Master's thesis
Master's Programme in Data Science

Inverse Mathematics Enhanced Neural Networks to Improve Defect Detection on Radiation Detectors

Muzaffer Gur Ersalan

December 4, 2019

Supervisor(s): Professor Samuli Siltanen, Dr. Alex Winkler

Examiner(s): Professor Samuli Siltanen
Dr. Alex Winkler

University of Helsinki
Faculty of Science

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Degree programme
Faculty of Science	Master's Programme in Data Science

Tekijä — Författare — Author

Muzaffer Gur Ersalan

Työn nimi — Arbetets titel — Title

Inverse Mathematics Enhanced Neural Networks to Improve Defect Detection on Radiation Detectors

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidantal — Number of pages
Master's thesis	December 4, 2019	66

Tiivistelmä — Referat — Abstract

In this thesis, Convolutional Neural Networks (CNN) and Inverse Mathematic methods will be discussed for automated defect detection in materials that are used for radiation detectors. The first part of the thesis is dedicated to the literature review on the methods that are used. These include a general overview of Neural Networks, computer vision algorithms and Inverse Mathematics methods, such as wavelet transformations, or total variation denoising. In the Materials and Methods section, how these methods can be utilized in this problem setting will be examined. Results and Discussions part will reveal the outcomes and takeaways from the experiments. A focus of this thesis is put on the CNN architecture that fits the task best, how to optimize that chosen CNN architecture and discuss, how selected inputs created by Inverse Mathematics influence the Neural Network and it's performance. The results of this research reveal that the initially chosen Retina-Net is well suited for the task and the Inverse Mathematics methods utilized in this thesis provided useful insights.

Avainsanat — Nyckelord — Keywords

Convolutional Neural Networks, Defect Detection, Inverse Problems, Wavelet Transformation

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Acknowledgement

I would like to thank my advisor Alex Winkler for actively supporting me with my thesis, providing me with invaluable knowledge and being reachable all the time. I would like to thank Prof. Samuli Siltanen for giving me the opportunity to write this thesis in conjunction with Inverse Mathematics research group in University of Helsinki. Also, I would like to thank Detection Technology OY for recruiting me as a Thesis Worker, enabling me to see my work contributing to a real time project. Finally, I would like to thank my family who made starting this program possible in the first place.

Contents

1. Introduction

Defect detection can be defined as the process of finding imperfections or weaknesses in a given material. Such materials can vary from being metals, photographs, a crack in a pipe or a flaw in a textile product. This thesis focus on digital image analysis, where images are produced by infrared measurements and aim of this work is to detect defects on these images. Doing defect detection manually is a time consuming process and not an efficient way. So, with the emergence of computer powered systems the process of automating this process has became a field of computational science. For the rest of this thesis, defect detection will be referred as automated way of doing it. Automated defect detection has also became an important application area of Machine Learning and Neural Networks recently. Old conventional computer vision techniques are being replaced by the emergence of the more data dependant machine learning methods [1]. One of the industrial domains of these techniques is defect detection. Automated defect detection is commonly used in sectors like textile, automotive, industrial imaging, material imaging, manufacturing and many others [2] [3]. In this thesis, a problem will be tackled in infrared imaging domain where the aim is to detect defects occurring on a material called Cadmium Zinc Telluride Cd(Zn)Te [†], which is used in radiation detectors. Scientific aim of this research is to detect defects in the radiation detectors by using Neural Networks and improve the performance of the pre-trained architectures by applying Inverse Mathematics methods for the pre-processing step, also to investigate what kind of Neural Networks are suitable for this task. In addition, investigating why some networks working well or not in this problem setting. The motivation of Detection Technology Oy (DT) to support this thesis is to improve the quality and efficiency of future radiation detectors. In order to make this advancement, highly effective semiconductor materials should be used. One of which is called Cadmium (Zinc) Telluride, which is being researched since the 1970's for radiation detectors, but still shows quality and stability issues [4] [5]. One way to iteratively improve the material quality is to examine it for microscopic defects. These defects effect the quality of the charge propagation process inside the detector material and effect the efficiency of

[†]Within the scope of this thesis both CdTe and CdZnTe are considered to be the same material, despite being minor different materials. Cd(Zn)Te is used to depict either, or both materials.

the detection process [6] [7]. The author's motivation is to tackle this problem by using Neural Network models supported by Inverse Mathematics methods and investigate which algorithm would work best for this task. The benchmark model will be Keras implementation of Retina-Net [8] which is one of the most recent Convolutional Neural Network architectures for object detection. Justification of model selection will be discussed. Retina-Net architecture will be trained with our own data first to see how well it fits to our problem. Data used in this research is exclusively measured by Detection Technology and research questions are considered in conjunction with them.

Before stepping into the implementation of the architecture, looking more closer to the problem and proposed solutions are beneficial. Detection and identification of manufacturing defects and defects on texture surfaces can be considered as in the similar category with our task. Finding a flaw on a textile surface is similar to finding a defect in a material in terms of the spatial pixels corresponding to the artifact or non artifact areas, and they have been an interesting research field since in most of the manufacturing lines it is important to produce products as fast as it can be, but with a minimal defect rate [9]. Due to the fact that manual inspection of all the products would take a lot of time and resources, using automated intelligent systems are mostly considered as more efficient solutions. Defect detection can be considered as a branch of object detection/classification. In this thesis, different solutions to defect detection would be first examined and experimented. Then, novel solutions by combining Inverse Mathematics techniques with Neural Networks will be implemented to see if they help to increase the accuracy of the Neural Networks.

In this thesis, defect detection using Neural Networks will be analyzed in different aspects including the effects of the hyperparameters. These are the parameters in a machine learning setting which can be inputted by the humans to change the way how a machine learns. E.g. learning rate, early stopping, regularization parameters etc and determining the number of epochs. An epoch is one pass of a neural network where it sees every training sample at least once and updates the weights -which are the parameters of a Neural Network where the ultimate goal is to find best weights which has the highest generalization power-. It is important to know how many epochs are sufficient to train a network and for comparison of several architectures, thus it will be one of the first investigation points of this thesis. In chapter 2; the background information will be described to closer formulate the research questions formulated in chapter 1.1. In chapter Materials and Methods, the actual implementation details will be shared. In the Results and Discussions section, there will be the display and interpretation of the results.

The images will be used in experiments would be digital Infrared (IR) images. IR imaging uses the density of materials to output a pixel wise image matrix of the scanned

region. Since every material has a different attenuation, each material's absorbance of the IR's will differ. Thus, these differing densities of materials can output an image, which can be used to highlight the differences between the materials and their positions. To mathematically define the Infrared (IR) attenuation along a line, the below formula can be used. I corresponds to intensity. When an IR light hits the object it's intensity is denoted by I_0 and when it's out of the object I_1.

$$I_1 = I_0 e^{-\mu s} \tag{1.1}$$

where μ is the attenuation coefficient, s is the distance traveled by the IR light inside the material substance. Pixel detectors detect these output and each output of an IR light corresponds to a pixel in the image matrix, in this experiment ending up with 1280 x 1024 pixels for each image. This is how the digital images which will be used in this experiment is formed. An example image containing defects is shown below in Figure 1.1:

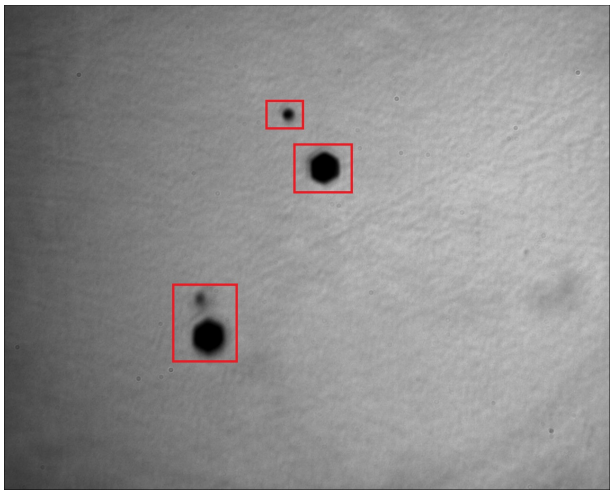

Figure 1.1: Example Image of a defect in the detectors that the research of this thesis aims to find

A general mathematical formulation and structure of a Neural Network's single node can be formulated in the following figure and equation consecutively. A network has many of these connected to each other:

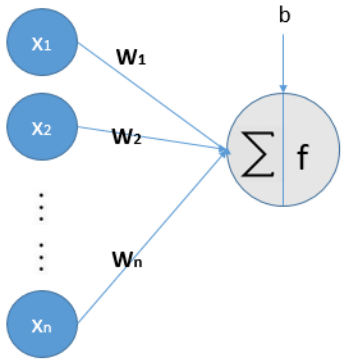

Figure 1.2: Example node of a Neural Network where $x_1 ... x_n$ corresponds to inputs to the node [10]

Figure 1.2 shows one example neuron in a feed forward Neural Network and usually in a Neural Network there are multiple nodes and layers (number of layers determine the depth of the neural network, each layer consists of certain number of nodes. If number of layers are 5 this means 1 input layer, 1 output layer and 3 hidden layers, each containing particular amount of nodes, nodes such that in Figure 1.2) . Weight $(w1, w2, ..w_n)$ and bias (bn) matrices are used to represent each layer and node mathematically. Mathematical formulation of a node can be seen in the formula below:

$$f(b + \sum_{i=1}^{n} x_i w_i), \tag{1.2}$$

where b is the bias term, w denotes the weights, x denotes the inputs, n is the number of inputs from the incoming layer and f is the activation function. Each input of a node is multiplied by the corresponding weights (e.g. $x_1 w_1$), which are summed together. Then, bias term is added to this sum. Scalar output is then passed through the activation function and activation function outputs a final scalar value. An activation function maps the output value of a node to introduce non-linearity, it will be examined in more detail in the upcoming sections.

Formulation of a neural network node is explained above. A neural network usually consists of many layers and each layer consists of many nodes. One simple example of it can be seen in figure below, more complex network architectures will be discussed throughout the thesis.

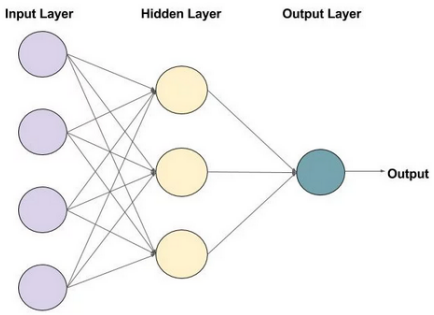

Figure 1.3: A simple Feed Forward Neural Network [11]

A network has an input layer, output layer and hidden layer(s) in a simple Neural Network design. One example is shown in Figure 1.3 and that kind of networks are referred as feed forward Neural Networks. Number of nodes and hidden layers can vary in different NN architectures and based on the task of the network.

1.1 Problem Formulation

In this thesis, the main practical goal is to examine some different aspects of Neural Networks in our problem setting and how different parameters effect the outcomes with the ultimate goal to improve the defect detection performance.

1.2 Research Task and Questions

The main research questions that are focused on this work are summarized below.

- What network architecture and which features, if any, lead to the best defect detection performance?
- How many epochs are enough for the given task on Keras Retina Net?
- Understand why longer epochs gives less accuracy on Keras Retina Net
- Train a model
  - Architecture Selection and Justification
  - Wavelet Transform Supported CNN
  - Retina Net with different HyperParameters
  - Other methods
- Can Inverse Mathematics help us to improve CNNs learning rate in this domain?

- Do manually crafted features help CNNs to learn better in this setting?

In the upcoming sections it will be discussed, how answers for these questions can be found systematically.

1.3 Improving Convolutional Neural Networks by Inverse Mathematics

Inverse Problems research group led by Professor Samuli Siltanen at the University of Helsinki [12] has a Deep Learning sub-group, which is using combinations of Neural Networks and Inverse Mathematic techniques in a wide array of applications. Mostly however, in the X-Ray imaging domain [13]. Their research inspired this thesis about how Inverse Mathematics can be used in conjunction with Neural Networks to boost the performance of the learning algorithms.

First, to start with the definition of a feature in a machine learning setting, which is a term referred constantly throughout in this thesis, it can be thought as the descriptive information extracted from the input data in order to make the approximation function between the input and output more easy to learn. In other words, it is the measurable properties of input data. A feature vector is the vector containing these descriptive characteristics of the data. These features can be extracted in many different ways which some of the examples will be discussed in this thesis.

When working with CNNs it has been mentioned in above section, why it performs better than the other approaches such that it does automatic feature extraction unlike the other conventional Machine Learning methods which rely on manually crafted features from the data. However, some studies suggest that even though CNNs are good at doing automatic feature extraction, sometimes there is a room for improvement. In a paper published by Brebisson et al. [14], it can be seen that when the date feature is fed to the Neural Network (NN) formatted like day, month and year results in better learning (this will be revisited in chapter 3.4.1 Total Variation) compared to giving it to NN unformulated, which is depicted as the "wolf case" in another paper [15]. In the paper, there is a Convolutional Neural Network classified dogs and wolfs based on the background rather than the animals themselves. In photos containing wolf, background with snow was the distinguishing feature whereas the dog photos had green terrain as the distinguishing feature rather than the animals themselves. This shows that it is not enough that Neural Networks learns well, but how it learns is another important question, which it's importance is also supported by another study [16]. This shows us even though CNNs are very good at extracting features by themselves, if they are fed with hand crafted features they might even perform better [16]. This is where Inverse

Mathematics comes in to the equation. It has tools, which has a potential to help the NNs with the feature extraction and pre-processing of the data. Some of these tools which will be used in this thesis are TV denoising, Wavelet Transformations and other methods. Can popularly used Inverse Mathematics methods such as Total Variation Denoising or Wavelet Transformation of the input image, prior to feeding it to the Neural Network help the network to learn better? This will be one of the investigation points of this thesis.

It is an important research question that if it's better to feed Convolutional Neural Networks with extracted features, raw image or both raw image and extracted features concatenated. Many NN researchers tried to apply different input images to the neural network to see the effect of different inputs. What input will make CNN to perform better? Some studies show that image transformation (of the input image) can make Neural Networks more robust against adversarial examples [17]. Some shows that using pure images as inputs would make the neural network to perform better [18] and some suggest that original image concatenated with a transformed version of it might possibly enhance the performance of the learning algorithm [19].

More details about what kind of a Neural Network architecture should be initially chosen, how Inverse Mathematics can be used to improve the Neural Network's performance and more information about the used methods such as Wavelet Transformations and TV Denoising will be explained in more depth in the Chapter 3. Before they will be examined, a background information about pre-Neural Network era will be provided in the upcoming Background chapter to give a look at the bigger picture before focusing deeper onto the Neural Networks.

