Its RGB values were [0 111 61] and its Y transmission was 10%, which was also considered too low. Figure 5.8 shows the transmission of light over the visible wavelengths.

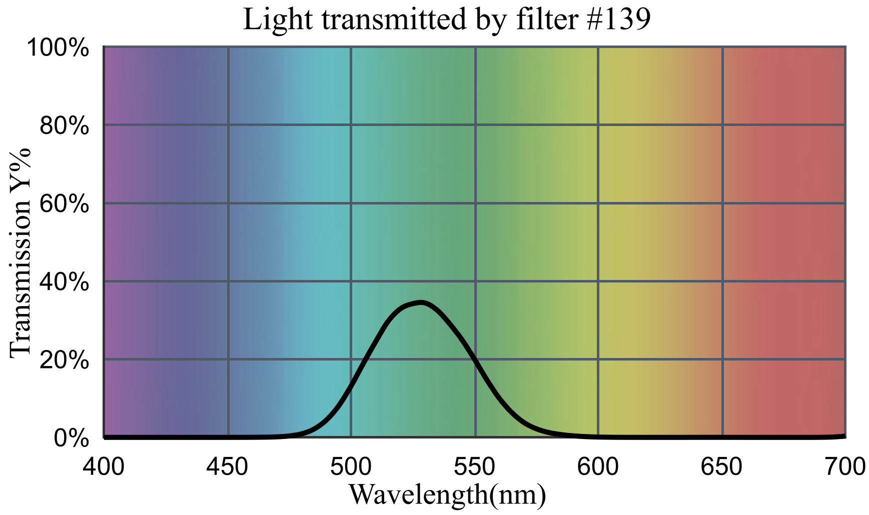


FIGURE 5.8: The transmission of primary green filter #139.

**Primary red filter** The red filter tested was called #106 Primary Red by Rosco E-Colour+. Its RGB values were [207 41 0] and its Y transmission was 15%. This filter is quite dark, but not as dark ad the primary blue or primary green. Figure 5.9 shows the transmission of light over the visible wavelengths.

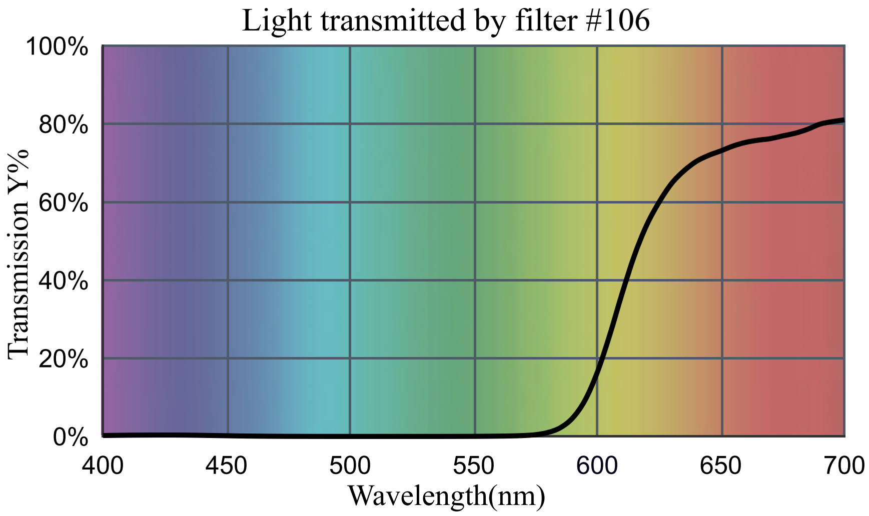


FIGURE 5.9: The transmission of primary red filter #106.

**Yellow filter** The yellow filter tested was the #010 Medium Yellow by Rosco E-Colour+. Its RGB values were [225 221 0] and its Y transmission was 85%. This filter let through far more light and allowed more of the defect to be visible. The results of this filter led to the purchase of the other coloured filters with far higher Y% values. Figure 5.10 shows the transmission of light over the visible wavelengths.

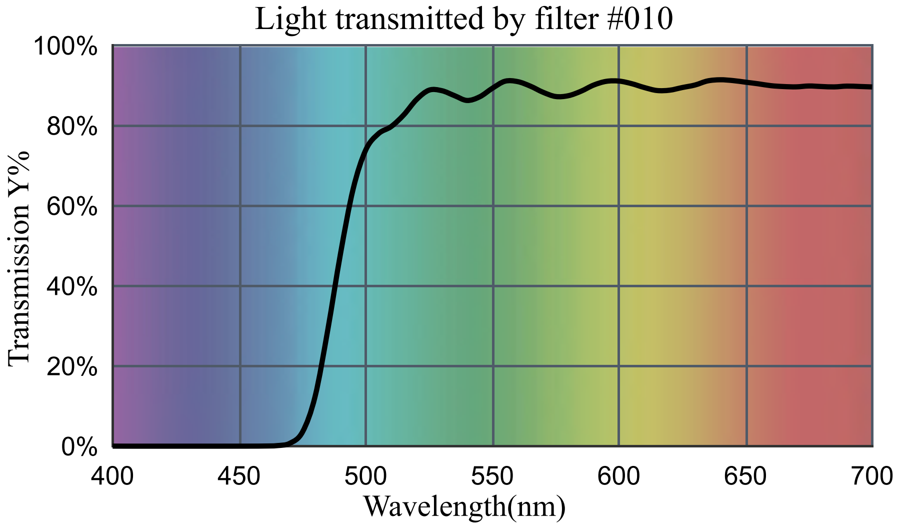


FIGURE 5.10: The transmission of medium yellow filter #010.

**Light blue filter** The light blue filter tested was the #202 Half C.T. Blue by Rosco E-Colour+. Its RGB values were [133 186 228] and its Y transmission was 45%. This filter makes images look more blue and less yellow. Figure 5.11 shows the transmission of light over the visible wavelengths.

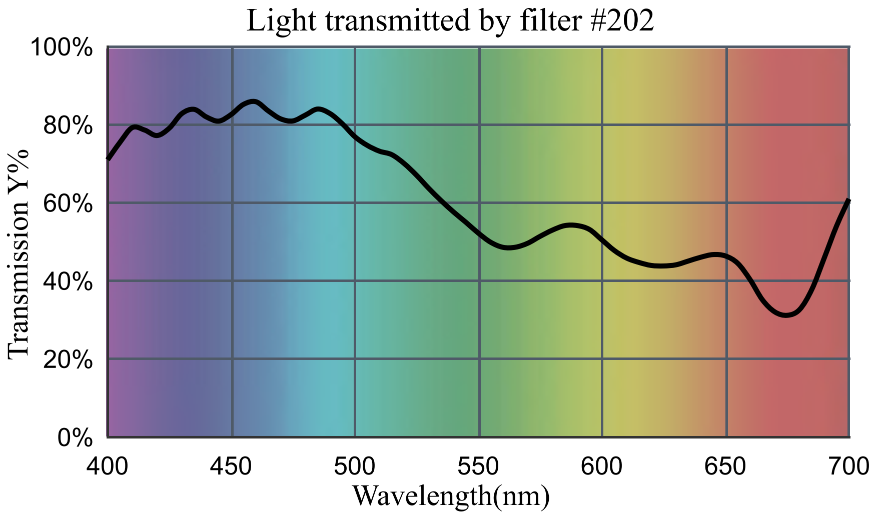


FIGURE 5.11: The transmission of light blue filter #202.

**Light green filter** The light green filter tested was the #244 Plus Green by LEE Filters. The RGB values were not provided on the manufacturer's data sheet. Its Y transmission was 73%. This filter is supposed to enhance green shades. Figure 5.12 shows the transmission of light over the visible wavelengths.

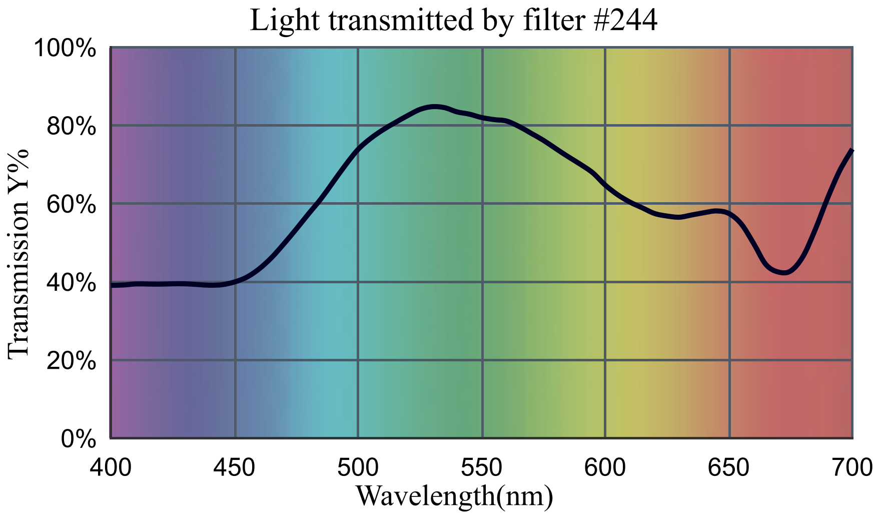


FIGURE 5.12: The transmission of light green filter #244.

**Orange filter** The orange filter tested was the #204 Full C.T. Orange by Rosco E-Colour+. Its RGB values were [251 183 114] and its Y transmission was 58%. This filter makes images look less blue and more orange. Figure 5.13 shows the transmission of light over the visible wavelengths.

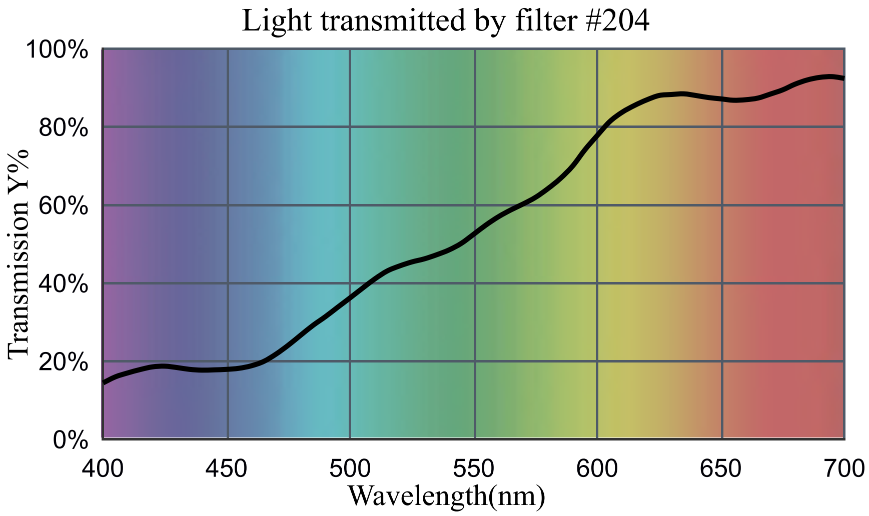


FIGURE 5.13: The transmission of orange filter #204.

**Pink filter** The pink filter tested was the #247 Minus Green by LEE Filters. Its RGB values were not provided on the manufacturer's data sheet. Its Y transmission was 59%. This filter removes green shades from images. Figure 5.14 shows the transmission of light over the visible wavelengths.

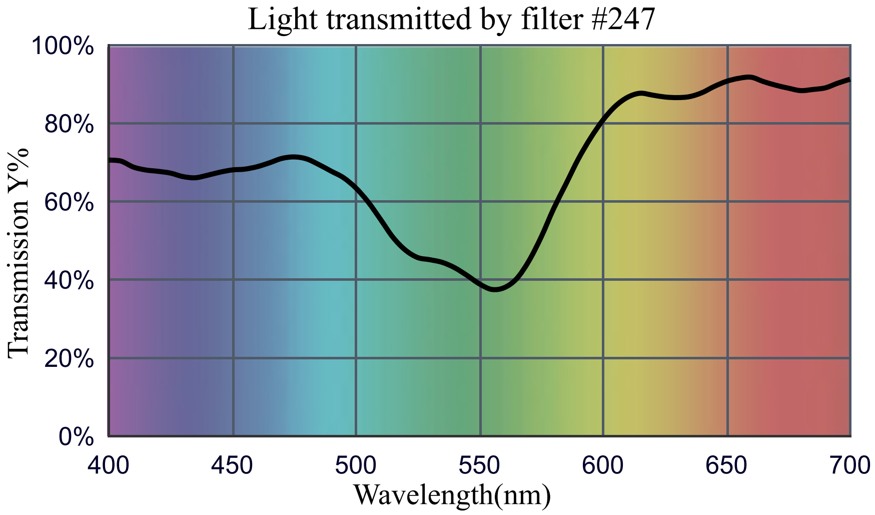
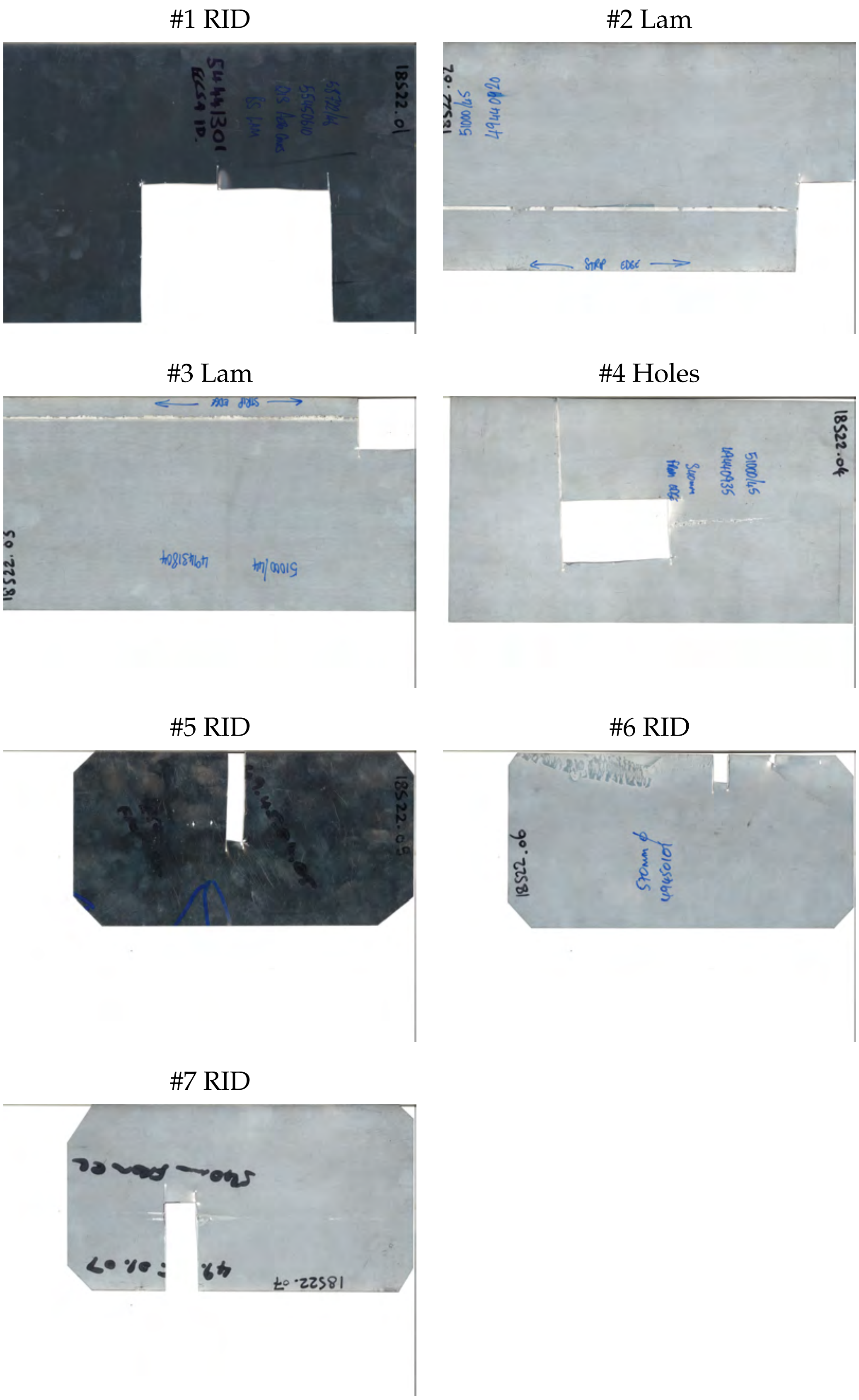


FIGURE 5.14: The transmission of pink filter #247.

## 5.2 Samples

There were seven samples provided, of three different defect types. Images of these samples taken in the scanner can be seen in Figure 5.15. The scanner images are A4 size - 210mm x 297mm and have resolution 7015 pixels wide and 4960 pixels high.