2. Background

2.1 Background and history of algorithmic methods for image classification - Pre Deep Learning Era

Before the Deep Learning methods got widespread and started to be commonly used, there were other methods predominantly used for image classification and object detection tasks. The main task was similar, extracting some feature descriptors from the data and use a classifier to evaluate the nature of the problem. However, the biggest difference between the older methods and the recent Deep Learning powered methods were that in the conventional algorithms the feature descriptors are manually extracted and crafted. In addition, their features are also more ridged and difficult to adopt to new problems. Next, some of the conventional algorithms in the pre-deep learning era, which became important milestones of the computer vision will be examined.

2.1.1 Overview of Viola and Jones algorithm for face detection

Viola and Jones [20] published a paper in 2001, which proved to be state of the art face detection algorithm by the time. It is a haar feature based simple algorithm, yet quite powerful. Haar like features are the basis of very trivial computer vision algorithms which are created by dividing a rectangle into various different black and white parts. The idea of using Haar like features originated from Haar Wavelets and adopted to computer vision feature extraction tasks by Viola and Jones. It has a long training time because of the ensemble approach it uses however prediction is fast. Basically, it is a combination of several weak classifiers which are combined at the end. Some of the common haar classifiers were consisted of haar like features below, some consisting of rectangles which are used to find out if there is a face in the image [20]. The Viola and Jones's paper described how to use simple features in a boosted cascade setting to quickly apply it to the test set.

A simple example can be seen in figure below. For instance, the second feature in

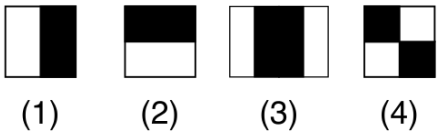

Figure 2.1: Common Haar Features [21]

Figure 2.1 can be used to detect the eyes on an image by sliding throughout the image or the third one can be used to detect the nose. An example of this can be seen more clearly in Figure 2.2. In Figure 2.2, Haar-like features are applied to eye parts of an human face within a sliding window, where it then detects the location of the eyes in the face.

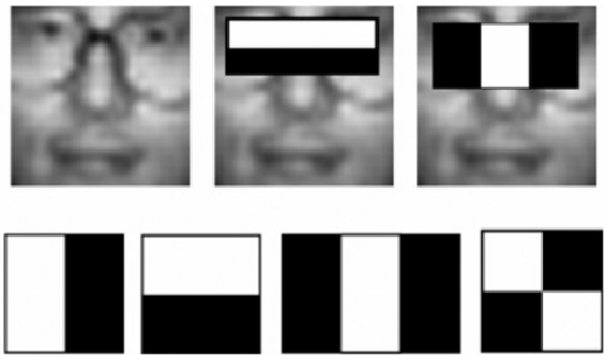

Figure 2.2: Example Haar Image, with detection of features like noise or eyes [22]

2.1.2 Overview of Histogram of Oriented Gradients (HOG)

As the computer vision algorithms were proceeding, an important milestone was the discovery and usage of Histogram of Oriented Gradients in 2005 which brought improvements on object detection [23].

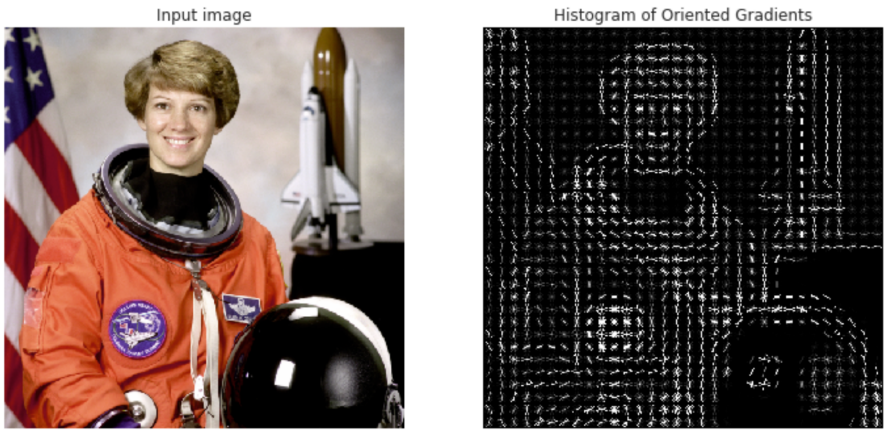

Figure 2.3: Overview of Histogram of Oriented Gradients (HOG) [24].

HOG uses gradients (derivatives) to calculate the gradient vectors of an image, which is a simplified version of it. This gradient vectors extracted from an image provides important, descriptive information about specific features of the image. As it can be seen in the Figure 2.3, edges are calculated based on each pixel's x and y coordinate gradients. Based on the intensity of the gradients, directions of the edges are calculated. Gradients highlights the information about edge and corners since their values increases when it detects one. The image on the right is the extracted features from the original image on the left. The extracted features then would be used in a classifier or saved to used as an encoding of the original image, which then can be used for image processing tasks like object detection.

2.1.3 Overview of Scale Invariant Feature Transform (SIFT)

Scale-invariant feature transform (SIFT) is another important conventional computer vision technique, which had a big impact on the computer vision history. SIFT can be thought of as accomplishing some modern features of Deep Learning when it comes to being invariant to positional changes of the input image. Like Convolutional Neural Networks the main reason SIFT was a big progression is that it made the learning system robust to scale, rotation and small changes in perspective.

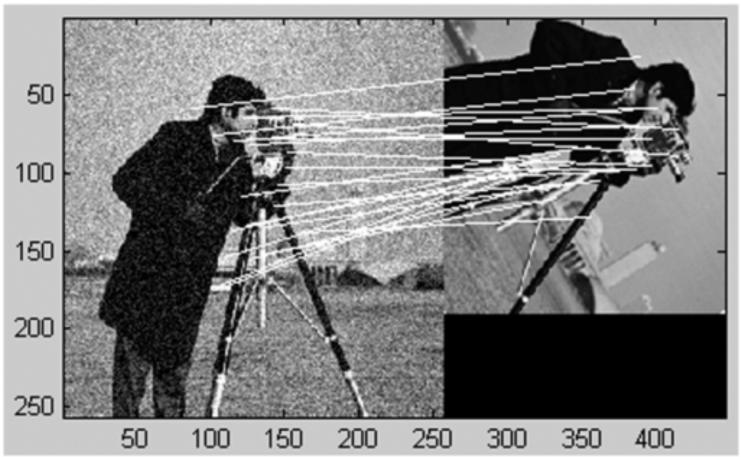

Figure 2.4: Illustration of SIFT

SIFT is a feature extractor, which works by ensuring invariances using an initial scale spacing. It defines key points and descriptors which ends up extracting the SIFT features. For each key point, an orientation is calculated which makes the extracted features orientation invariant [25].

As it can be seen in Figure 2.4, SIFT features extracted from the photographer on the left can be matched with the rotated and cropped image on the right. This illustration shows the importance of SIFT because it tackles a major challenge of

computer vision algorithms, which is being invariant to change of the same object in altered images, smoothly. Maintaining invariance is still an important aspect of modern computer vision algorithms, even though CNNs tend to perform well on ensuring invariance, applying data augmentation before running the model is a common practice to ensure invariance to the object location and position in machine learning [26]. SIFT is an important milestone in computer vision, especially in image matching and object detection tasks.

2.1.4 Support Vector Machines on Image Classification

When the times before we reach to peak of Deep Learning domination Era is considered, machine learning methods were the proceeders of them and are still used actively. Usage of machine learning methods such as Support Vector Machines, Logistic Regression and Decision Trees were quite often and still being used in computer vision tasks actively. They can be seen as a step between the conventional algorithms to data dependant deep learning based methods. Among these conventional machine learning methods for image classification, Support Vector Machines were the one of the most popular ones [27].

Support Vector Machines (SVM) fall into the supervising learning algorithms category in machine learning. Supervising learning refers to learning from data where every data point has corresponding labels. In the figure below, it can be examined how SVMs work. It is a binary case where the output can be either 0 or 1 -green or red in Figure 2.5-. Support points, one for each class in this case, are the closest points to the other class. After support vectors are defined, the decision boundary is drawn based on maximizing the margin between the support vectors as shown with the bold blue line between the support vectors [23].

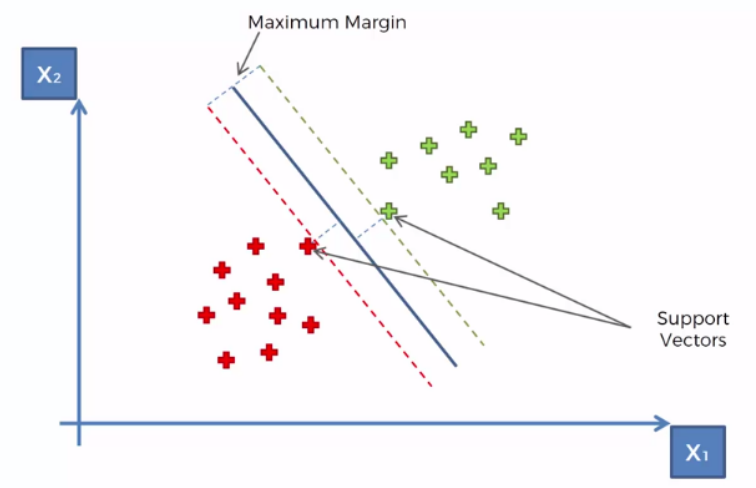

Figure 2.5: Illustration of SVM [28]

Mathematical formulation of a linearly separable classification SVM's optimization problem can be formulated as follows:

$$\min_{w,b} 1/2||w||^2,$$

$$\text{subject to } y\_i(w.x + b) - 1 \geq 0, \qquad i = 1...m$$

where y_i represents the linearly separable ground truth classes for each input(x_i,y_i) belongs (e.g. $y_i = +1$ or $y_i =$ -1), w (vector) and b are the weights and biases of the hyperplane (classifier) to be predicted, M is the geometric margin of the dataset and m is the number of samples [29].

Before moving into the image processing with Deep Learning methods, taking a look at the complete picture from extraction of the features to classifiers would be beneficial. SVM is good for classification after the features are extracted with some other feature extractor algorithm beforehand such as HOG, Haar like features or SIFT features mentioned above. Extracted features from a feature extractor algorithm e.g. HOG, can then be fed to the SVM learning algorithm to apply the classification and learn from these features. To give a complete picture of a machine learning classification framework below graph can be examined 2.6. The input image is pre-processed, then the features are extracted by using a feature extractor such as Haar or HOG. After that, a classifier (learning algorithm) is applied to learn from the extracted features, which can be thought of an SVM and at the end, in the label assignment, part prediction output is produced.

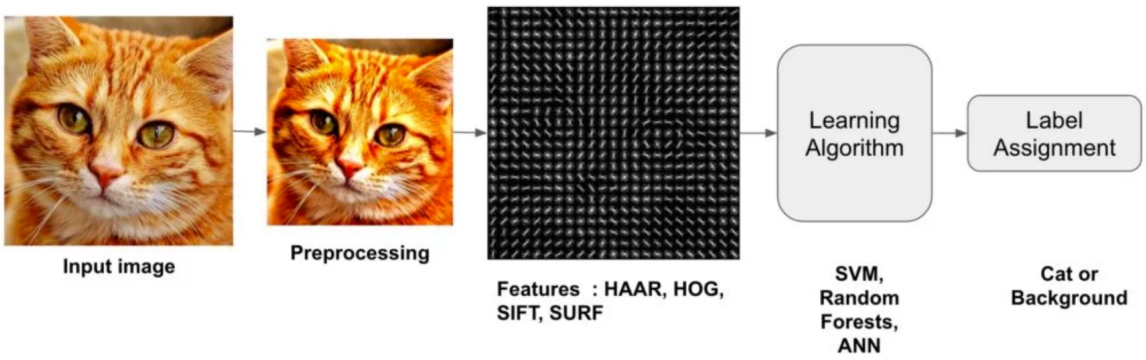

Figure 2.6: Complete picture of a NON Deep Learning Classifier [30]

2.1.5 Neural Networks and Defect Detection

Deep Learning based methods has shown a great performance on defect detection in the last decade. Especially, after the possibility of large amounts of data storage and increase in the computational power, practical usage and the importance of the Neural Networks got boosted in image recognition domain, as well as defect detection tasks. There has been various different approaches on using CNNs and referencing models, mostly training an existing Neural Network architecture such as Alex-Net, VGNet, ResNet etc. with task specific data [31]. Comparison between the model based approaches and Neural Network based approaches have been examined in various studies. Neural Networks outperformed the model based approaches mostly in all the cases [32].

The main reason Neural Networks doing better than other approaches in image recognition and defect detection tasks is that, first, existing approaches used before Neural Networks needed hand crafted feature extraction, whereas one of the most strong aspects of Neural Networks is that they are able to do the feature extraction by themselves. They do it by learning from the data and finding the features, features that are the most important (descriptive) when it comes to distinguishing the objects from background or defects from non-defects.

In regards to defect detection and more generically image classification tasks, one type of neural network is dominating the field which is the Convolutional Neural Networks (CNNs). It is important to understand why they work better and what makes them superior to the other approaches. Their structure will be examined in more detail in the upcoming section, but the main reason they are doing better than other algorithmic methods (and traditional Machine Learning methods) is that they can leverage the spatial information better [33], hence does better representation of the inputs. In algorithmic approaches, where images are examined on a pixel level, spatial information between pixels might be ignored, however by the usage of convolution kernels and downsampling of adjacent pixels, CNNs make use of this spatial information. Another thing is; in algorithmic approaches, features are tried be extracted manually and then these features are often used to feed a classification algorithm like SVMs, which requires special feature engineering for different tasks, since optimal features for one image classification task might differ from another. Here, where CNNs comes into the picture. Because of their ability to extract features automatically, CNNs does considerably better [32] than algorithmic approaches and traditional Machine Learning methods.

Before going more deeper into the CNN architectures, let's briefly define how a convolution operation takes place in a Neural Network setting. Convolutions filters or

kernels, have a shape of Height (H), Width (W) and Depth (D). These are (indeed) tuneable, but common practice is to use 5x5 or 3x3 of convolution kernels same with the backbone network used in our experiments ResNet [8], where the depth is equal to the input's depth (d_input). In each layer, convolution kernels are applied to the input and dot products for the corresponding regions are calculated. The output of this operation is referred as a feature map. How to work with convolutional kernels in Keras environment, which is used in our experiments, is explained in more detail here [34]. Mathematical formulation of convolution is described below:

$$(p * f)_j = \sum_{l=-\infty}^{\infty} p_l f_{j-l} \tag{2.1}$$

where * is the convolution operation symbol, p is the point spread function which is also referred as the convolution filter, f is the original signal (function) to be convoluted with point spread function, j denotes a point in time, and l defines the size the convolution filter, it can be thought of a window size where convolution operation will take place [35].