A scanner was decided to be the most consistent method of photographing the samples in this chapter as a flatbed scanner uses consistent lighting and angles and eliminates many possible points of human error. Discussions with my industrial sponsors came to the conclusion that, whilst their staff had tried various methods of photographing their cut-out defect samples over the years, they have found that a flatbed scanner was the method which produced the most consistent and appropriate quality images which are similar to those images taken on the production line.



#1 RID

#2 Lam

#3 Lam

#4 Holes

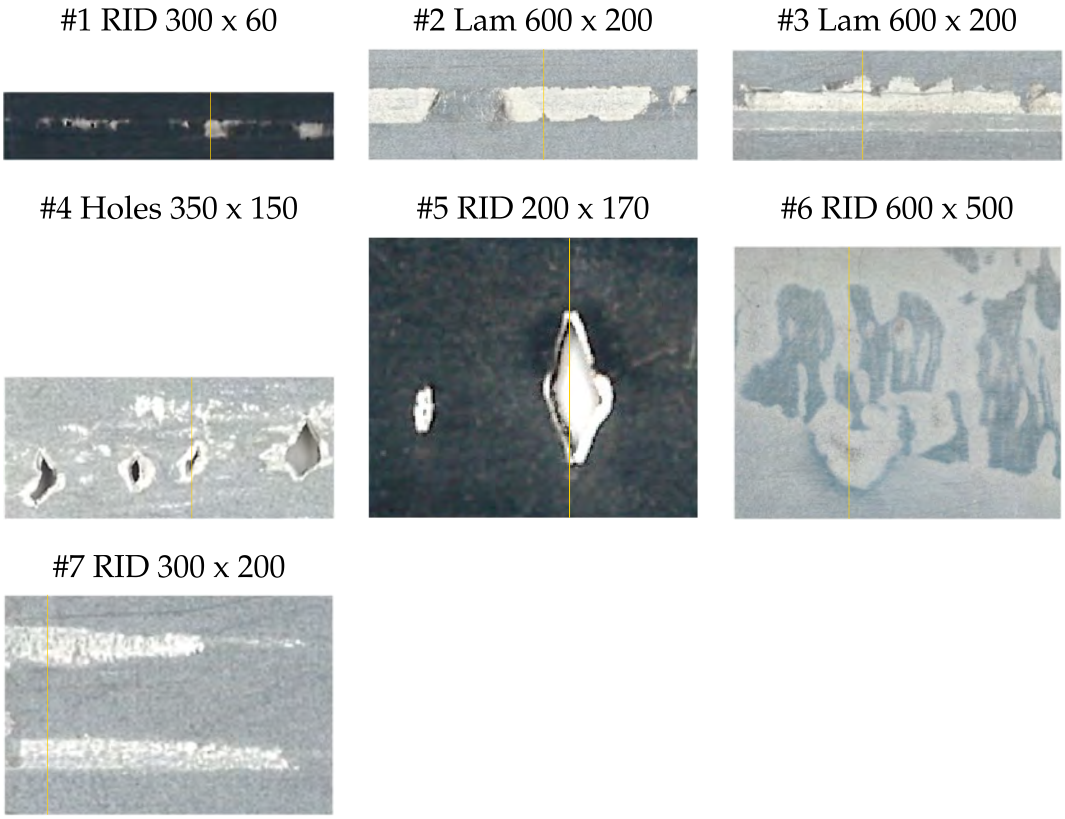
#5 RID

#6 RID

#7 RID

FIGURE 5.15: Full images of samples, the images are A4 size - 297mm x 210mm represented by 7015 x 4960 pixels.

The regions of interest are shown in Figure 5.16. These were selected based on their ability to be reproduced easily by aligning certain defect points at certain locations, ensuring that each new image taken of the samples would be analysed in the same place.



#1 RID 300 x 60

#2 Lam 600 x 200

#3 Lam 600 x 200

#4 Holes 350 x 150

#5 RID 200 x 170

#6 RID 600 x 500

#7 RID 300 x 200

FIGURE 5.16: Sample ROIs where the yellow lines indicate where the range has been calculated on all images of this sample. The size of each sample is listed in pixels relative to the A4 images in Figure 5.15.

A particular edge of the defect is lined up to be at specific coordinates $(x, y)$ and the ROI, the column on which the yellow line is, is then selected based on that point. The same column of each image is then selected to measure the maximum and minimum intensity values to generate the range of that column. These ranges are used as a measure to determine if contrast is better or worse using different greyscale methods and different colour filters. Columns were selected based on having a good range of defect and non-defect areas and their ease of lining up to ensure the same areas are being measured in each sample.

## 5.3 Methods

### 5.3.1 Part 1

A palette of the colours of steel was created by using a colour identifier tool on random defect and non-defect areas within the scanned images in Figure 5.16. The contrast between non-defect and defect colours was calculated by using the sum of the ranges calculated for each combination of defect and non-defect colours. This was calculated in both the RGB colour space and after applying the eight greyscaling methods described in section 5.1.2. The colour palette can be seen in Figure 5.17 with the defect colours having a red 'D' label and the non-defect colours having a green 'ND' label.

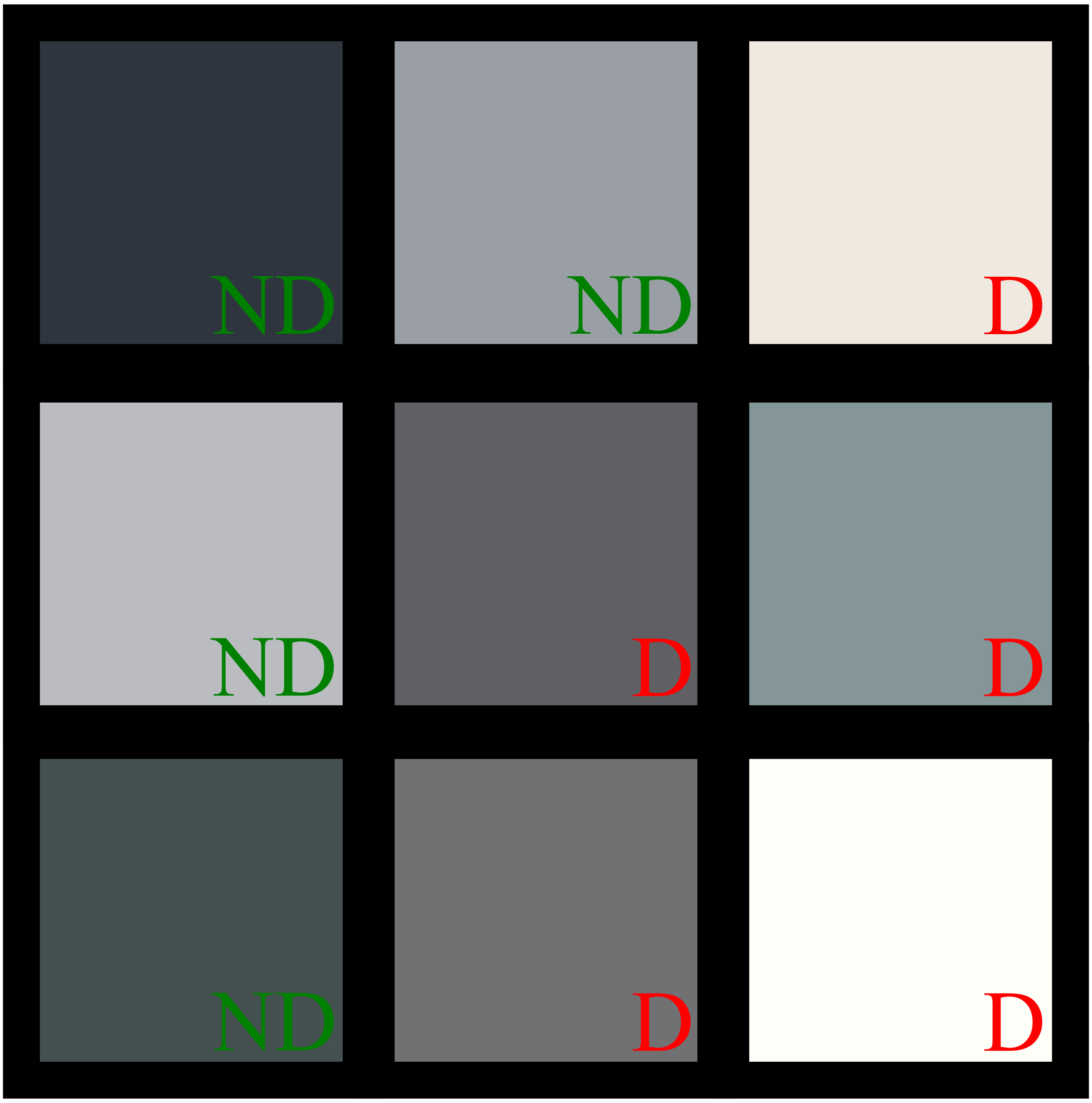


FIGURE 5.17: Steel colour palette with defect (D) and non-defect (ND) colours.

The same greyscaling methods were then applied to colour images of defects taken with white light and analysed by calculating their range. Figure 5.18 shows the values for the ROI column of sample 2 photographed under white light. For example, the blue and yellow arrows highlight the range of values taken for the Decolorize and Lightness methods as a measure of contrast. The aim of calculating the contrasts between defect and non-defect colours and images is that the higher the contrast between defect and non-defect regions, the more effective the image will be when used for training a classifier for an inspection system.

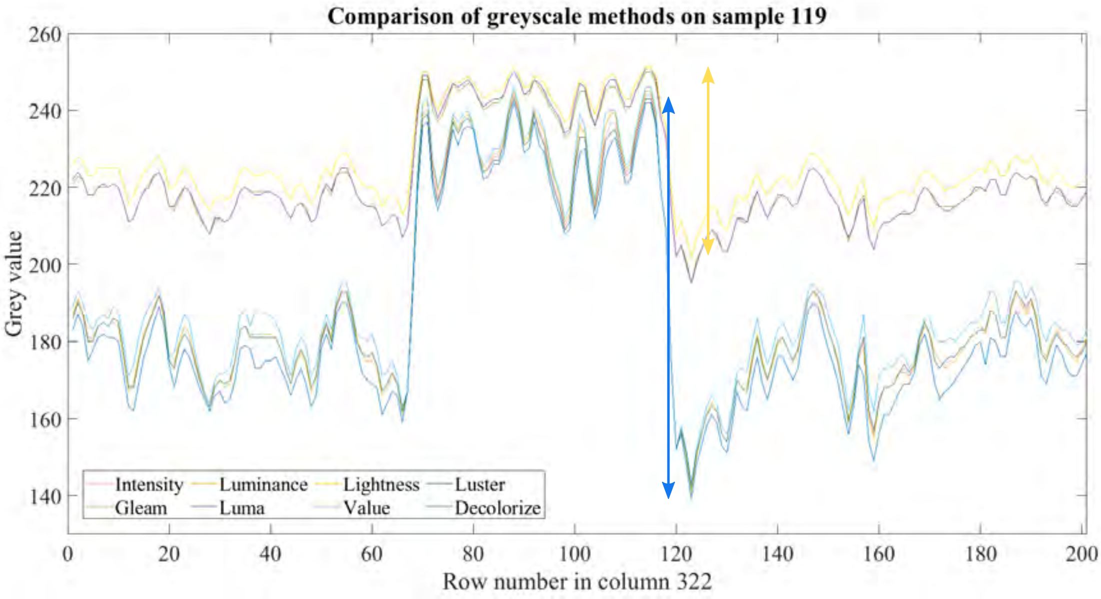


FIGURE 5.18: Comparison of greyscale methods for sample 119 which is physical sample number 2, a lamination, with the image taken in full colour with no filter.

The sum of ranges for the colour palette was calculated by working out the difference between each of the defect colours and each of the non-defect colours and summing their absolute values, to give a measure of total contrast. Equation 5.23 shows an example of the equation between four colours, where $g_1$ and $g_2$ are the intensity values of the two defect colours, and $g_3$ and $g_4$ are the intensity values of the two non-defect colours.

$$\text{Sum of palette ranges} = |g_1 - g_3| + |g_1 - g_4| + |g_2 - g_3| + |g_2 - g_4| \tag{5.23}$$

To calculate the sum of ranges for the samples, the minimum and maximum values of the ROI column of the sample image were calculated and subtracted as in equation 5.24. Here $\text{ROI}_1$ represents the ROI column for sample 1 and $n$ is the total number of samples.

$$\text{Sum} = (max(\text{ROI}_1) - min(\text{ROI}_1)) + \ldots + (max(\text{ROI}_n) - min(\text{ROI}_n)) \quad (5.24)$$

The samples were photographed using a flatbed scanner, the set up is seen in Figure 5.19 where colour filters were only used in method Q2. The scanner glass was cleaned of dust between each scan, to limit the amount of dust visible on the scans which improves the quality of the images. The samples were placed straight onto the scanner glass and images were produced in '.jpg' format at the highest resolution possible of 600x600 dpi (dots per inch). There were no manual controls for lighting, so this remained consistent throughout.

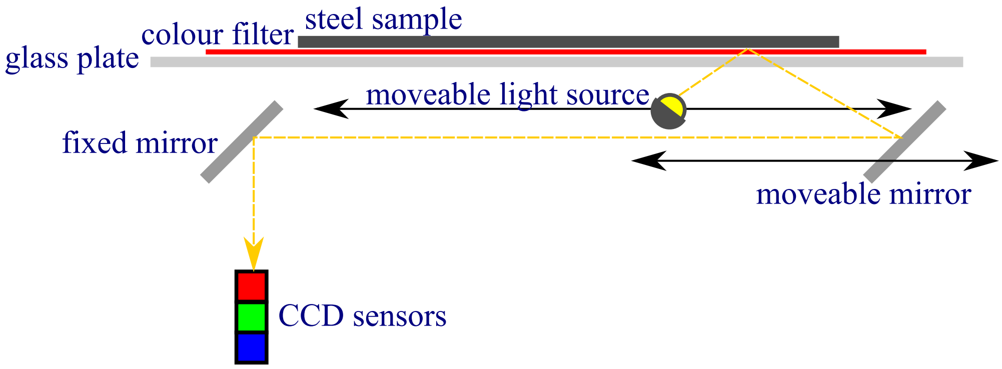


FIGURE 5.19: The flatbed scanner setup used in Methods Q1 (without filter) and Q2 (with filter).

Not only were the ranges calculated to figure out which greyscale method produces the highest contrast, but the images were also put through a binary decision tree classification to determine which greyscale method had the highest performance. This uses the function `fitctree` in Matlab and trains a singular decision tree. A singular decision tree was used instead of a random forest, because when initially tested with a random forest classifier growing 100, 50, and then 5 trees, all results were 100% accurate until only 5 trees were grown, which does not assist us in determining the differences in performance that are possible when using different greyscale methods. The data was input in 1-D form, with defect and non-defect slices of all 7 samples and an equal number of each. The lengths of the slices were normalised to 200 pixels in length for uniformity. In total, 25 positive and 25 negative slices were taken from each sample, making a 350x200 data set. A random subset of 50 were withheld from training and used in testing the performances. These performance results were measured by calculating the overall accuracy and false negative rate, as in a confusion matrix, and the K-Fold loss. Similar to the 'OOBerror', the K-Fold loss is a measure of loss calculated by splitting the data up into 10 folds and training the model on 'out-of-fold' observations then computing the loss for 'in-fold' predictions [160].

### 5.3.2 Part 2

Question two investigates what impact colour has on contrast and performance. First each sample was photographed using a scanner, as described in Figure 5.19. The colour filters were placed between the scanner glass and the sample. Different colour filters were tested to determine whether the colour of the filter has an effect on contrast and classification performance. These colour images were then subject to the same greyscaling methods. The same ROIs were used on the colour filtered images as with the full colour images. The same sections of images were used as training and test data in 1-D decision tree binary classification for each colour filter. The colour filter put through its optimal filter was used to train and test the classifier to determine performance affects of colour filters.

### 5.3.3 Part 3

Question 4 considers the impact of green filters on detecting the presence of oil. A machine oil called Shell Morlina S1B230 was obtained from Tata Steel Europe in Port Talbot. It is used on site for lubricating bearings and moving parts so is an example of an oil that could be encountered on the line. Further old steel samples were provided and three different steel samples were selected and the oil was dropped onto the surface of the steel with a pipette and smeared over the surface with a sponge or the end of the pipette. The defect type of these samples was irrelevant in this case. Oil marks on steel are not consistent in shape or thickness, so it made sense to add an element of randomness to the oil spread on the samples. These samples were then photographed in a similar manner to the previous images, using a flatbed scanner, except the samples were propped up with a cardboard frame to stop the oil contacting the scanner glass or the colour filter. The configuration can be seen in Figure 5.20.

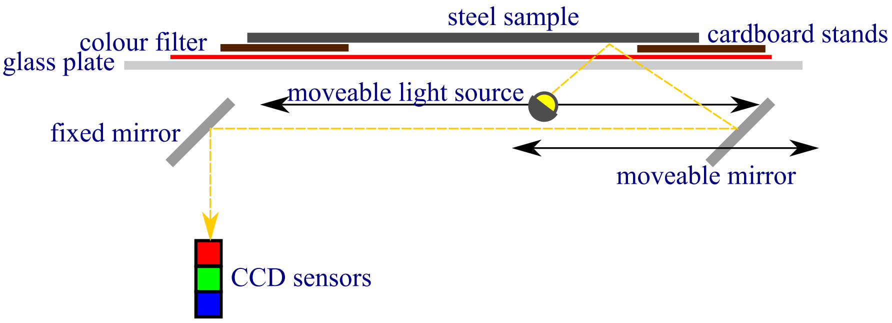


FIGURE 5.20: The flatbed scanner setup for photographing oiled samples.

A region of interest was then identified for each sample which involved a crossover from clean steel onto an oil-covered portion. These images were put through the greyscale methods as in parts 1 and 2 of this section. In the ROIs the grey values should increase over the oil. The intention is to determine which method has the largest increase between these grey values of clean steel and the grey values of oil on steel. This is measured by calculating the range of the grey values in the ROIs, an example of an ROI can be seen in Figure 5.21. One could use the mean of the grey values of the oil and then the mean of the grey values of the clean steel, from the ROI, however the issue with this method is that which pixel contains the very edge of the oil can be difficult to determine with adequate accuracy, whereas the maximum and minimum grey values consistently fall on either the oil or clean side of the sample, but not on the same side.

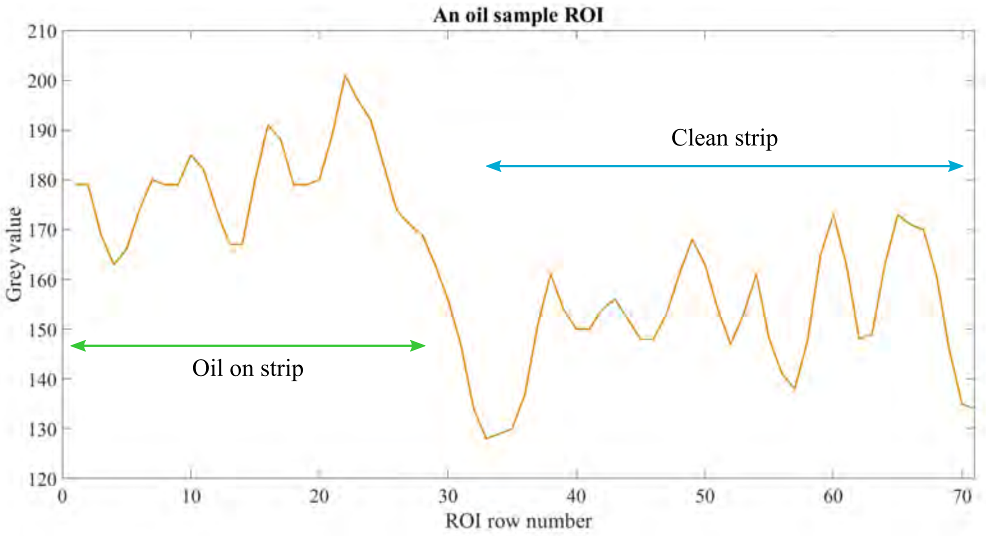


FIGURE 5.21: An example of an oil sample's region of interest.

## 5.4 Results Analysis

### 5.4.1 Part 1

Figure 5.22 shows the results of applying the different greyscale methods to the steel colour palette. Greyscale methods such as Lightness, Luma, and Gleam have lost a lot of their colour information compared to the original. Visually there is little difference between the original and Decolorize, Luster, Value, Luminance, and Intensity.

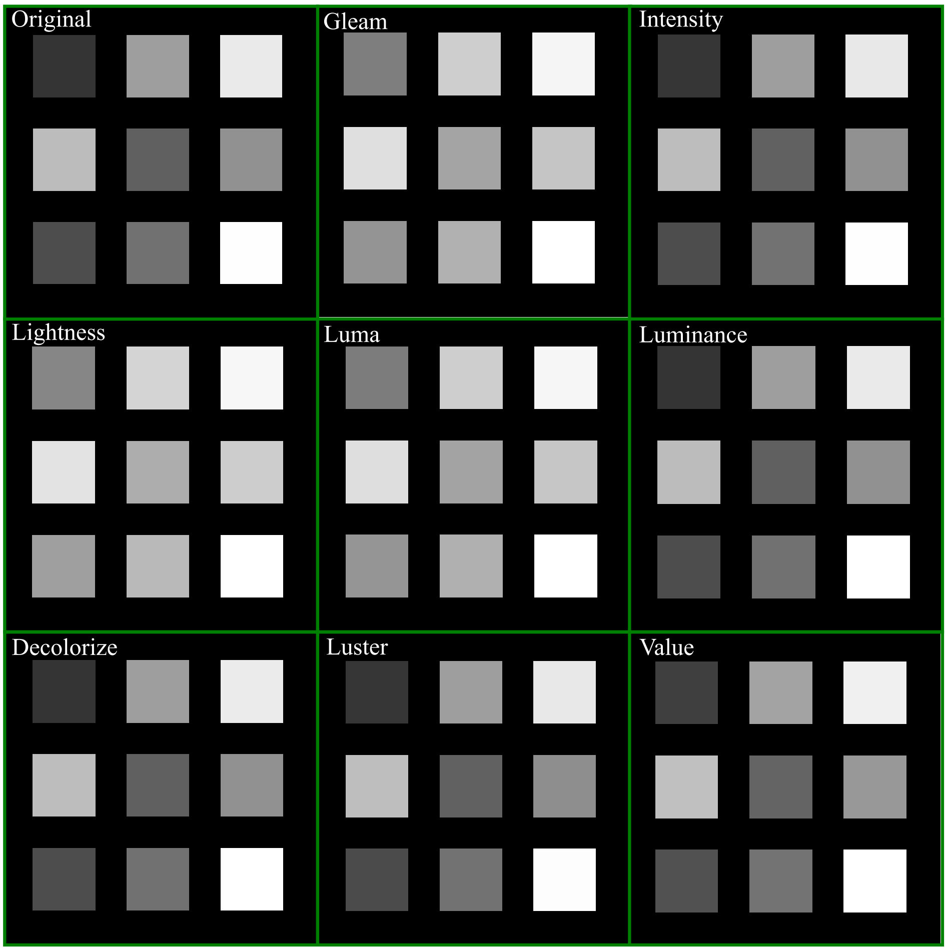
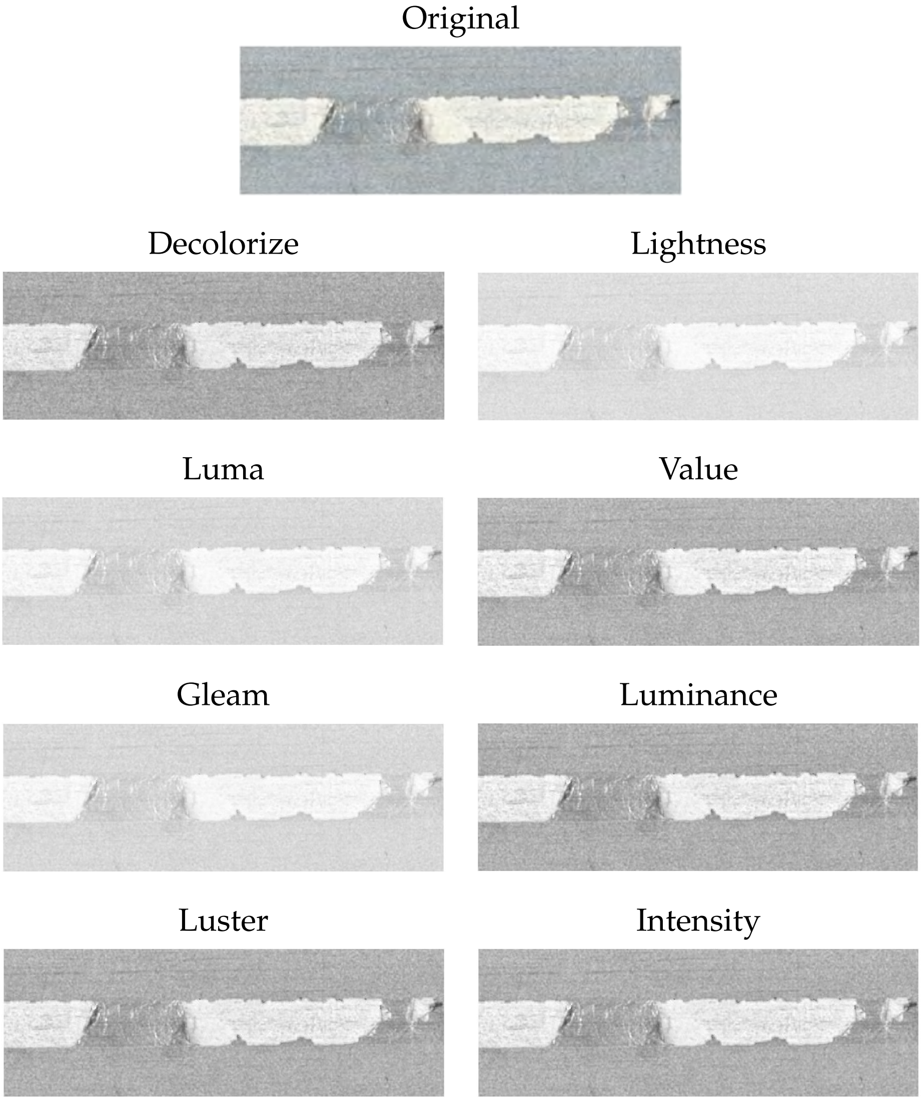


FIGURE 5.22: Greyscaling results for the steel colour palette.

Figure 5.23 shows the different greyscale results for sample 2. The ROI column is column 322 and the range is calculated for each of these images and for the rest of the samples. The ranges are then grouped by greyscale method and summed. Measuring the contrasts numerically is a more appropriate measure than just visually analysing the colours as the differences in these shades are very subtle and difficult to distinguish for the human eye. Like with the colour palette images, Lightness, Luma, and Gleam have lost some of the colour information, but the other methods have produced images fairly similar to the original.



Original

Decolorize

Lightness

Luma

Value

Gleam

Luminance

Luster

Intensity

FIGURE 5.23: Colour image of sample 2 put through the greyscale methods.

Figure 5.24 shows the results of the sum of ranges for each of the greyscale methods with the data categories being the colour palette (seen in Figure 5.22), the colour images (an example of which is in Figure 5.23) and then broken down by the defect type of the samples to RIDs, Laminations, and Holes.

Decolorize is the best method with the highest sum of ranges at about 6.8 for the colour palette and 4.1 for the samples, meaning it produces the highest contrast of all of the methods. Luminance, Luster, Intensity, and Value all have a similar results for each data category; the sum of ranges for the colour palette is around 6.5 and around 3.9 for the samples. Luma and Gleam perform similarly, but considerably worse than the previous methods; the sum of ranges is around 4.2 for the colour palette and just 2.7 for the samples. Lightness is the worst performer, with a sum of ranges of 3.8 for the colour palette and 3.6 for the samples.

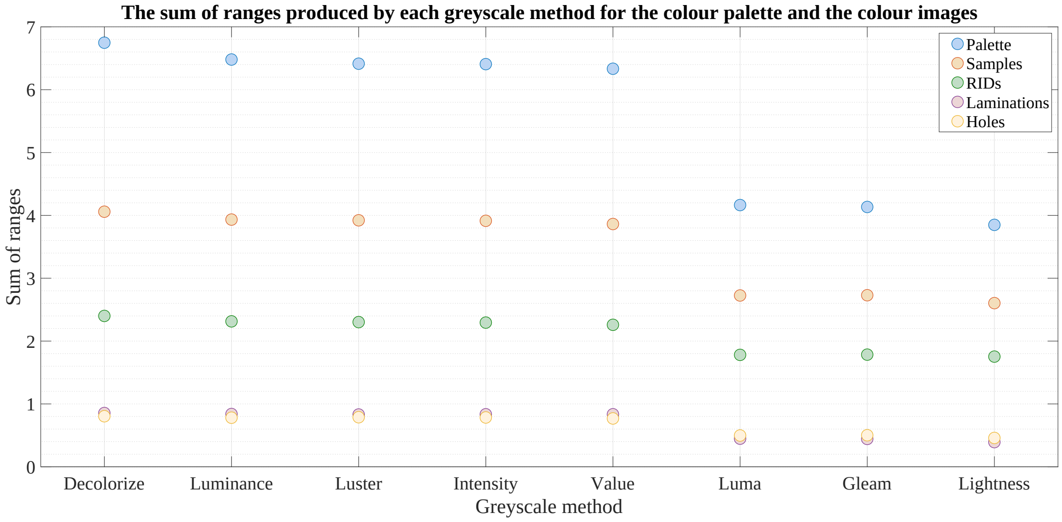


FIGURE 5.24: The sum of ranges for each greyscale method applied to the colour palette and the unfiltered colour images of the samples.

The performances remain consistent for the different defect types. However for Holes there is the least variation between methods. This could just be down to the quality of the only Holes sample supplied though, and so further testing with a large amount of different defect types should be carried out before claims based on defect type can be hypothesised.

Each of the greyscale methods have similar levels of performance relative to each of the data categories; the colour palette, all of the samples, and even when split up into defect type. The actual value of the sum of ranges is not an important factor, what this graph highlights is that some greyscale methods consistently produce higher contrast images than others. This helps in identifying which greyscale method is the most suitable to use.

Figure 5.25 shows the performance results from the decision tree predictions. The aim with this graph is to identify which method(s) are consistently low in all three measures; K-Fold loss, overall error (which is 100 - the overall accuracy) and the false negative rate (FNR). Some of the methods register a 0% error meaning 100% accuracy, this does not mean the methods will consistently always be perfect when exposed to further new and broader different sets of data, but it is an indicator of more accurate performance than methods with high error rates on this small data set.

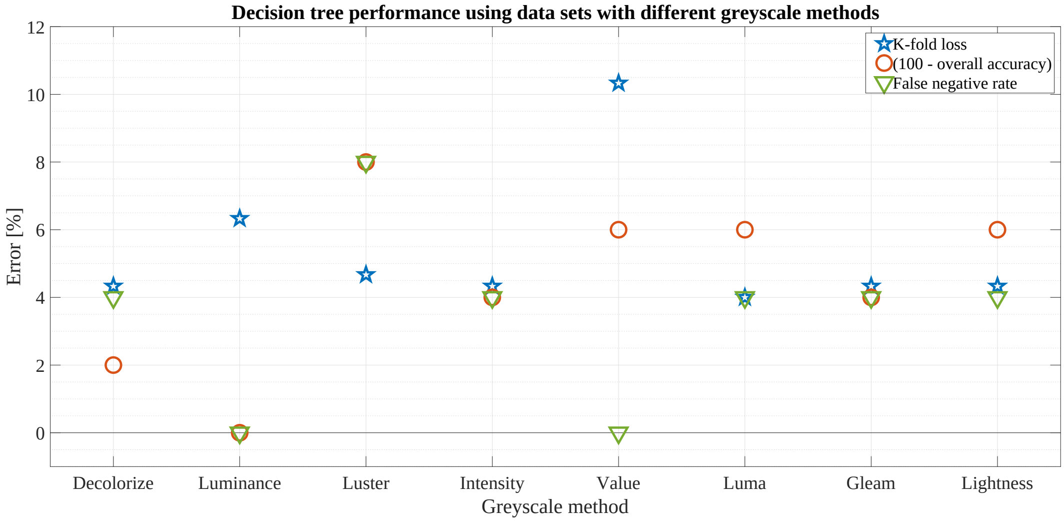


FIGURE 5.25: Performance results for a decision tree trained on data sets with different greyscale methods.

Decolorize performs very well in all three measures, with an overall error of 2% and a K-fold loss and FNR of around 4%. Luminance has very spread results, with an overall error and FNR of 0%, but a high k-fold loss of 6%. These high K-fold loss results indicate that there may be issues when scaling up this method to larger, more varied data sets. Luster has very high results with a K-fold loss of 4.7% and the overall error and FNR are around 8%, which is very high. Intensity and Gleam have consistent results with all measures around 4-4.5%, but these values would ideally be closer to 0%. Value has the most spread of results, with a K-fold loss of 10.3%, an overall error of 6%, and an FNR of 0%. Value could also be another method which may have issues when scaling up. Luma and Lightness have similar results, with a 6% overall error, and the

K-fold loss and FNR around 4-4.5%. Decolorize, Intensity, Luma, Gleam, and Lightness all perform fairly similarly and consistently through all measures, but Decolorize has a lower overall error than the others, giving it a slightly better performance overall.