Mathematical formulation of a fully connected Neural Network is explained in the introduction chapter. A Convolutional Neural Network follows the same structure by adding additional layers on top of the fully connected ones, a convolution layer followed by a pooling layer. A pooling layer is basically a downsampling layer. It reduces the spatial dimensions of the feature maps, which also means reducing the number of weights (parameters), thus increasing the computational efficiency of the network. It can be thought as summarizing the information in a feature map into a smaller feature map. Output of a convolution filter applied to the input image is called the feature map. These feature maps are calculated based on this formula for a 2D image:

$$(p * f)[m, n] = \sum_j \sum_k h[j, k] f[m - 1, n - k], \tag{2.2}$$

where f corresponds to the input image, p is the convolution filter (kernel) and m denotes the index of the rows and n denotes the index of columns [36]. Applying this formula to an image would look like the calculation in the figure below (2.7).

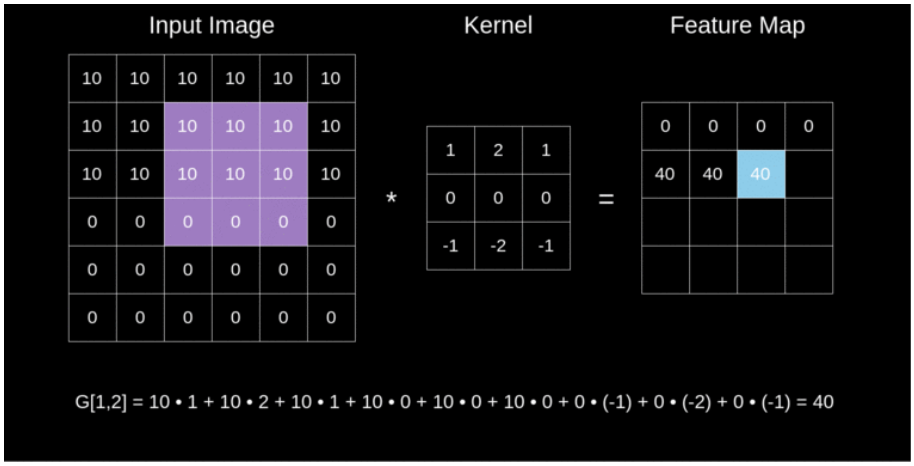

Figure 2.7: Example of how a feature map is extracted by convolving a kernel with an image [36]

A good research paper, which shows how CNNs are successfully utilized in a defect detection problem is explained here [31], where the defects are in a manufacturing setting. Images used are produced from X-Ray measurements. The network used is a version of ResNet, ResNet-101 where number "101" indicating the depth of the network. Results of the defect detection network (without segmentation) indicates a successful detection accuracy with 0.931 accuracy rate. In another paper, authors used different CNN configurations, comparing deep and shallow CNN approaches to detect the defects on textured surfaces of greyscaled wood images [37]. They used LesNet to test the shallow network's (basic CNN) performance and a VGG variant (VGG19) to test the deep network's performance. Their work proved, once again how CNNs are successful on these defect detection tasks with both networks performing above %90 accuracy, where the deep CNN with %99 accuracy is further outperformed the shallow one %91. In this work, CNNs will be used in defect detection in a similar manner, but this time, also trying to enhance the learning performance by using different inputs created by Inverse Mathematic techniques.

2.2 Overview and comparison of Common Convolutional Neural Network Architectures

Nowadays, there are many different CNN architectures used in object detection and classification tasks. In this section, two early Convolutional Neural Network architectures will be explained, which are the ones started the CNN publicity and brought more attention to them. LeNet-5 (1998) and Alexnet (2012) can be considered as pioneers of this era. In addition, ResNet (2015), which will be used in our experiments as the backbone network, will be explained.

2.2.1 LeNet-5(1998)

LeNet-5 [38] is a very trivial CNN architecture compared to the ones currently in use today, however it doesn't change the fact that it is the building blocks and basis of the CNN architectures used nowadays. The idea of stacking convolution layers one another and usage of pooling layers after convolution layers are still mainly used. A pooling layer is where the feature map output of convolution filters are downsampled. Also, using fully connected layers for the output, after the convolution layers is a common approach in modern Neural Network architectures.

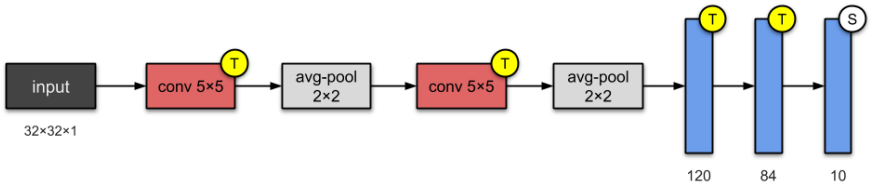

Figure 2.8: LeNet-5 Architecture
[38] [39]

In Figure 2.8, LeNet-5 architecture is shown. Even looking from the Figure it can be seen that it is trivial compared to the modern CNNs yet similar stacking of convolution and pooling layers are still widely used. LeNet can be thought as the basis of standard CNN architectures.

2.2.2 Alex-net(2012)

Alex Net can be considered as the actual attention bringer to the CNNs by winning the 2012 ImageNet challenge with a clear gap to the 2nd best performer. In 2012, "ImageNet Classification with Deep Convolutional Neural Networks" article is published [18]. After the release of this paper, it is proven that this CNN architecture achieved

better accuracy scores compared to the previous state of the art models and CNNs have much more potential which could be utilized. It won the 2012 ImageNet challenge.

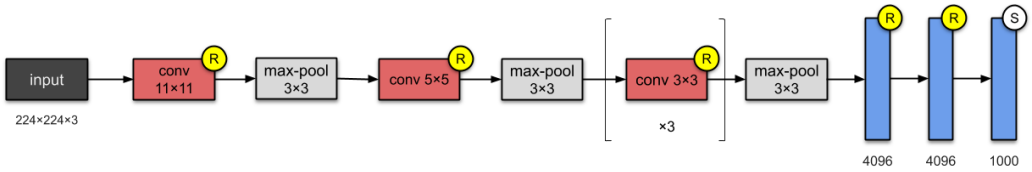

Figure 2.9: AlexNet-5 Architecture
[18] [39]

Basically, it is an extension of the previously mentioned LeNet-5 architecture. It has 5 convolution and 3 fully connected layers. Increased computational capacity within years enabled the AlexNet authors to add more layers on top of the LeNet-5 architecture. It has $\approx$ 60M parameters which is mentioned as the one of the biggest parameter containing CNNs at that time, by the authors of the article [18]. One of the novelties it brought to the CNN research is that making the usage of ReLU as an activation function (which will be explained in detail) for the first time. Another big contribution, which is commonly used in the modern architectures now, is the usage of drop-out layers. Drop out in deep learning setting can be defined as a regularization technique, which randomly ignores (sets to zero) some nodes in the corresponding layer. This helps improving generalization power and reducing overfitting by regularizing the network. Basic idea behind drop out regularization is that, randomly ignoring some nodes in a layer in each epoch will prevent the network from being over-dependant to a one particular node. To deal with overfitting drop-out layers has introduced and used in the paper which showed a successful performance.