For both contrast and performance, Decolorize is the method which performs the best overall. It produces consistently high contrasts for both the colour palette and the full colour sample images. It has the best overall performance in 1-D prediction of defects.

### 5.4.2 Part 2

Figure 5.26 shows the colour filter results for sample 2; similar results were produced for all seven samples and the full image results are in Appendix C. It is clear that certain colour filters (dark blue, dark green, and possibly red) do not let enough light through to get an adequate image. However the rest show the defects clearly, but there are subtle differences in the shape and contrast of the defect under different filters.

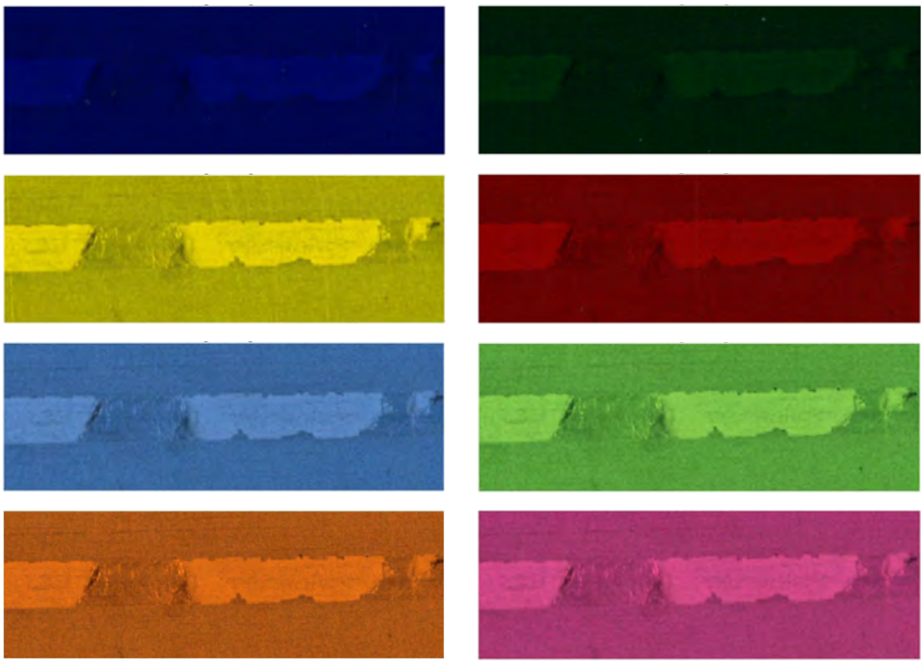


FIGURE 5.26: Sample number 2 photographed using the different colour filters.

Figure 5.27 shows the orange filtered sample 2 put through the different greyscale methods. All colour filtered images were put through the eight greyscale methods in a similar fashion to produce the results in Figure 5.28. Some methods, such as Lightness, Luma, Value, and Gleam appear to produce quite bright images, but the contrast between the defect and clean steel is lacking in all except the Value image. The resulting images from Luminance, Luster, and Intensity lack brightness, but still maintain some adequate contrast between the defect and the clean steel. Decolorize appears to both darken the clean steel and brighten the defect, making the defect really stand out.

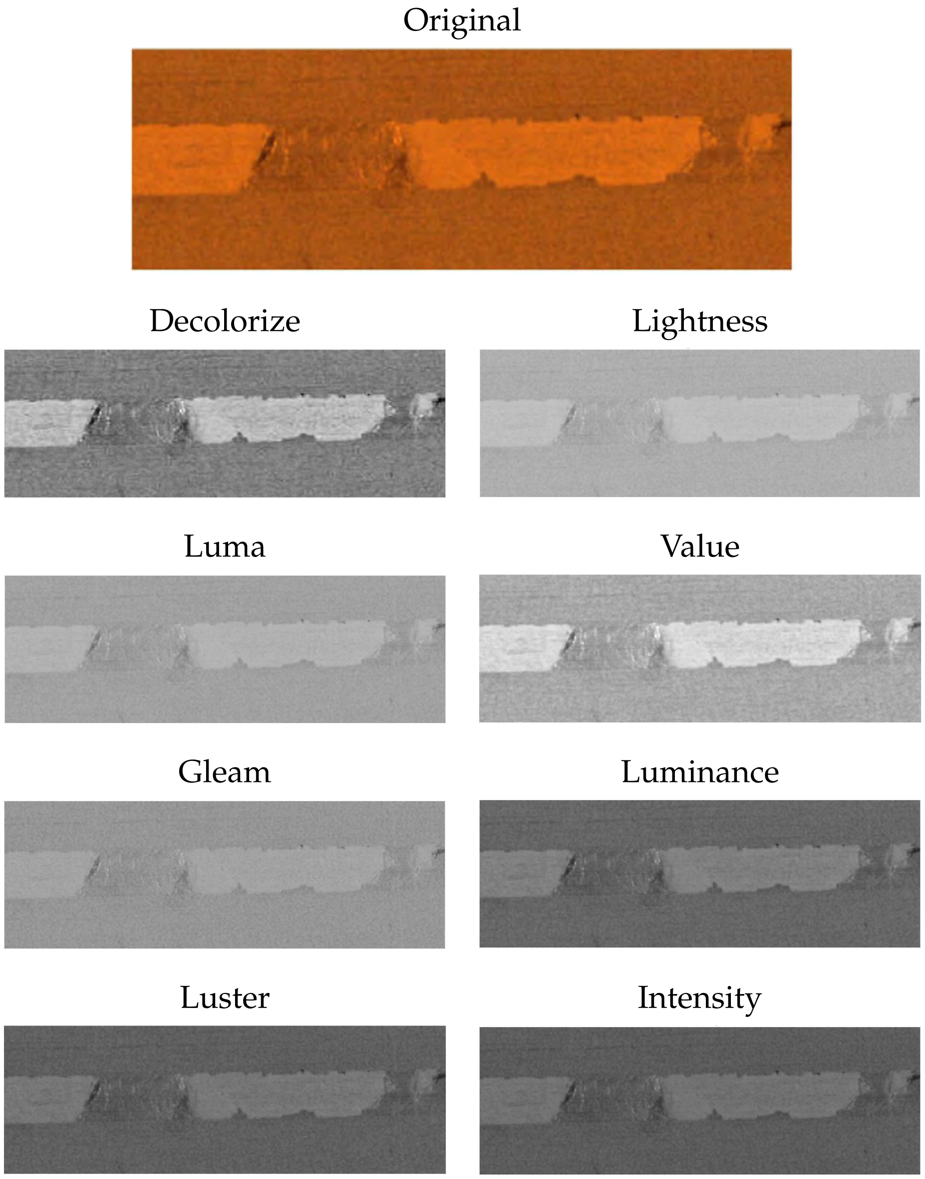


FIGURE 5.27: Sample number 2 orange filtered image put through each of the greyscale methods.

Figure 5.28 and Table 5.1 show the sum of grey value ranges of each greyscale method using each colour filter. Each circle represents the summed ranges of the ROI from each of the seven samples which have been photographed using the same colour filter and put through the same greyscale method. The different colour circles represent different greyscale methods and each column represents a different colour filter. In the following analysis where the term 'contrast' is used, it is referring to the sum of the range values; that is the summed differences between the maximum and minimum grey values of each sample. The importance in this graph is not the actual value of the contrast as there is not a known value to aim for, but the contrast values for each filter compared to the rest of the filters. A higher contrast value indicates a filter which provides more defect information in the image, so higher is better.

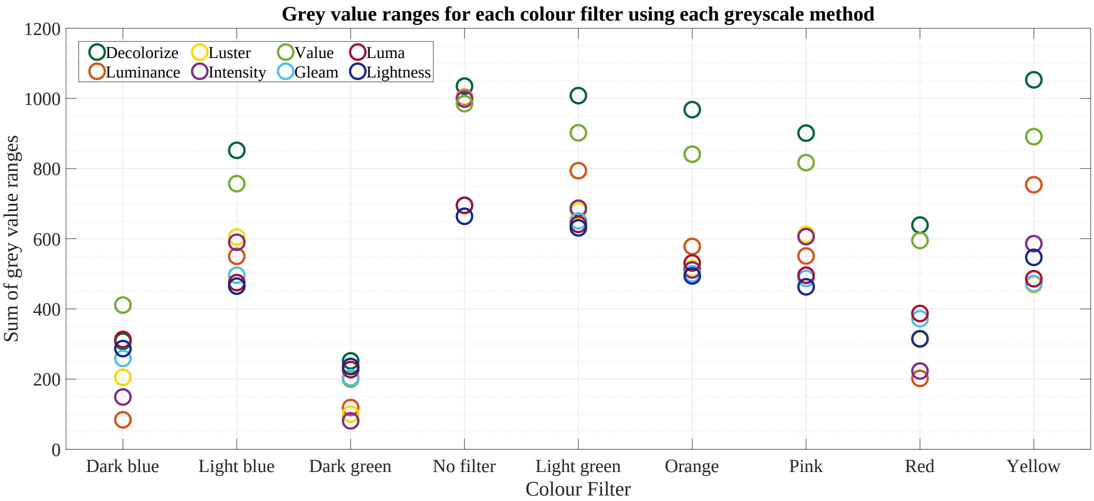


FIGURE 5.28: The sum of the grey value ranges for each colour filter using each greyscale method.

The dark blue filter and the dark green filter (a.k.a. Primary Blue #119 and Primary Green #139) produced very low contrast images, regardless of which greyscale method was used. The minimum contrast produced out of all of the coloured filters was 81; this was achieved by the dark green filter with the Intensity greyscale method applied. The dark blue filter shortly followed with a contrast of 84 with the Luminance method applied. The highest contrasts achieved by the dark blue and dark green filters were 411 and 252, respectively, which were both lower than the lowest contrasts of every other colour filter except red.

TABLE 5.1: The sum of the grey value ranges for each colour filter using each greyscale method.

| | **Dark blue** | **Light blue** | **Dark green** | **No filter** | **Light green** | **Orange** | **Pink** | **Red** | **Yellow** |
|---|---|---|---|---|---|---|---|---|---|
| **Decolorize** | 308 | 852 | 252 | 1035 | 1008 | 968 | 901 | 639 | 1053 |
| **Luminance** | 84 | 550 | 119 | 1003 | 794 | 578 | 551 | 202 | 754 |
| **Luster** | 205 | 605 | 100 | 1000 | 680 | 515 | 612 | 313 | 469 |
| **Intensity** | 149 | 590 | 81 | 998 | 687 | 511 | 606 | 223 | 586 |
| **Value** | 411 | 757 | 200 | 985 | 902 | 841 | 817 | 595 | 891 |
| **Gleam** | 259 | 496 | 204 | 696 | 651 | 500 | 487 | 372 | 472 |
| **Luma** | 313 | 475 | 227 | 695 | 641 | 531 | 496 | 387 | 486 |
| **Lightness** | 287 | 465 | 236 | 664 | 631 | 494 | 463 | 315 | 547 |

The red filter produced slightly higher contrast images than the dark blue and dark green, but not as high as the rest of the filters. Its lowest contrast was achieved with Luminance greyscale method and had contrast 202; whereas its highest contrast was achieved with the Decolorize method and had contrast of 639. This is a significant improvement in contrast compared to the dark blue and dark green filters, but nowhere near as high of a contrast as, for example, the yellow filter.

Light blue, light green, orange and pink filters all produced slightly lower contrasts than when no filter was used. These four colour filters all have a similar spread of results, with similar greyscale methods producing similar contrasts. The highest and lowest contrasts for the light blue filter were 852 and 465 when Decolorize and Lightness methods were applied, respectively. For the light green filter the highest and lowest contrasts were 1008 and 631, again with Decolorize and Lightness methods applied, respectively. The orange filter had high and low contrasts of 968 and 494 and the pink filter had high and low contrasts of 901 and 463; both high contrasts occurred when using the Decolorize method and both low contrasts occurred when using the Lightness method.

Using no filter produced generally higher contrast than most colours, which makes sense as it allowed the most light to be captured before applying the greyscale method. Most of the greyscale methods produced similar contrasts around 1000, except from the Lightness and Luma methods which produced lower contrasts of around 700. Applying the Decolorize method to the no filter images produced the highest contrast of 1035. This was only exceeded by the yellow filtered image with the Decolorize method applied which produced contrast of 1053. The yellow filter produced quite a wide spread of results, with a lowest contrast of 469 when the Luster method was applied and the highest contrast of 1053.

Decolorize is the method of highest contrast for all filters except dark blue. The Value method generally produces the second highest contrast. Using a yellow filter with the Decolorize method produced a higher contrast than any greyscale method applied to the non-filtered images. The light green filter with the Decolorize method applied also produced a similar contrast to the non-filtered contrast. This means that the optimal filter for creating the most contrast possible on steel defect images is likely some shade of yellow, somewhere towards the yellow end of the yellow-green colour spectrum.

Each sample was scanned with each filter, and each of those images had each of the greyscale methods applied, making 63 images in total. The grey value range for the ROI of each colour filtered greyscale image was calculated. These values were then ordered highest contrast to lowest contrast to determine which greyscale method produced better contrast results across all filters and samples. For example, for sample 3 with a light blue filter, the highest grey value ranges were produced by the greyscale methods Decolorize (grey value range=77), Value (grey value range=75), and Luster (grey value range=57). Figure 5.29 shows how many times each greyscale method featured in the top three contrast results for all of the samples.

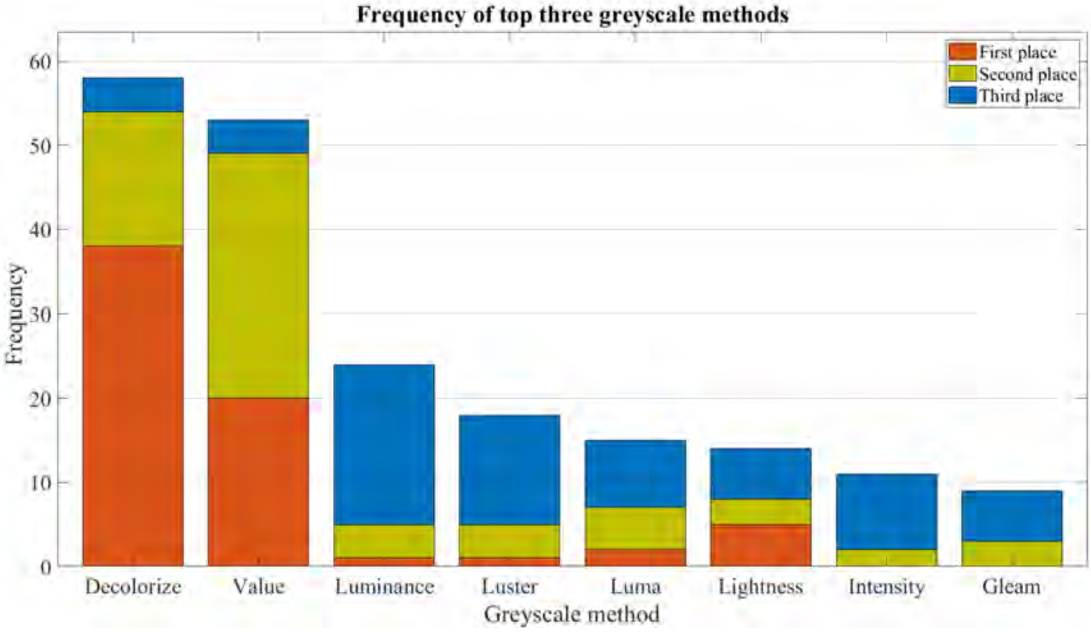


FIGURE 5.29: Frequency of which greyscale methods rank in the top three highest grey value ranges for each sample.

The Decolorize method occurs most frequently in the top three, being featured in the top three for all, except five, of the 63 images. The Decolorize method ranks as producing the highest grey value range in 38 out of 63 samples, the second highest in 16 samples, and the third highest in 4 samples. The Value method ranks highest for 20 samples, second highest for 29 samples, and third highest for 4 samples. The other methods rarely rank in highest or second highest grey value ranges, but share the spot for third highest between them. The Luminance method is most frequently the third highest method, as is the Luster method. Figure 5.29 further shows that Decolorize is the greyscale method of choice, it consistently produced higher levels of contrast than the other methods, even when colour filters were introduced.

The samples were then grouped by colour filter and put through the greyscale methods. This set of images was then used for training the decision tree to determine how colour filter affects performance. Figure 5.30 shows the best performance results of all of the greyscale methods of each colour, ie. The best performance of each colour filter is shown, regardless of greyscale method was used.

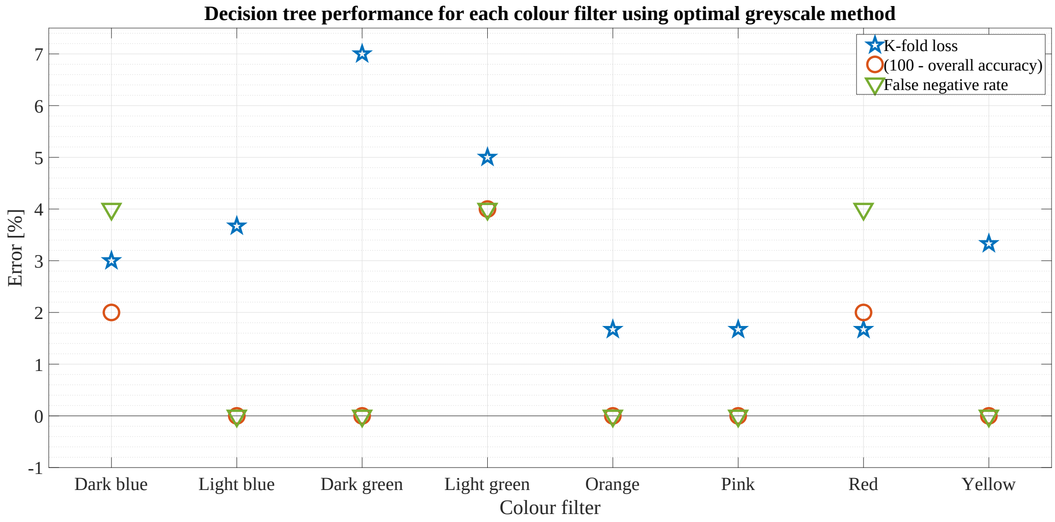


FIGURE 5.30: Performance results for a decision tree trained on each colour filtered set of images using their optimal greyscale methods.

Dark green and light green filtered images have produced classifiers with very high K-fold loss values of 7% and 5%, respectively, so these methods would likely see a decrease in accuracy when scaled up to larger data sets. Light blue and dark blue filtered images also have a wide spread of results; light blue has the third largest K-fold loss, but achieved 100% in its test predictions whilst dark blue has a middle of the range K-fold loss, but high overall error and false negative rate. Red filtered images have a low K-fold loss, but performed poorly in the test predictions, which could indicate some confusion in the data set as it is expected that the K-fold loss value is larger than the overall error from the test predictions. Orange and Pink filtered images produced the best performing classifiers, with the lowest K-fold losses of 1.67% and both scoring 100% on the test predictions. Yellow filtered images also produced a classifier which performs fairly well, only with a slightly higher K-fold loss of 3.3%.

These results show that the overall error alone is not indicative of test performance, since light blue, dark green, orange, pink and yellow filtered images all have significant K-fold loss values, but achieved 100% in their test data predictions. Compared to Figure 5.25 which shows the performance results for the non-filtered image predictions using various greyscale methods, many different colour filtered images have a better performance than the non-filtered images. The best performing greyscale method was Decolorize, with a K-Fold loss of 4.3% and an overall error of 2%. Whereas the best performing colour filtered images have a K-fold loss of 1.67% and test prediction accuracies of 100%. This would indicate that using specific coloured filters and their optimal greyscale methods could produce better performing classifiers. In particular, Orange, Pink, and possibly Yellow filters could lead to better performing classifiers.

Further testing of these colour filters and shade variations would next be carried out on a test rig on a pilot production line or roll-out table where different lighting set-ups and angles could be tested to determine the truly optimal lighting conditions. These tests give an indication of where on the colour spectrum to begin exploring for optimal conditions.

### 5.4.3 Part 3

Figure 5.31 shows the sum of the grey value ranges for each of the oil steel samples which were scanned using the colour filters and the greyscale methods, as in part 2. Each circle shows the sum of the grey value ranges for the three samples which used the same colour filter and the same greyscale method.

The dark blue and dark green filters performed particularly poorly, with minimal contrast compared to the other filter types, regardless of greyscale method. The highest contrasts produced were 77 and 65, respectively, whilst the lowest contrast produced was 16 for both filters. This is the lowest contrast produced by any of the filters.

Pink and orange colour filters produced similar results with respect to each greyscale method, except when the Decolorize method was used, which produced higher contrast with the orange filtered images compared to the pink filtered images. Their lowest contrasts were both 59, however their highest were 122 and 148, giving the orange filter a similar performance using the Decolorize method to the light blue, blight green, red, and yellow filtered images when using the Decolorize method.

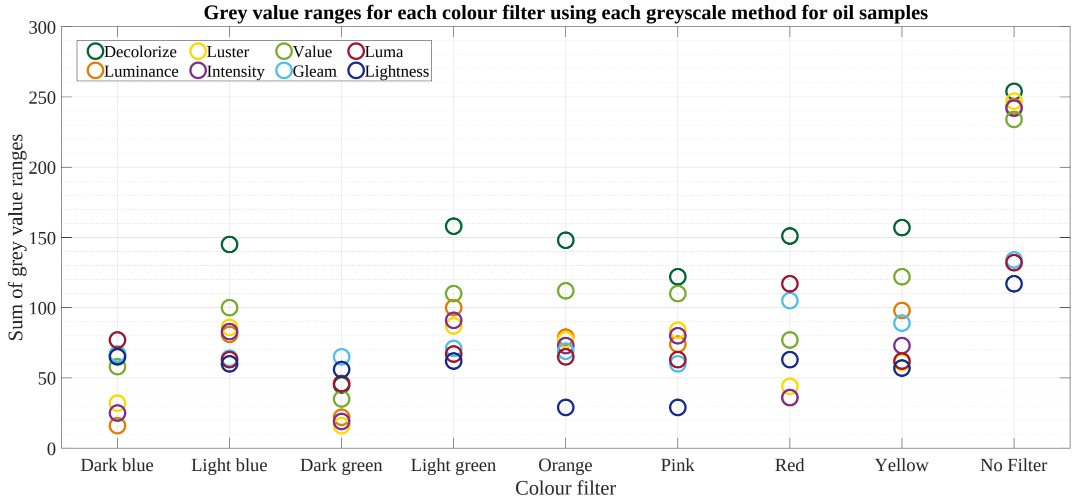


FIGURE 5.31: The sum of the grey value ranges for each colour filter using each greyscale method for the oil samples.

Light blue, light green, and yellow have all performed very similarly. Their highest contrasts were all produced by the Decolorize greyscale method with contrast around 150 and their lowest by the Lightness method with contrast around 60. Their second highest contrast was produced by the Value method averaging at 110.

The red filter produced a wider spread of results for each of the greyscale methods than any other colour filter. Typically results for different greyscale methods with the same colour filter have been grouped quite closely, e.g. three of four results will be similar, with one or two being much higher or much lower. Its highest and lowest contrasts are similar to the orange filtered images, however several of the greyscale methods produced quite different results.

The non-filtered images produced significantly higher contrasts than the colour filtered images, particularly when greyscale methods Decolorize, Luster, Value, and Intensity were used. The non-filtered image which used Luma, Gleam, and Lightness greyscale methods were similar to some of the better colour filtered results.

The light green filter did perform the best out of all the colour filters. However, considering the idea that using green light over white light is meant to make oil more visible, these results suggest that this is not necessarily the case. Taking the worst performing non-filtered image, and the best performing light green filtered image would give higher contrast when using the light green filter.

However, when taking the best-performing of each filter type, light green filter using the Decolorize method and non-filtered using the Decolorize method, the non-filtered images produced more than 60% higher contrast. So in this instance, using white light produced a far better contrast between oil and clean steel than using the light green filter.

## 5.5 Conclusions

This chapter is aimed at addressing four questions in the pursuit of an optimal image for steel defect detection. Given the challenging environment in steel production, any method which produces more useful images should enable more accurate classification.

First an optimal greyscale method is discussed and it was found that for full RGB colour images of steel the Decolorize greyscale method by Grundland and Dodgson produced the highest contrast between steel colours and defect colours. These greyscale methods were then tested alongside using several different colour filters; red, blue, green, yellow, orange, and pink. The optimal colour appeared to be somewhere on the yellow end of the spectrum, possibly more towards orange or light green. The optimal greyscale method for coloured filters was also found to be the Decolorize method. Further work in this area would include testing for a larger sample size and with a variety of shades of yellow filter to determine the optimal colour.

With respect to increasing the contrast of oil on steel, no colour of filter performs as well as a non-filtered image, although the light green filter does perform the best out of the filtered images. Further testing with a variety of oils could be carried out to determine whether this is the case for general oils or just the specific type tested here.

There are many different factors which go into optimal visual inspection configurations. This chapter establishes a starting point to begin testing various configurations in a pilot line set-up to scale up these tests. The tests carried out suggest which colours of light work well and how to optimally transform these images into greyscale images to work with existing image processing systems. A pilot line would allow expanding the range of steel samples to determine whether these results scale-up as expected and also enable evaluation of the computational cost of these methods in real-time. The findings from such a pilot-line could then be implemented as systems are upgraded across site to improve root-cause analysis and defect understanding gained from visual inspection images.

# Chapter 6

# Conclusions and Further Work

This section discusses the conclusions drawn in each chapter and how this project explored possible methods that could be used to increase efficiency and/or accuracy in the management of surface features at Tata Steel. This project looked at three areas; defect detection, weld-hole classification, and the effect of colour and greyscale methods on achieving higher contrast between defects and strip surface.

There are many generic blackbox systems on the market, but at Tata Steel Europe in South Wales, they have taken the in-house approach to defect management. The mills on site are of various ages and designs, so a generic, blanket approach to tackling defect management in all of the different areas does not suit. Furthermore, whilst typically in computer vision one looks to minimise the number of false positives, in steel defect detection it is the false negatives that will really cost the company. A large amount of false positives leads to increased computational cost, whereas any false negatives are defects which could be missed and potentially be delivered to a customer. So improving the in-house systems is a vital part of the continual management and optimisation of steel features.

In the work covered in chapter three, using Gabor filters and/or histograms to detect different types of defects, the proposed methods worked well compared to the existing method used on this particular set of defect images. Taking into consideration speed and computational cost, the method variation 2 which used just histograms, and not Gabor filters, was considered the best option. However anything short of 100% accuracy will always have room for improvement. Using machine learning to carry out defect detection means that optimal training images and classifier settings will be different for each type of material that is rolled through the mill. So for the matte steel images used in this work, this configuration worked well, but in order to implement our own detection system, settings and training data would be required for the full variety of materials which go through each inspection bed. The next steps would be to build up the data set, creating a source of training data for each material, labelling the data similarly to the proposed method, and then finding the optimal combination of methods and settings for each material.

The ability to track a weld hole's location through a steel coil's production is important as these holes are an indicator for weld locations and may require action mid-process. The work covered in chapter four used some state of the art methods to predict weld hole quality, although the networks didn't perform as well as hoped. Accurate predictions on how the networks would perform could not be made, given that there were many different variables that could be changed and that this work considered only the first and second predicted labels. The R-CNN adaptation of the GoogLeNet network available in Matlab performed the best, with a mean of 79% accuracy when considering the highest predicted score compared to the label. Two limitations were the small amount of training data it was feasible to generate and the ability to carry out very few training and testing runs due to time limitations. However, this work did highlight some important learning points for future work in this area. The data set had an uneven amount of each weld hole quality, meaning the network was exposed to bias towards the worst quality weld hole images. Yet cutting the data set down to be unbiased yielded poor results also, as there were very few images to work with. Next steps in this area would be to create a much larger data set. Further, the values for positive and negative overlap appeared to have a significant impact on accuracy when all other values were held constant, so a study of various overlap values should be carried out to optimise accuracy. The selection of anchor boxes for Faster R-CNNs is important as it allows faster computation and they should accurately reflect the shapes in the data to enable multiple sizes of features to be predicted at one time. In any further work this step would need to be repeated to ensure that the variety in the data would be included. All three of the methods used did get the correct locations of the weld holes in all of the test images. So in further work to determine appropriate detection methods these networks should be tested as defect detection methods.

There are four questions explored in chapter four; each with the goal to help increase the contrast between defect and clean steel which would improve the quality of images input into the inspection system. The methods tested both specific colours drawn from full colour sample images and taking the range of intensity values across defect regions of interest to determine optimal greyscale methods and optimal colours for filters. Answering the first question, it was determined that the greyscale method used does significantly affect the contrast in the resulting greyscale image. This result is not one widely considered in the computer vision world, as was suggested in the paper by Kanan and Cottrell [142]. However, when there is only one chance to capture a defect image, it is vital that as much information as possible is captured. So losing significant amounts of detail because the wrong greyscale method was used would drop detection and classification performance considerably. The optimal greyscale method out of those tested was the Decolorize method, which outperformed the other methods in the majority of tests it was put through. The best performing colour filters were yellow, orange, and light green, with no filter performing almost as well. Many of the filters took out too much intensity information, but using a yellow filter with the Decolorize method had higher contrast than using no filter with the Decolorize method. When investigating whether green filters increased contrast for oiled samples, it was found that light green was the best performing filter. However using no filter produced significantly higher contrast images, so if trying to highlight oil on a strip full colour images should be used. Further work in this area would involve testing these methods on a much larger array of defect types, and on a pilot line setup to more accurately recreate the lighting that would be possible in a mill setup. Other image capture aspects could also be incorporated such as camera and lighting angles, different lense types and the distance between camera, lights, and strip. It would be possible to capture the images using full spectrum lights and then, in post-processing, extract the relevant colours which provide the best information for specific defect types to aid classification. Improved classification would then allow improved root-cause analysis and defect management.

The work undertaken has inspired a pilot line project which will take the considerations of this thesis and explore line, camera and lighting setups, as well as detection and classification techniques which could influence any future production line modifications to enhance defect management.

# Appendix A

# Computer specification

| | |
|---|---|
| Processor | Intel(R) Xeon(R) CPU E5-2430 dual core |
| Speed | 2.5 GHz |
| Ram | 96 GB |
| Graphics card | NVIDIA Titan X (Pascal) |
| Operating System | Windows 7 Professional |

# Appendix B

# Figure B.1 in full detail

Full size image on next page to ensure defect names are legible.

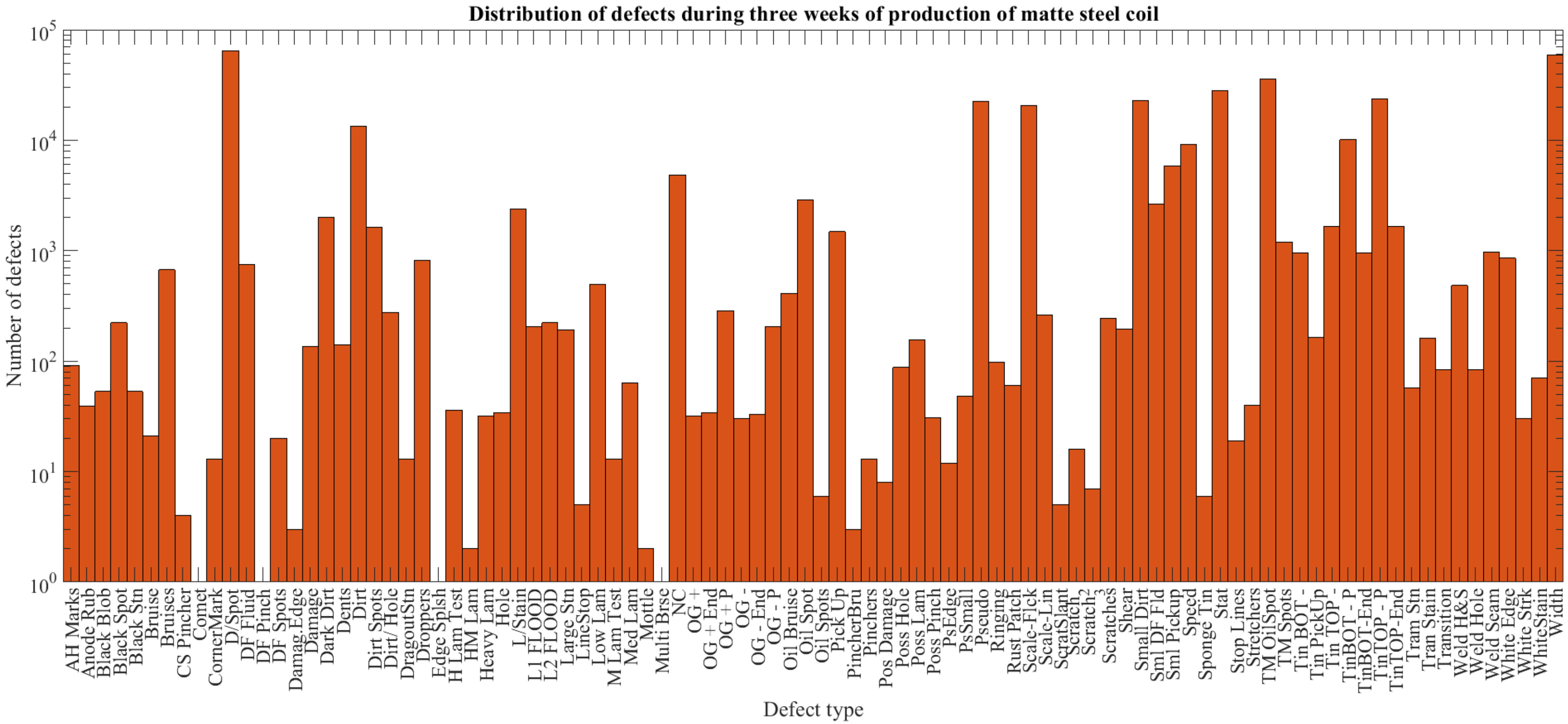


FIGURE B.1: Frequency of defects which occurred over a three week period of production of matte steel coil.