2.2.3 ResNet-50

As the previous two architectures suggests, the popular idea in the Neural Network architectures were that the more layers are better and if the number of layers is increased in the network, then the generalization power should be also increased. In theory, it was true but because of a phenomenon called vanishing gradients, this assumption did not hold. As the network depth increased, it ended up in saturation in the learning and decrease in the accuracy. So, stacking up more layers is better but it doesn't work that well all the time. Vanishing gradient problem is the reason of this saturation, because as the network gets deeper, the gradients become less sensitive to the updates in the parameters than in the early layers, which ends up in a state where update of the parameters not effecting the final output enough. One way to tackle this problem is

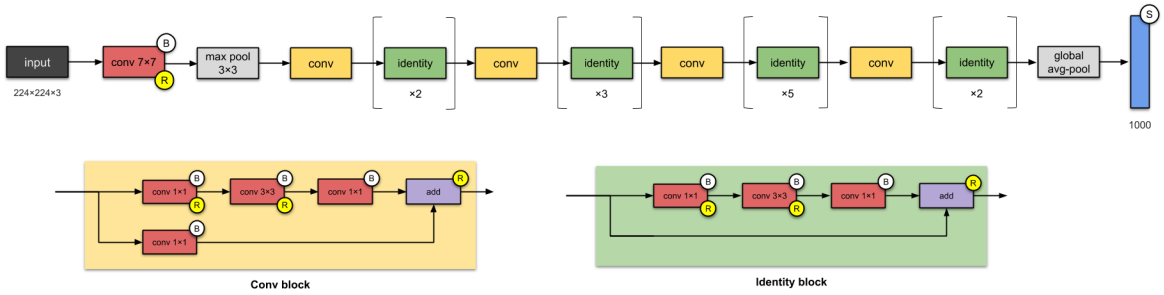

Figure 2.10: ResNet-50 Architecture
[40] [39]

using skip connections between two not adjacent layers so that the earlier information coming from previous layers wouldn't be vanished [40]. Res-Net made this idea possible and allowed us to work with deeper networks, yet not having a vanishing gradient issue where it would take away all the benefits of using a larger network. ResNet solves this by using identity matrices to move the information from earlier layers to latter layers which can be seen in the Figure 2.10, in identity block sub-figure. An identity block merges the previous convolution layer's information with the currently calculated ones. By using these skip or residual connections this architecture allowed us to use more layers in the network, thus make the networks deeper without saturating the accuracy.

There are two activation functions used in ResNet architecture, Sigmoid and ReLU (Rectified Linear Unit). They are shown in Figure 2.11. It is mentioned before that an activation function maps the output value of a node to another scalar by introducing non-linearity. Sigmoid and ReLU are the most commonly used non-linear activation functions in Neural Networks [41]. In Figure 2.11, it can be seen that Sigmoid function takes values between 0 and 1, whereas ReLU takes values between 0 and infinity. Sigmoid is mostly used in the output layers where we want to have a distinct probability score or a probability distribution which sums up to 1, whereas ReLU is used mostly after each convolutional (or pooling) layer. In ReLU, all the negative values are mapped to 0 and all the positive values are mapped to itself. It is still a research topic why ReLU works that well but one of the reasons of ReLU's success is that, it is mathematically simple which makes it computationally efficient, yet still able to approximate non-linear relationships quite well. Activation functions in a Neural Network context will be revisited in FPN architecture section in more detail.

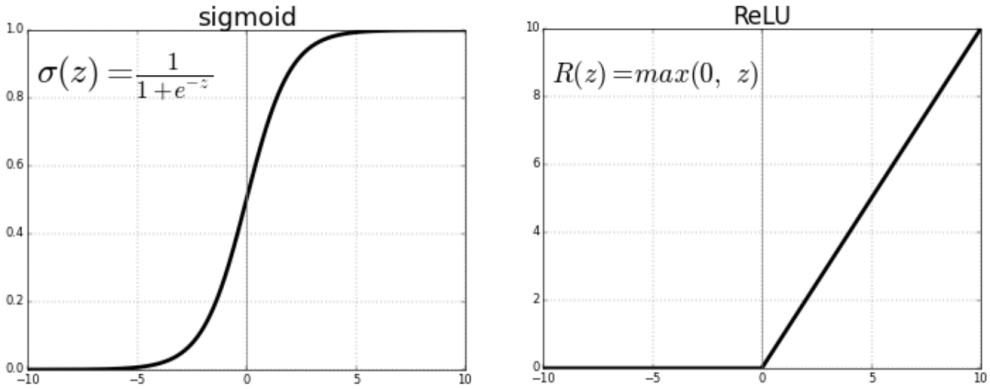

Figure 2.11: Sigmoid and ReLU Function
[42]

3. Materials and Methods

3.1 Data

The data set for this thesis consisted of 10.000 training images. Each image has corresponding defect coordinates and shape as labels. An example image looks like in Figure 3.1:

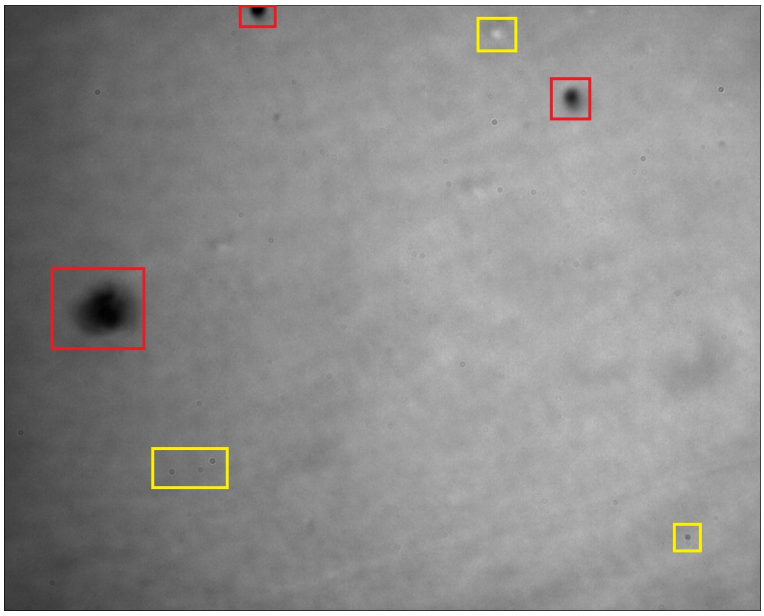

Figure 3.1: Example sample image with defects in red and non-defects in yellow

What can be seen from Figure 3.1 is that a sample IR image of Cadmium Zinc Telluride Cd(Zn)Te. There are 3 defects marked in red. Our aim is to detect them while not detecting the yellow ones which are coming from either the measurement device, defects from another image layer or the background dirt and dust. Another challenge when trying to detect these defects is, as will be explained in more detailed in the upcoming section, when the network is tried to be optimized to detect both the large (on the left) and the smaller defects (on the top) the accuracy of detecting the small defects underperforms the detection accuracy of the large defects.

As it can be seen in several examples below, the nature of the defects can vary.

There are 10 categories that roughly classify each type of defect such as:

- round single
- round double
- unclear single
- unclear double
- hexagonal single
- hexagonal double
- square single
- trigonal single
- void single
- bubbles single

Some examples can be seen below. There are 2 main goals. First, detect and count the number of defects in a given image and secondly classify the defects based on their shape. For this purpose, Neural Networks are used to train our models. Data is divided into training and validation sets. Models are trained and evaluations are conducted. For the computing environments Ukko2 [43] and NVidia P4000 GPU from the Inverse Problems Research group is used [44].

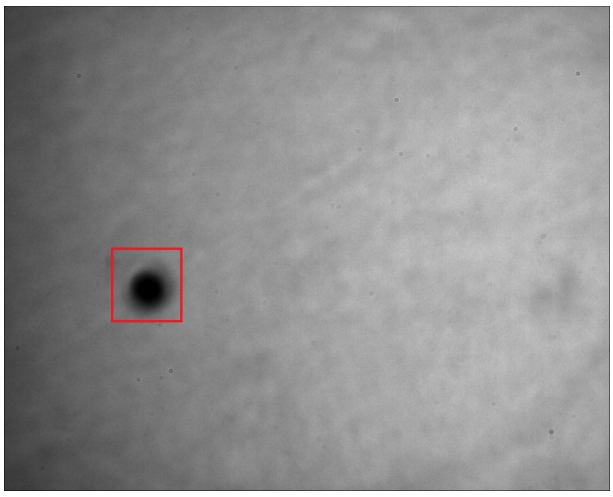

Figure 3.2: round single

Figure 3.4 is one of the examples, which makes the detection task harder because the defect is not clear, somehow blurry and hard to distinguish from the background image. Also there is a risk of the yellow marked background shape can be interpreted as a defect.

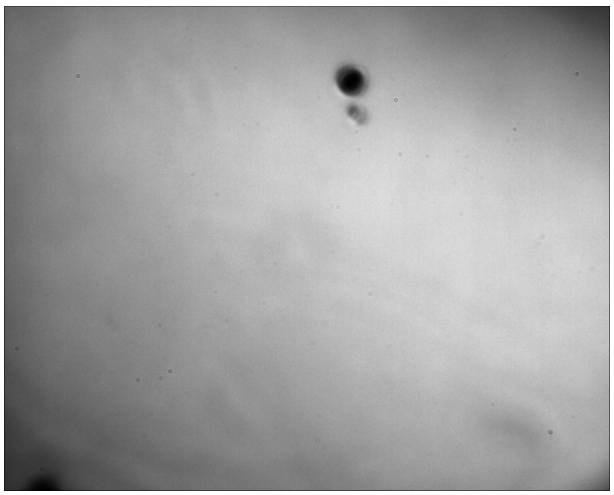

Figure 3.3: Round double defect example

Figure 3.4: Unclear single defect example

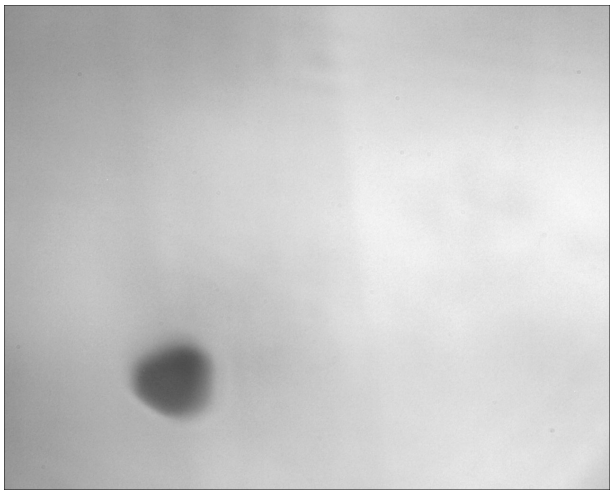

Figure 3.5: Hexagonal single defect example

3.2 Keras Retina-Net as Benchmark

Retina-Net network architecture is one of the most popular CNN architectures for object detection in tensorflow backend (tensorflow is a commonly used open-source machine learning library developed by Google [45]). Among the state of the art CNN architectures like Yolo [46], Faster-R-CNN [47] and Retina Net [8], Retina-Net performs better in most of the classification tasks among other single stage detectors by leveraging the focal loss [48] which is explained in section 3.2.1. It is implemented based on a paper called Focal Loss for Dense Object Detection [8].

When it comes to object detection using Convolutional Neural Networks (CNNs), there are 2 main approaches exist. One stage detectors and two stage detectors. A one stage detector means it only makes a single pass to extract the features and make the prediction, whereas two stage approach consists of 2 different passes on the image. In the first pass, it extracts the areas of interest where potentially an object is present, then for each region that is extracted, CNN is applied to those regions for classification. In contrary, in one stage approach classification and detection is done within a single pass. This brings a trade off between the accuracy and performance, performance in terms of speed. Generally, two stage approaches result in better accuracy and one stage approach results in better performance. Here comes the Retina-Net which uses the one stage approach but still able to beat some state of the art 2 stage approach detectors, such as plain Feature Pyramid Networks (FPN) and Mask R-CNN which is shown in the original paper evaluated on COCO dataset [8].

In the paper, the reason for one stage detectors to not achieve as high accuracy levels as the two stage detectors is explained as the class imbalance problem, which Retina-Net proposes to solve by introducing a unique loss function. This loss function is called the focal loss. Before going deeper in the focal loss -which is a highly distinguishing feature of the architecture we use (Retina-Net)- first, in order to understand the network better we need to focus on FPN type networks more in detail which Retina-Net can be considered as an improvement on top of them.

The need emerged for FPN based architectures, because of the size mismatch problems when trying to detect objects in different sizes especially for the small ones. For instance, when a task requires of detecting a different sized objects with the same network, such as a tennis ball and a human or a big defect on a surface and a very small one, CNNs were experiencing problems with these settings. Feature Pyramid Network architecture is designed to bring a solution to this problem.

Initially pyramids can be constructed in 3 different ways [49]. Most trivially, different sizes of image can be used in the pyramid. For example, the bottom image would be the original, higher layers would be the scaled versions of each previous

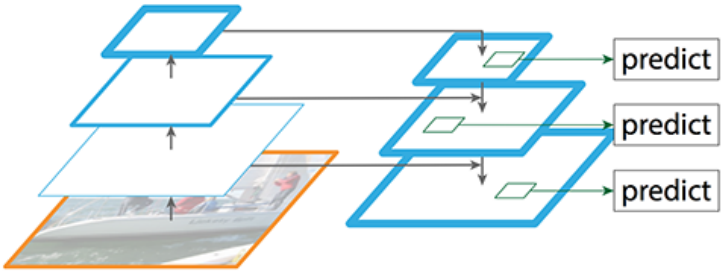

Figure 3.6: Feature Pyramid Network [49] architecture example

layer, but this would consume a lot of computational resources, hence it is not very feasible. Another approach is using the feature maps of each layer to do predictions, but in this case there will be a lot of generic information coming from the low level layers, which would effect the accuracy negatively. Third and the state of the art approach uses some feature extractor layers from another network (e.g. ResNet as a feature extractor), consisting of several convolution layers, max pooling and activation functions and combine them in the upsampling layers with the original image. As it can be seen in the Figure 3.6, 2 different feature maps are merged, which makes the feature extraction process more powerful and faster. To give a brief explanation of convolution layers, max pooling and activation function in this context, convolution layers can be thought as the layers where relationship between the pixels are extracted with a filter (kernel) applied to the input image. Activation function is an essential part of the Neural Network architectures, which conventionally applied right after the convolution layer. The main purpose of using activation function is it decides if a node on a network would get activated or not based on the input values it receives from the output of the convolution layer. It has many different types, but the most commonly used ones are Sigmoid and ReLU which are also the ones used in Retina Net. Max pooling layer can be thought of a downsampling layer, applied after the activation layer in this context, which decreases the size of the input by either averaging or taking the maximum of the adjacent pixels.