# Appendix C

# Full image results for section 5.4.2

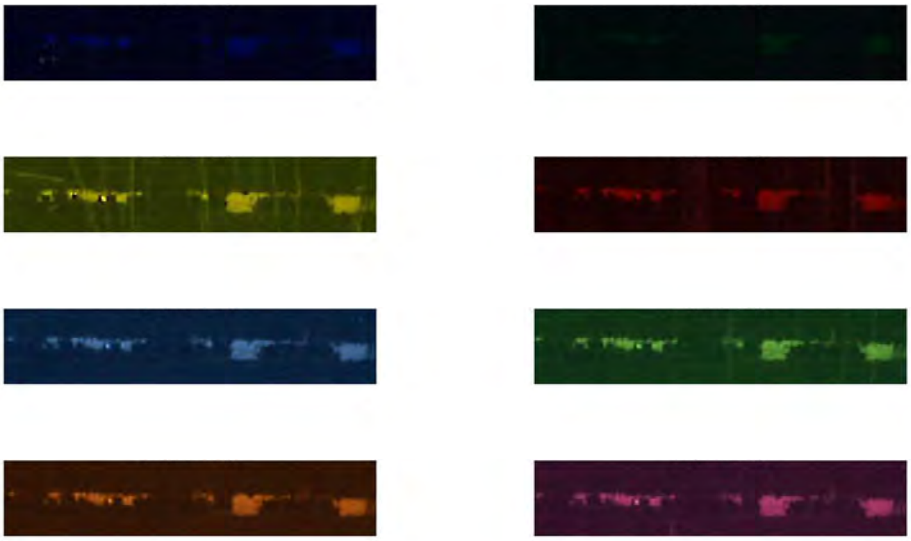


FIGURE C.1: Sample number 1 photographed using the different colour filters.

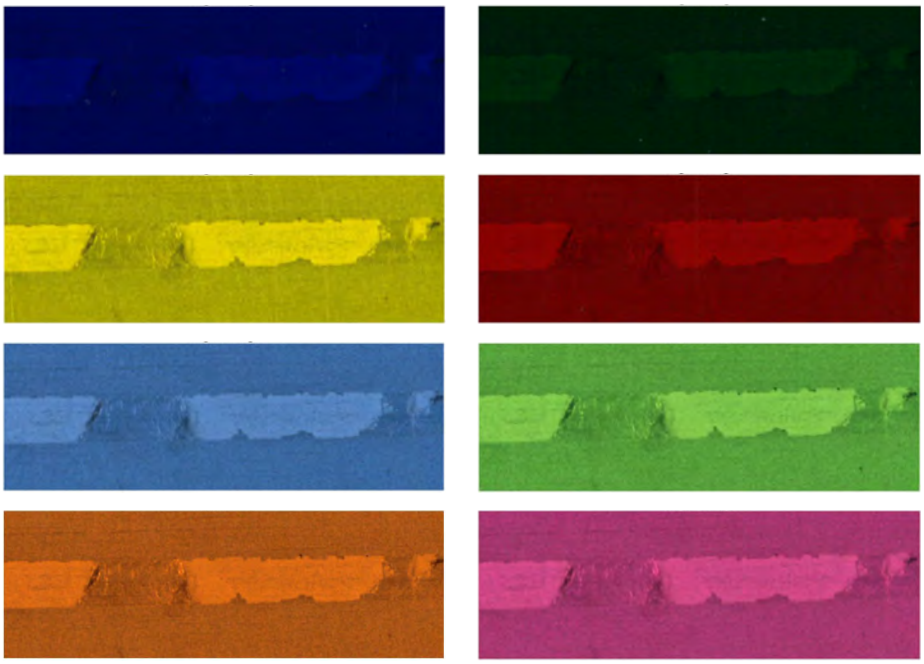


FIGURE C.2: Sample number 2 photographed using the different colour filters.

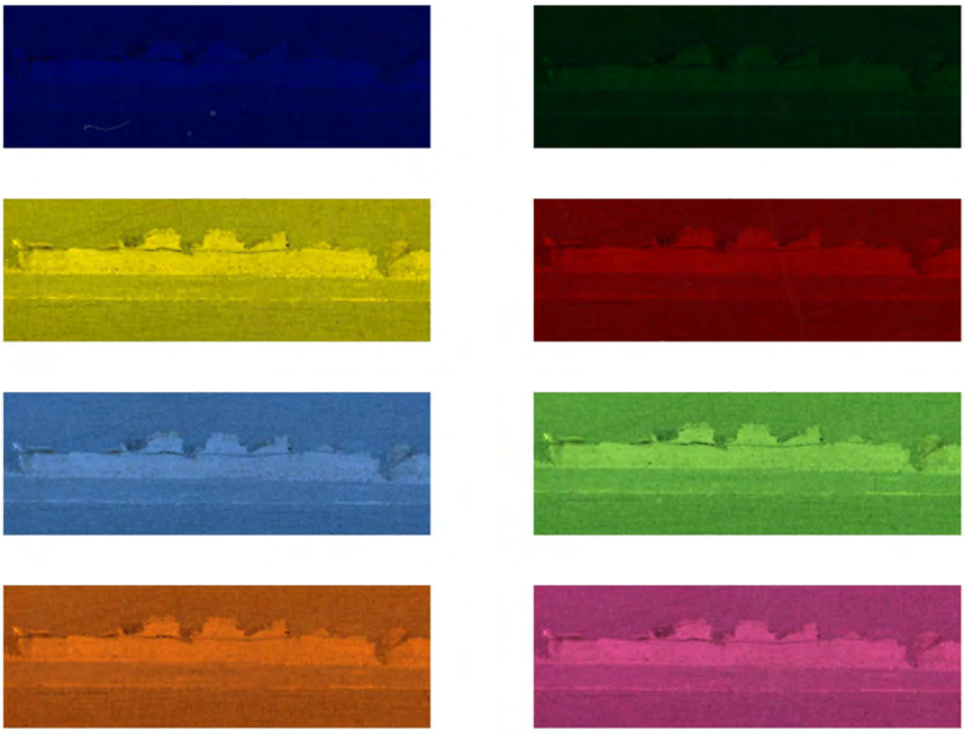


FIGURE C.3: Sample number 3 photographed using the different colour filters.

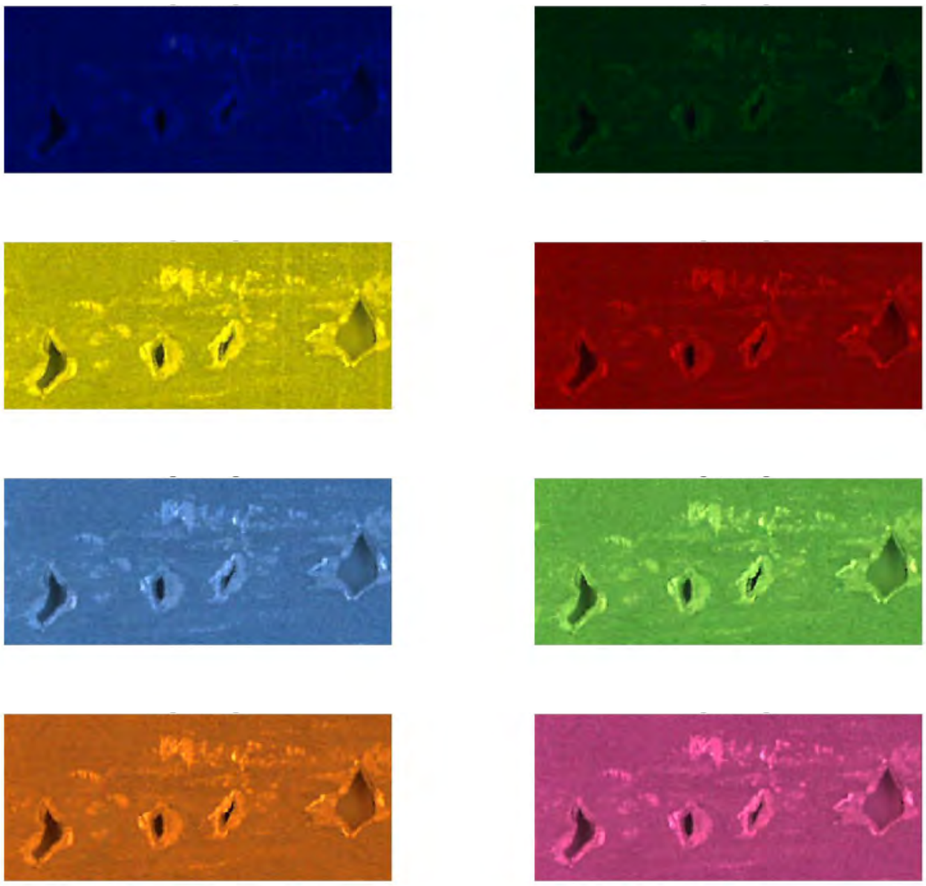


FIGURE C.4: Sample number 4 photographed using the different colour filters.

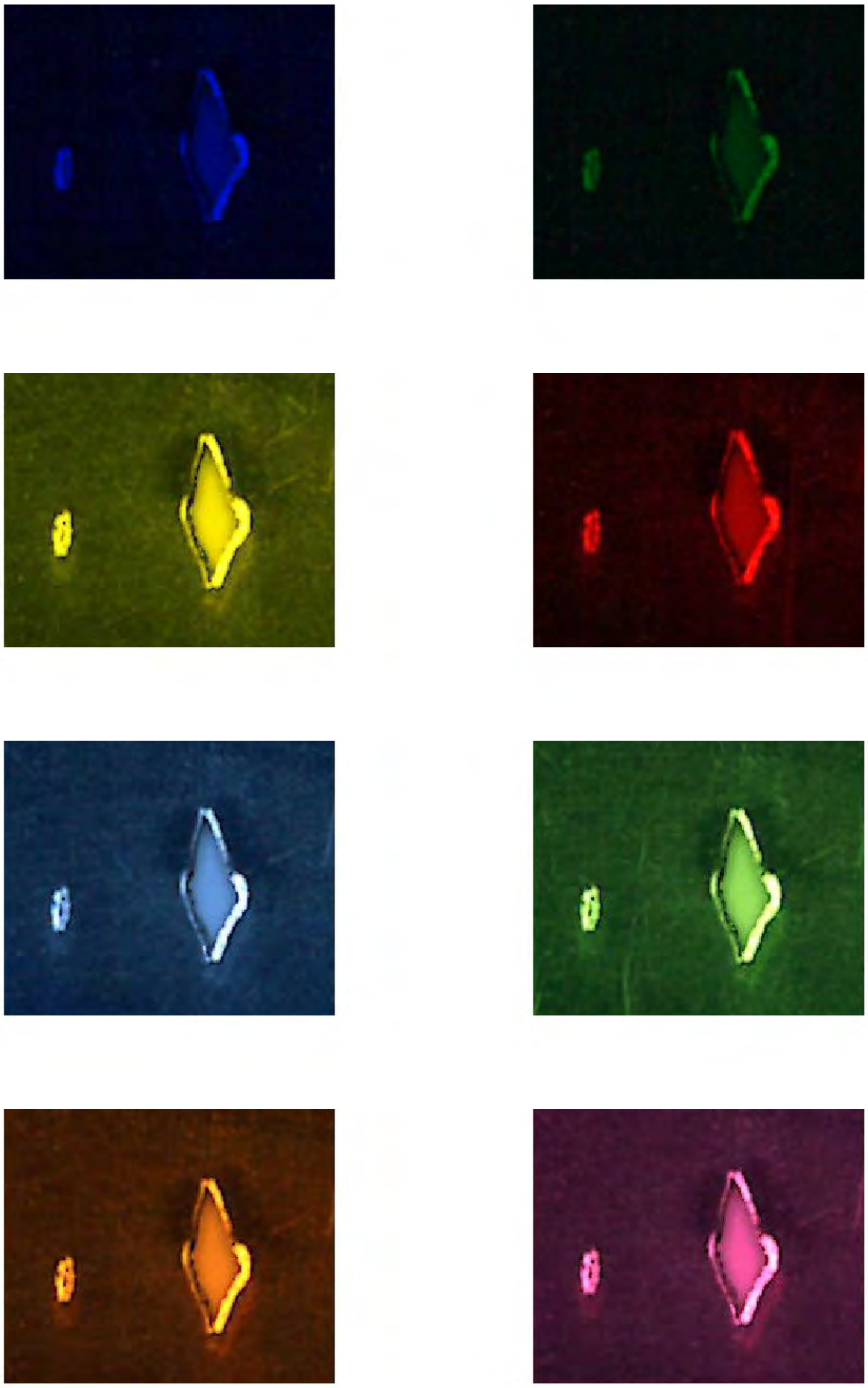


FIGURE C.5: Sample number 5 photographed using the different colour filters.

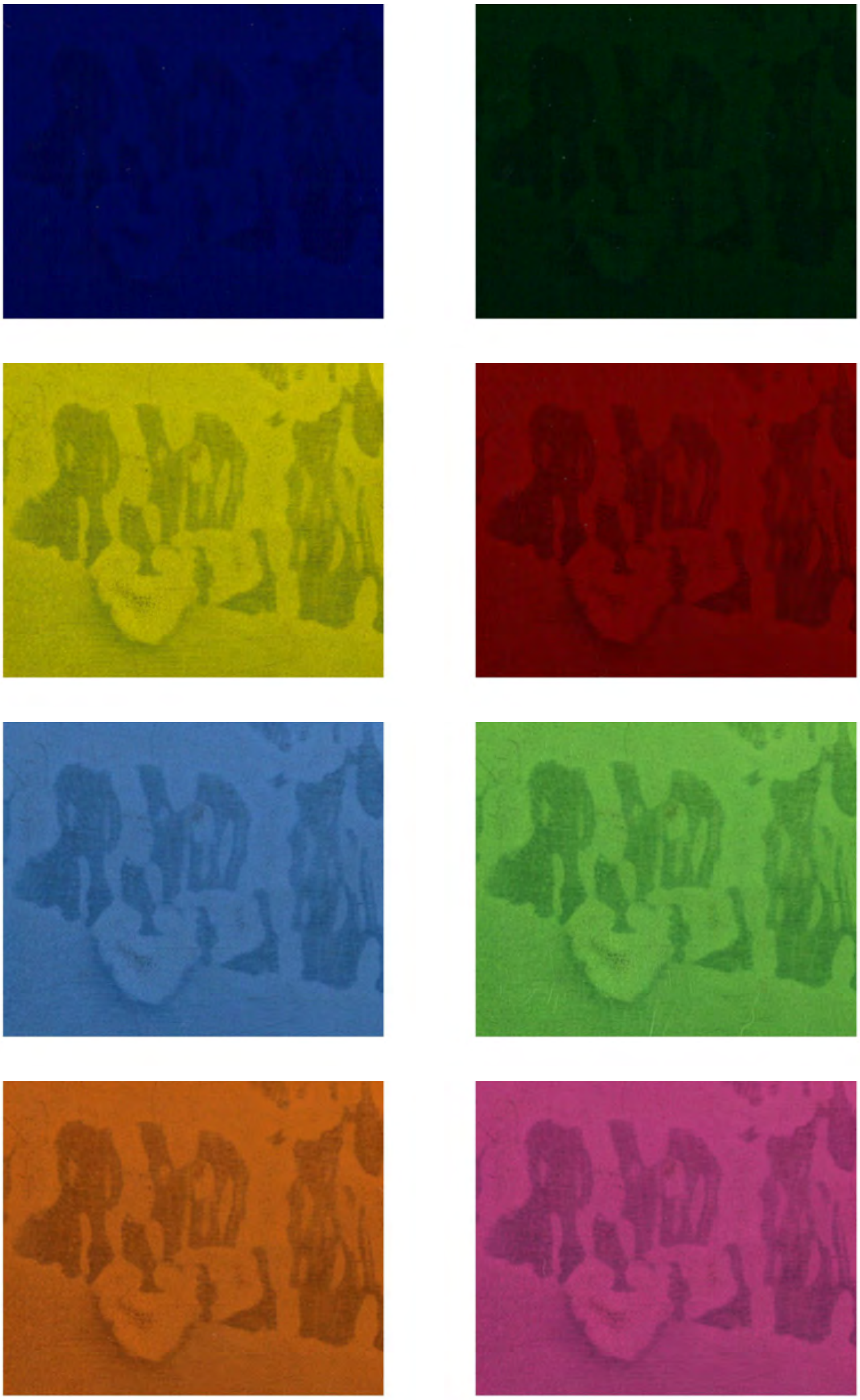


FIGURE C.6: Sample number 6 photographed using the different colour filters.

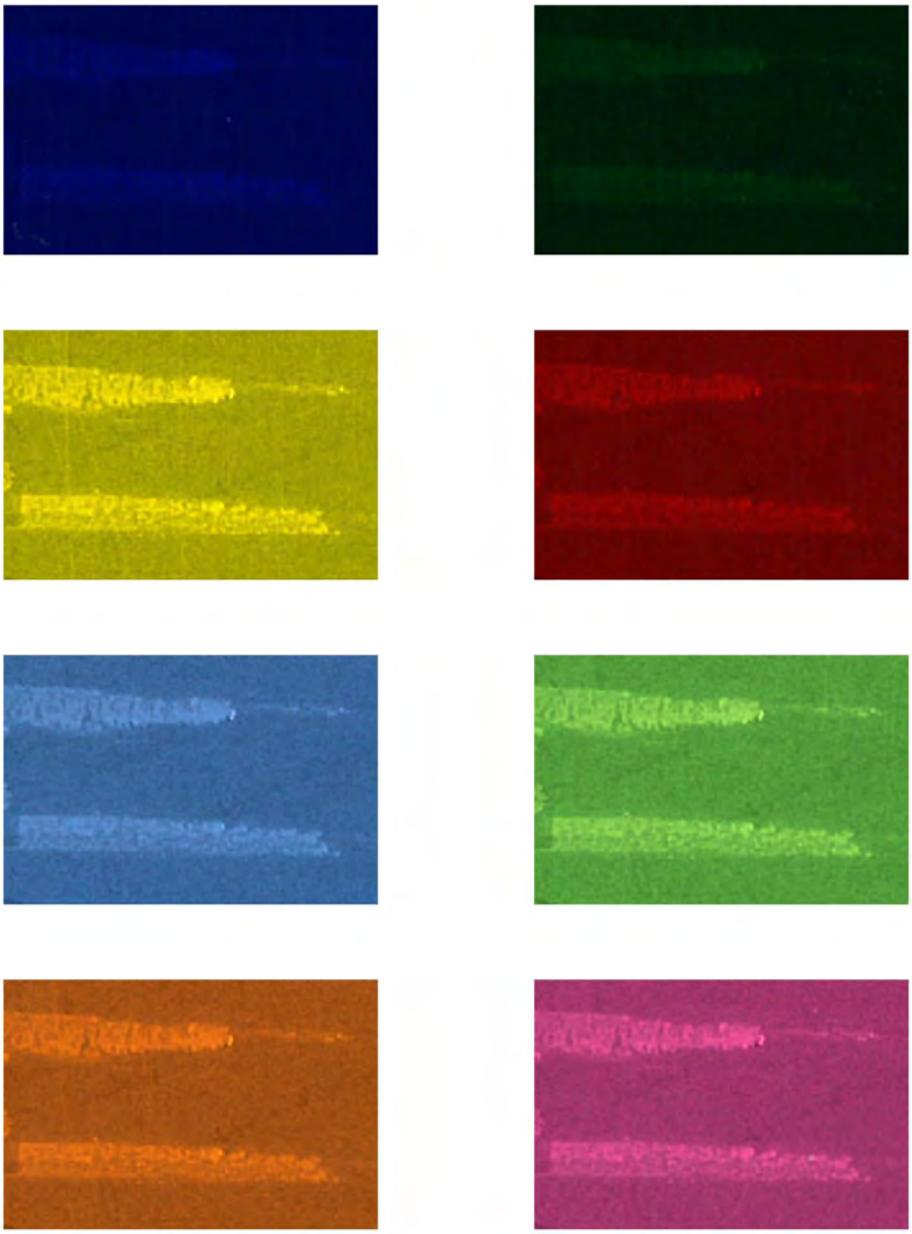


FIGURE C.7: Sample number 7 photographed using the different colour filters.

# Bibliography

[1] World Steel Association. *Monthly production 2018 - 2017*. [Online: https://www.worldsteel.org/steel-by-topic/statistics/monthly-crude-steel-and-iron-production.html Accessed: 08/05/2018].

[2] National Instruments. *A Practical Guide to Machine Vision Lighting*. [Online: http://www.ni.com/white-paper/6901/en/ Accessed 25/07/17].

[3] The Mathworks Inc. *MATLAB R2015b - 2018b*. Natick, Massachusetts, 2015-2018.

[4] Corus. *An Introduction to the Processing of Strip Steel*. January 2004.

[5] Valeria L. de la Concepción, Hernán N. Lorusso, and Hernán G. Svoboda. "Effect of carbon content on microstructure and mechanical properties of dual phase steels". In: *Procedia Materials Science* **8** (2015), pp. 1047–1056.

[6] Adnan Calik et al. "Effect of Carbon Content on the Mechanical Properties of Medium Carbon Steels". In: *A Journal of Physical Sciences: Zeitschrift für Naturforschung A* **65**(5) (2010), pp. 468–472.

[7] Jing Guo et al. "Effect of Sulphur Content on the Properties and MnS Morphologies of DH36 Structural Steel". In: *Metals* **8**(11) (2018), p. 945.

[8] World Steel Association. *Overview of the Steelmaking Process*. [Online: https://www.worldsteel.org/dms/internetDocumentList/bookshop/Steelmaking-poster/document/Overview/%20of%20th%20steelmaking%20process.pdf Accessed: 02/06/16].

[9] SteelConstruction.info. *Steel Manufacture*. [Online: http://www.steelconstruction.info/Steel_manufacture Accessed: 02/06/16].

[10] Tata Steel Europe. *Tata Steel Port Talbot Works Visitor Guide*. [Online: http://www.tatasteeleurope.com/static_files/StaticFiles/Business_Units/CSPUK/Tata-PT-Visitor-Leaflet%20Interactive.pdf Accessed: 25/05/16].

[11] Corus. *Module 1: Metallurgy and Processing of Strip Products*. Technology Training for Corus Strip Products.

[12] Tata Steel. *Defect Recognition All Products*. November 2013.

[13] Reza Abbaschian and Robert E. Reed-Hill. *Physical Metallurgy Principles*. Fourth. Cengage Learning, 2009.

[14] PMPA Speaking of Precision. *Scabs on Rolled Steel Products*. [Online: https://pmpaspeakingofprecision.com/2012/05/29/scabs-on-rolled-steel-products/ Accessed: 17/06/16].

[15] B. Nath and G. S. Cholakov. *Pollution Control Technologies - Volume III*. EOLSS Publications, 2009, p. 122.

[16] William L. Roberts. *Cold Rolling of Steel*. Marcel Dekker, Inc., 1978.

[17] P. K. Tripathy et al. "Migration of slab defects during hot rolling". In: *Ironmaking & Steelmaking* **33**(6) (2006), pp. 477–483.

[18] C. S. Krishnamoorthy and S. Rajeev. *Artificial Intelligence and Expert Systems for Engineers*. CRC Press, Inc., 2018.

[19] Enrique Castillo, Jose Manuel Gutierrez, and Ali S. Hadi. *Expert Systems and Probabilistic Network Models*. Springer, 1997.

[20] Cornelius T. Leondes. *Expert Systems: The Technology of Knowledge Management and Decision Making for the 21st Century*. Vol. **1**. Academic Press, 2002, pp. 6–15.

[21] C. E. Shannon. "Programming a Computer for Playing Chess". In: *Philosophical Magazine Series 7* **41**(314) (1950).

[22] A. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM Journal* **3**(3) (1959).

[23] Arend Hintze for The Conversation. *Understanding the four types of AI, from reactive robots to self-aware beings*. [Online: https://theconversation.com/understanding-the-four-types-of-ai-from-reactive-robots-to-self-aware-beings-67616 Accessed 10/07/2019].

[24] B. J. Copeland for Encyclopedia Britannica. *Artificial Intelligence - Expert Systems*. [Online: https://www.britannica.com/technology/artificial-intelligence/Expert-systems Accessed 10/07/2019].

[25] Viral Nagori and Bhushan Trivedi. "Types of Expert System: Comparative Study". In: *Asian Journal of Computer and Information Systems* **2**(2) (2014), pp. 20–33.

[26] Carol E. Brown and Daniel E. O'Leary. *Introduction to Artificial Intelligence and Expert Systems*. [Online: https://www.marshall.usc.edu/sites/default/files/oleary/intellcont/Introduction%20to%20AI%20and%20ES-1.pdf Accessed: 18/07/19].

[27] TechTarget. *Industrial internet of things*. [Online: https://internetofthingsagenda.techtarget.com/definition/Industrial-Internet-of-Things-IIoT Accessed 12/12/2018].

[28] Bernd Jähne. *Digital Image Processing: Concepts, Algorithms, and Scientific Applications*. Third. Springer-Verlag Berlin Heidelberg GmbH, 1995.

[29] Abhishek Yadav and Poonam Yadav. *Digital Image Processing*. University Science Press, 2009.

[30] Edmund Optics. *Section 11.1 of Imaging Resource Guide - Imaging Electronics 101: Camera Types and Interfaces for Machine Vision Applications*. [Online: https://www.edmundoptics.com/resources/application-notes/imaging/camera-types-and-interfaces-for-machine-vision-applications/ Accessed 16/07/2019.

[31] Vision Doctor. *Area scan cameras*. [Online: http://www.vision-doctor.co.uk/area-scan-cameras.html Accessed: 14/07/16].

[32] Vision Doctor. *Line scan camera basics*. [Online: http://www.vision-doctor.co.uk/line-scan-cameras.html Accessed: 20/07/16].

[33] Newton Labs. *Comparing Line Scan and Area Scan Technologies*. [Online: http://www.newtonlabs.com/line_systems.htm Accessed: 21/07/16].

[34] Jack Loughran. *Halogen bulbs to be banned in the EU; phase-out starts next week*. [Online: https://eandt.theiet.org/content/articles/2018/

08/halogen-bulbs-to-be-banned-in-the-eu-phase-out-starts-next-week/ Accessed 16/07/2019].

[35] JAI. *Color imaging in machine vision: how to choose the right camera for your application - Chapter 3*. [Online: https://www.jai.com/machine-vision-color-imaging-cameras Accessed 17/07/2019].

[36] Yu-Cheng Fan and Yi-Feng Chiang. "Discrete Wavelet Transform on Color Picture Interpolation of Digital Still Camera". In: *VLSI Design* (2013). Article ID 738057.

[37] Omron Microscan. *Bright Field and Dark Field Lighting*. [Online: https://www.microscan.com/en-us/resources/know-your-tech/bright-field-and-dark-field-lighting Accessed 17/07/2019].

[38] Ramesh Jain, Rangachar Kasturi, and Brian G. Schunck. *Machine Vision*. McGraw-Hill, Inc., 1995, pp. 140 –185.

[39] R. Jain et al. *Machine Vision*. McGraw-Hill, 1995, p. 158.

[40] Sunil Kumar Katiyar and P. V. Arun. "Comparitive analysis of common edge detection techniques in context of object extraction". In: *IEEE TGRS* **50** (2014), pp. 68–78.

[41] Raman Maini and Dr. Himanshu Aggarwal. "Study and comparison of various image edge detection techniques". In: *International Journal of Image Processing* **3** (2009), pp. 1–11.

[42] G. Shrivakshan and Dr. C. Chandrasekar. "A comparison of various edge detection techniques used in image processing". In: *International Journal of Computer Science* **9**(5) (2012), pp. 269–276.

[43] Mohsen Sharifi, Mahmoud Fathy, and Maryam Tayefeh Mahmoudi. "A classified and comparitive study of edge detection algorithms". In: *Proceedings of the International Conference on Information Technology: Coding and Computing*. 2002.

[44] Chris Solomon and Toby Breckon. *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. John Wiley and Sons, 2011, pp. 44–46.

[45] Nayan Patel, Abhishek Shah, and Mayur Mistry. "A Study of Digital Image Filtering Techniques in Spatial Image Processing". In: *International Conference on Convergence of Technology* (2014).

[46] Yanuar Wicaksono, Romi Satria Wahono, and Vincent Suharto. "Color and Texture Feature Extraction Using Gabor Filter - Local Binary Patterns for Image Segmentation with Fuzzy C-Means". In: *Intelligent Systems* (2015), pp. 15–21.

[47] Duo Zhang, Guangshuai Gao, and Chunlei Li. "Fabric defect detection algorithm based on Gabor filter and low-rank decomposition". In: *International Conference on Digital Image Processing* **10033** (2016).

[48] Yundong Li and Cheng Zhang. "Automated vision system for fabric defect inspection using Gabor filters and PCNN". In: *SpringerPlus* **5**(765) (2016).

[49] Junfeng Jing et al. "The fabric defect detection based on CIE L*a*b* color space using 2-D Gabor filter". In: *The Journal of the Textile Institute* **107** (2015), pp. 1305–1313.

[50] Le Tong et al. "Optimal Gabor Filtering for the Inspection of Striped Fabric". In: *Artifical Intelligence on Fashion and Textiles. AITA 2018. Advances in Intelligent Systems and Computing* **849** (2019), pp. 291–297.

[51] J. G. Daugman. "Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters". In: *Optical Society of America A: Optics and Image Science, and Vision* **2**(7) (1985), pp. 1160–1169.

[52] Ville Kyrki, Joni-Kristian Kamarainen, and Heikki Kälviäinen. "Simple Gabor feature space for invariant object recognition". In: *Pattern Recognition Letters* **25** (2004), pp. 311–318.

[53] Hal Daumé III. *A Course in Machine Learning*. TODO, 2015.

[54] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2013.

[55] M. Seetha, I. V. MuraliKrishna, and B. L. Deekshatulu. "Comparison of Advanced Techniques of Image Classification". In: GIS Development.

[56] Pooja Kamavisdar, Sonam Saluja, and Sonu Agrawal. "A survey on Image Classification Approaches and Techniques". In: *International Journal of Advanced Research in Computer and Communication Engienering* **2** (2013), pp. 1005–1009.

[57] Eduardo Massad et al. "Basic Concepts of Fuzzy Sets Theory". In: *Fuzzy Logic in Action: Applications in Epidemiology and Beyond* (2008), pp. 11–12.

[58] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[59] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory To Algorithms*. Cambridge University Press, 2014.

[60] Jason Brownlee. *Gradient Descent With Momentum from Scratch*. [Online: `https://machinelearningmastery.com/gradient-descent-with-momentum-from-scratch/` Accessed: 07/07/2021]. Machine Learning Mastery.

[61] Liang Jia et al. "Fabric defect inspection based on lattice segmentation and Gabor filtering". In: *Neurocomputing* **238** (2017), pp. 84–102.

[62] G. M. Atiqur Rahaman and Md. Mobarak Hossain. "Automatic defect detection and classification technique from image: A special case using ceramic tile". In: *International Journal of Computer Science and Information Security* (2009), pp. 22–30.

[63] Yuan Li et al. "Measurement and Defect Detection of the Weld Bead Based on Online Vision Inspection". In: *Instrumentation and Measurement* **59**(7) (2010), pp. 1841–1849.

[64] Colin S. C. Tsang, Henry Y. T. Hgan, and Grantham K. H. Pang. "Fabric inspection based on the Elo rating method". In: *Pattern Recognition* **51** (2016), pp. 378–394.

[65] Yong Jie Zhao, Yun Hui Yan, and Ke Chen Song. "Vision-based automatic detection of steel surface defects in the cold rolling process: considering the influence of industrial liquids and surface textures". In: *International Journal of Advanced Manufacturing Technology* **90**(5-8) (2017), pp. 1665–1678.

[66] Yunwon Park and In So Kweon. “Ambiguous Surface Defect Image Classification of AMOLED Displays in Smartphones”. In: *Industrial Informatics* **12**(2) (2016), pp. 597–607.

[67] Saeed Hosseinzadeh Hanzaei, Ahmad Afshar, and Farshad Barazandeh. “Automatic detection and classification of the ceramic tiles’ surface defects”. In: *Pattern Recognition* **66** (2017), pp. 174–189.

[68] LinLin Shen, Li Bai, and Michael Fairhurst. “Gabor wavelets and General Discriminant Analysis for face identification and verification”. In: *Image and Vision Computing* **25** (2007), pp. 553–563.

[69] Beom-Seok Oh et al. “A gabor-based network for heterogeneous face recognition”. In: *Neurocomputing* **261** (2017), pp. 253–265.

[70] Lingraj Dora et al. “An evolutionary single gabor kernel based filter approach to face recognition”. In: *Engineering applications of artificial intelligence* **62** (2017), pp. 286–301.

[71] Liliang Zhang et al. “Is Faster R-CNN Doing Well for Pedestrian Detection?” In: *Computer Vision ECCV* (2016), pp. 443–457.

[72] Jamie Shotton et al. “Real-time human pose recognition in parts from single depth images”. In: *Conference on Computer Vision and Pattern Recognition* (2011), pp. 1297–1304.

[73] Ali Younesi and Mehdi Chehel Amirani. “Gabor filter and Texture based Features for Palmprint Recognition”. In: *International Conference on Computational Science* **238** (2017).

[74] Somnath Dey and Debasis Samanta. “Iris data indexing method using gabor energy features”. In: *Information Forensics and Security* **7**(4) (2018), pp. 18–24.

[75] Carsten Gottschlich. “Curved-Region-Based Ridge Frequency Estimation and Curved Gabor Filters for Fingerprint Image Enhancement”. In: *Image Processing* **21**(4) (2012), pp. 2220–2227.

[76] Zehang Sun, George Bebis, and Ronald Miller. “On-Road Vehicle Detection Using Evolutionary Gabor Filter Optimization”. In: *Intelligent Transportation Systems* **6**(2) (2005), pp. 125–137.

[77] Vladimir Tadic, Miodrag Popovic, and Peter Odry. “Fuzzified gabor filter for license plate detection”. In: *Engineering applications of artificial intelligence* **48** (2016), pp. 40–58.

[78] U. Rajendra Acharya et al. “Deep convolutional neural network for the automated detection and diagnosis of seizure using EEG signals”. In: *Computers in Biology and Medicine* **100** (2018), pp. 270–278.

[79] Yiting Xie et al. “Automated segmentation of cardiac visceral fat in low-dose non-contrast chest CT images”. In: *Medical Imaging 2015: Computer-Aided Diagnosis* **9414** (2015).

[80] Tonya White et al. “Automated quality assessment of structural magnetic resonance images in children: Comparison with visual inspection and surface-based reconstruction”. In: *Human Brain Mapping* **39** (2017), pp. 1218–1231.

[81] Nirbhar Neogi, Dusmanta K. Mohanta, and Pranab K. Dutta. “Review of vision-based steel surface inspection systems”. In: *EURASIP Journal on Image and Video Processing* **50** (2014).

[82] Xiaohong Sun et al. "Research Progress of Visual Inspection Technology of Steel Products - A Review". In: *Applied Sciences* **8**(11) (2018). Article number 2195.

[83] Ke Xu et al. "Application of RNAMlet to surface defect identification of steels". In: *Optics and Lasers in Engineering* **105** (2018), pp. 110 –117.

[84] Huijun Hu et al. "Surface defect classification in large-scale strip steel image collection via hybrid chromosome genetic algorithm". In: *Neurocomputing* **181** (2016), pp. 86 –95.

[85] Anastasia Paramore and Roeman Kirmse FIMMM. "Industry 4.0 - Surface Inspection of Steel Uncovered". In: *Materials World magazine* (2020).

[86] Judi E. See. *Visual Inspection: A Review of the Literature*. Sandia National Laboratories. Albuquerque, New Mexico, USA, 2012.

[87] UNC Vision Lab. *ImageNet Large scale Visual Recognition Challenge 2016 Results Task 2a*. [Online: `http://image-net.org/challenges/LSVRC/2016/results#det` Accessed: 02/05/2019].

[88] Jong Pil Yun et al. "Defect inspection system for steel wire rods produced by hot rolling process". In: *International Journal of Advanced Manufacturing Technology* **70** (2014), pp. 1625–1634.

[89] Prof. P. A. Wankhede, Prof. Ragini S. Gawande, and Prof. Neha A. Khare. "Gabor Filter for Surface Inspection in Metallic Industry". In: *International journal of advanced research in computer science and software engineering* **6**(3) (2016), pp. 89–92.

[90] Chiatali Tikhe and J. S. Chitode. "Metal Surface Inspection for Defect Detection and Classication using Gabor Filter". In: *International Journal of Innovative Research in Science, Engineering and Technology* **3**(6) (2016), pp. 13702–13709.

[91] Carlos G. Spinola et al. "Real-Time Image Processing for Edge Inspection and Defect Detection in Stainless Steel Production Lines". In: *IEEE International Conference on Image Systems and Techniques* (2011), pp. 170–175.

[92] Vilas H. Gaidhane et al. "An improved edge detection approach and its application in defect detection". In: *IOP Conference Series: Materials Science and Engineering* **244** (2017).

[93] R. Manish, Adigopula Venkatesh, and S. Denis Ashok. "Machine Vision Based Image Processing Techniques for Surface Finish and Defect Inspection in a Grinding Process". In: *ICMMM Materials Today: Proceedings* **5**(5 part 2) (2018), pp. 12792–12802.

[94] M. Selvi and D. Jenefa. "Automated Defect detection of Steel Surface Using Neural Network Classifier with Co-occurrence Features". In: *International Journal of Advanced Research in Computer Science and Software Engineering* **4**(3) (2014), pp. 60–67.

[95] M. Arun, R. Prathipa, and P. S. G. Krishna. "Automatic Defect Detection of Steel Products Using Supervised Classifier". In: *International Journal of Innovative Research in Computer and Communication Engineering* **2**(3) (2014), pp. 3630–3635.

[96] Doo chul Choi et al. "Pinhole detection in steel slab images using Gabor filter and morphological features". In: *Applied Optics* **50**(26) (2011), pp. 5122–5129.

[97] Mostafa Sadeghi, Faezeh Memarzadehzavareh, and Mojtabaa Abedzadehzavareh. "Applying gabor wavelet in image processing for defect detection in steel plates". In: *International journal of advanced biotechnology and research* **7**(4) (2016), pp. 253–258.

[98] Jingjing Deng et al. "Age-related Macular Degeneration Detection and Stage Classification using Choroidal OCT Images". In: *International Conference on Image Analysis and Recognition* (2016), pp. 707–715.

[99] *ISRA Vision Parsytec*. [Online: `https://www.isra-parsytec.com/` Accessed: 03/05/2019].

[100] Cognex. *Industries Overview*. [Online: `https://www.cognex.com/en-gb/industries` Accessed: 03/05/2019].

[101] Ametek. *Surface Vision*. [Online: `https://www.ametek.com/products/businessunits/processandanalyticalinstruments/surfacevision` Accessed: 12/07/2019].

[102] Zhi-Qiang Liu, Jin-Hai Cai, and Richard Buse. *Handwriting Recognition: Soft Computing and Probabilistic Approaches*. Springer, 2003, pp. 32–33.

[103] Muhammad Hussain et al. "Mass Detection in Digital Mammograms Using Optimized Gabor Filter Bank". In: *International Symposium of Advances in Visual Computing* (2012), pp. 82–91.

[104] Kaushik Roy and Prabir Bhattacharya. "Level Set Approaches and Adaptive Asymmetrical SVMs Applied for Nonideal Iris Recognition". In: *International Conference in Image Analysis and Recognition* (2009), pp. 418–428.

[105] Tianqi Zhang and Bao-Liang Lu. "Selecting Optimal Orientations of Gabor Wavelet Filters". In: *International Conference in Image and Signal Processing* (2010), pp. 218–227.

[106] Fernando Cervantes-Sanchez et al. "Coronary artery segmentation in X-ray angiograms using gabor filters and differential evolution". In: *Applied Radiation and Isotopes* **13** (2018), pp. 18–24.

[107] A. Sherly Alphonse and Dejey Dharma. "Enhanced gabor, hypersphere-based normalization and pearson general kernel-based discriminant analysis for dimension reduction and classification of facial emotions". In: *Expert systems with applications* **90** (2017), pp. 127–145.

[108] Nam Chul Kim and Hyun Joo So. "Directional statistical gabor features for texture classification". In: *Pattern recognition letters* **112** (2018), pp. 18–26.

[109] Gang Zhang and Zong-Min Ma. "Texture feature extraction and description using gabor wavelet in content-based medical retrieval". In: *Proceedings of IC on wavelet analysis and pattern recognition* (2007).

[110] Mohammad Haghighat, Saman Zonouz, and Mohamed Abdel-Mottaleb. "Identification Using Encrypted Biometrics". In: *Computer Analysis of Images and Patterns* (2013), pp. 440–448.

[111] The Mathworks Inc. *Decision Trees*. [Online: http://uk.mathworks.com/help/stats/classification-trees-and-regression-trees.html Accessed: 24/07/2017].

[112] Ethem Alpaydin. *Introduction to Machine Learning*. Second edition, The MIT Press, 2010.

[113] The Mathworks Inc. *Growing Decision Trees*. [Online: https://uk.mathworks.com/help/stats/growing-decision-trees.html Accessed: 07/07/2021].

[114] Leo Breiman. "Random Forests". In: *Machine Learning* **45** (2001), pp. 5–32.

[115] The Mathworks Inc. *'fitctree'*. [Online: http://uk.mathworks.com/help/stats/fitctree.html Accessed: 18/11/16].

[116] J. R. Quinlan. "Simplifying Decision Trees". In: *International Journal of Human-Computer Studies* **51** (1999), pp. 497–510.

[117] The Mathworks Inc. *'imgaborfilt'*. [Online: https://uk.mathworks.com/help/images/ref/imgaborfilt.html Accessed: 03/05/2019].

[118] A. K. Jain and F. Farrokhnia. "Unsupervised texture segmentation using Gabor filters". In: *Pattern Recognition* **24**(12) (1991), pp. 1167–1186.

[119] L. Breiman. "Heuristics of Instability and Stabilization in Model Selection". In: *Annals of Statistics* (1996).

[120] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016, pp. 16–18.

[121] L. Fei-Fei. "ImageNet: crowdsourcing, benchmarking & other cool things". In: *CMU VASC Seminar* (March 2010).

[122] Stanford Vision Lab at Stanford University and Princeton University. *ImageNet*. [Online: http://www.image-net.org Accessed: 13/10/2019].

[123] Jason Brownlee at Machine Learning Mastery. *A Gentle Introduction to the ImageNet Challenge (ILSVRC)*. [Online: https://machinelearningmastery.com/introduction-to-the-imagenet-large-scale-visual-recognition-challenge-ilsvrc/ Accessed: 13/10/2019].

[124] Andrej Karpathy blog. *What I learned from competing against a ConvNet on ImageNet*. [Online: https://www.embedded-vision.com/industry-analysis/blog/deep-learning-five-and-half-minutes Accessed: 13/10/2019].

[125] Videantis. *Deep Learning in Five and a Half Minutes*. [Online: http://www.videantis.com/deep-learning-in-five-and-a-half-minutes.html Accessed: 13/10/2019].

[126] The MathWorks Inc. *Pretrained Deep Neural Networks*. [Online: https://uk.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html Accessed: 24/09/19].

[127] The Mathworks Inc. *resnet50*. [Online: https://uk.mathworks.com/help/deeplearning/ref/resnet50.html Accessed: 23/09/2019].

[128] The Mathworks Inc. *resnet101*. [Online: https://uk.mathworks.com/help/deeplearning/ref/resnet101.html Accessed: 24/09/2019].

[129] The Mathworks Inc. *googlenet*. [Online: https://uk.mathworks.com/help/deeplearning/ref/googlenet.html Accessed: 25/09/2019].

[130] C. Szegedy et al. "Going deeper with convolutions". In: *Computer Vision and Pattern Recognition* (2015), pp. 1–9.

[131] Kaiming He et al. "Deep residual learning for image recognition". In: *IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[132] The Mathworks Inc. *Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN*. [Online: `https://uk.mathworks.com/help/vision/ug/getting-started-with-r-cnn-fast-r-cnn-and-faster-r-cnn.html` Accessed: 25/09/2019].

[133] The Mathworks Inc. *Faster R-CNN Examples*. [Online: `https://uk.mathworks.com/help/vision/ug/faster-r-cnn-examples.html` Accessed: 23/09/2019].

[134] The Mathworks Inc. *fullyConnectedLayer*. [Online: `https://uk.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.fullyconnectedlayer.html` Accessed: 25/10/2019].

[135] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC, 2006, p. 198.

[136] The Mathworks Inc. *softmaxLayer*. [Online: `https://uk.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.softmaxlayer.html` Accessed: 25/10/2019].

[137] The Mathworks Inc. *classificationLayer*. [Online: `https://uk.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.classificationoutputlayer.html` Accessed: 08/11/2019].

[138] The Mathworks Inc. *rcnnBoxRegressionLayer*. [Online: `https://uk.mathworks.com/help/vision/ref/nnet.cnn.layer.rcnnboxregressionlayer.html` Accessed 20/11/2019].

[139] The Mathworks Inc. *Anchor Boxes forr Object Detection*. [Online: `https://uk.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html` Accessed 09/01/2019].

[140] The Mathworks Inc. *trainFasterRCNNObjectDetector*. [Online: `https://uk.mathworks.com/help/vision/ref/trainfasterrcnnobjectdetector.html` Accessed: 26/02/2020.

[141] H.H. Barnum Company. *Do you rely on parts inspection? Consider going green.* [Online: `https://www.hhbarnum.com/detect-hard-to-find-surface-imperfections-with-green-led-lighting/` Accessed 28/06/2019].

[142] Christopher Kanan and Garrison Cottrell. "Color-to-Grayscale: Does the Method Matter in Image Recognition?" In: *PloS one* **7** (2012).

[143] Joseph Bigun. *Vision with Direction: A Systematic Introduction to Image Processing and Computer Vision*. Springer-Verlag Berlin Heidelberg, 2006, pp. 21–31.

[144] Joseph Bigun. *Vision with Direction: A Systematic Introduction to Image Processing and Computer Vision*. Springer-Verlag Berlin Heidelberg, 2006, pp. 3–7.

[145] Stephen Westland, Caterina Ripamonti, and Vien Cheung. *Computational Colour Science using MATLAB*. John Wiley and Sons, Ltd., 2012, pp. 93–98.

[146] Michael Dattner and Daniel Bohn. *Printing on Polymers: 20 - Characterization of Print Quality in Terms of Colorimetric Aspects*. William Andrew Publishing, 2016, pp. 329–345.

[147] Colour & vision Research Laboratory at the Institute of Ophthalmology at UCL. *Cone Fundamentals*. [Online: `http://cvrl.ucl.ac.uk/cones.htm` Accessed: 27/08/2019].

[148] W. S. Stiles and J. M. Burch. "NPL colour-matching investigation: Final report". In: *Optica Acta* **6** (1959), pp. 1–26.

[149] A. Stockman, L. T. Sharpe, and C. C. Fach. "The spectral sensitivity of the human short-wavelength cones". In: *Vision Research* **39** (1999), pp. 2901–2927.

[150] A. Stockman and L. T. Sharpe. "Spectral Sensitivities of the middle- and long-wavelength sensitive cones derived from measurements in observers of known genotype". In: *Vision Rsearch* **40** (2000), pp. 1711–1737.

[151] Wikipedia. *File:CIExy1931.svg*. [Online: `https://en.wikipedia.org/wiki/File:CIExy1931.svg` Accessed: 28/08/2019].

[152] Chris Solomon and Toby Breckon. *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. John Wiley and Sons, 2011, pp. 9–11.

[153] Tinku Acharya and Ajoy K. Ray. *Image Processing: Principles and Applications*. John Wiley and Sons Inc., 2005, pp. 44–45.

[154] Cambridge in Colour. *Understanding Gamma Correction*. [Online: `https://www.cambridgeincolour.com/tutorials/gamma-correction.htm` Accessed 27/07/2020].

[155] The Mathworks Inc. *rgb2gray*. [Online: `https://uk.mathworks.com/help/matlab/ref/rgb2gray.html` Accessed: 02/03/2020].

[156] Rang M. H. Nguyen and Michael S. Brown. "Why You Should Forget Luminance Conversion and Do Something Better". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI* (2017), pp. 5920–5928.

[157] Mark Grundland and Neil Dodgson. "Decolorize: Fast, contrast enhancing, color to grayscale conversion". In: *Pattern Recognition* **40** (2007), pp. 2891 –2896.

[158] Rosco Laboratories Inc. *My Color Desktop*. [Online: `https://emea.rosco.com/en/mycolor` Accessed: 29/08/2019.

[159] LEE Filters. *Colour Effect filters*. [Online: `http://www.leefilters.com/lighting/colour-list.html` Accessed: 29/08/2019.

[160] The Mathworks Inc. *kfoldLoss*. [Online: `https://uk.mathworks.com/help/stats/classificationpartitionedmodel.kfoldloss.html` Accessed: 03/09/2019].