When it comes to our problem setting with the defects, it would be fair to ask this question, why would an FPN architecture be more useful than a Single Stage Detector (SSD), which also leverages the usage of pyramid networks? In Figure 3.7, there is a VGG-16 architecture, which is a Single Stage Detector. It can be seen that Single Stage Detectors, in this case VGG-16, only uses the last several feature maps when the classification starts (classification layers begins with the latest box that is marked by red). If the inputs the classification layers get are examined, which are depicted with arrows pointing to the red box, only feature maps extracted in recent layers are fed into the classification layer. Basically, the network applies convolution and max

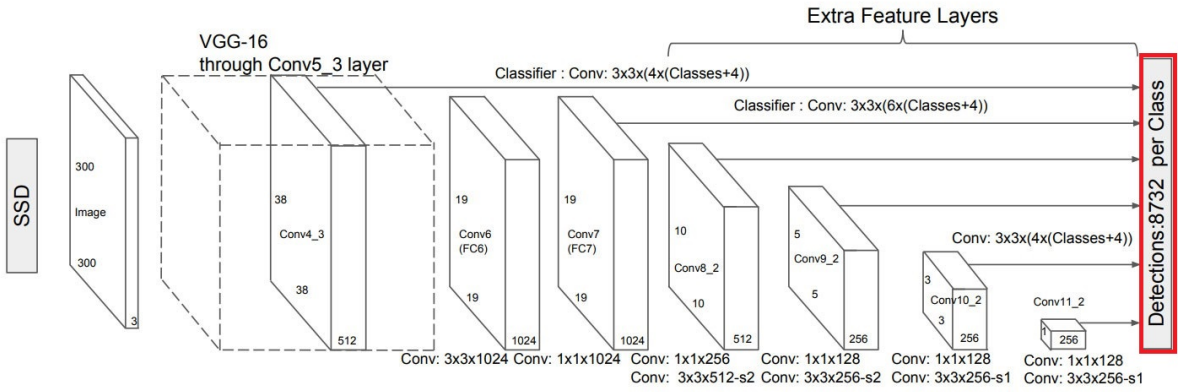

Figure 3.7: An example of Single Shot Detector Network - VGG-16 architecture [50]

pooling for layers from the start, but uses only last couple of layers to initiate the classification. Classification layer in this context corresponds to the fully connected layers, after convolutional layers extracted the features from the input image. This brings a problem when it comes to detecting smaller objects. Information about smaller objects might be lost and not present in high level features. However, if FPNs are used instead, it will be still possibles to leverage the information on the low level features by combining them with the high level features, hence it would increase the probability of accurate detections of the smaller objects. This is important in our problem setting, since in our defect detection case we indeed deal with different scaled and sized artifacts, which usually contains very small ones as well. It can be seen in the Figure 3.7 that pyramid hierarchy starts after conv4_3 layer of VGG, which skips the higher resolution information (feature maps) in the earlier layers. Feature Pyramid Networks for Object Detection article [49] proves that these features are important when it comes to detecting smaller objects, thus using FPN's rather than SSD's would be a better choice in our setting.

The architecture of Retina-Net consists of convolution layers instead of fully connected layers. It is a single stage detector, which consists of 3 different subnetworks. First one is the backbone, which is ResNet51 in our case on top of a Feature Pyramid Network. Second network is the classification subnetwork and third is the bounding box regression network, which runs in parallel and uses the output of the first network.

ResNet (2.3.3), which has been explained previously, now will be used as our feature extractor or put in other words, the backbone network. Here the key point is to understand how Retina-Net combines the low level feature maps with high level feature maps. There are 2 pathways as it can be seen in Figure 3.8. First, bottom to top pathway can be seen in 3.8a, it can be observed that ResNet extracting convolutional feature maps in different resolutions, where the higher the layer it goes the resolution

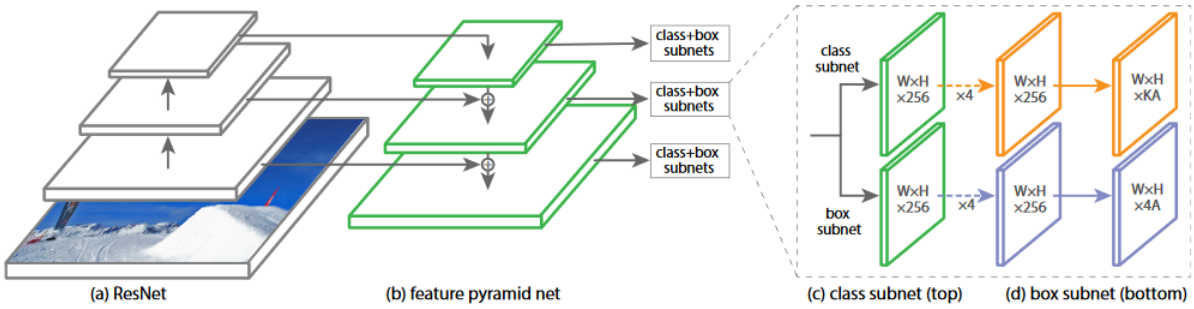

Figure 3.8: Retina Net architecture

[8]

drops and semantic information goes up. After the feature maps are extracted, we go to the top down path, which can be seen in Figure 3.8b. This time the usage of Feature Pyramids starts. Here, low resolution feature maps are merged with the higher resolution feature maps. What actually happens in the merging is that the last layer in the "a" part of Figure 3.8 is up-sampled to match the dimensions of the second last layer. Then, they are added together to form a new feature map, which can be seen in the top layer of Figure 3.8b where arrows depict the merged feature maps. This process goes on until there is no feature map to be matched. Each of the merged feature maps then produces an output independently, which will be used as an input for classification and regression sub networks [8].

At the end of the first subnetwork (subnet), feature map outputs for each pyramid layer will be calculated. These feature maps would be used to create the anchor boxes (9 directions for each feature map) and then for each feature map, classification and regression subnet would run in parallel [8]. An anchor box can be thought of a candidate bounding box with a probability of containing some target object, which predictions will be run for.

Classification subnet consists of Fully Convolutional Layers. Convolution layers has 3x3 dimensions. They use both ReLU and Sigmoid activations after the convolutions. They output a feature map at the end, which has the dimensions of (W,H,CxAB) where W is width, H is height, C is the number of classes and AB denotes anchor boxes, which means at the end, class probabilities for the objects in each anchor box would be extracted.

Regression subnet is also consist of Fully Convolutional Layers. It's design is same with the classification subnet and runs parallel with it. Only difference is the output would have a different dimension since it supposed to output a bounding box coordinates instead of the object classes, hence the output becomes 4 times anchor

boxes (W, H, 4 x AB) .

3.2.1 Loss Function

Another important aspect of the Retina Net is the loss function it uses. A loss function is one of the most important part of a Neural Network. After each forward pass of a network a prediction is outputted. In a supervised learning setting this output is called prediction and the prediction is compared with the ground truth label to calculate the difference between them (which is referred as the loss value) in the loss function. A loss function evaluates the difference between the prediction and the ground truth, then updates the neural network's weights based on this loss. Updating of the parameters is done with an optimization algorithm called stochastic gradient descent. Stochastic gradient descent uses only 1 sample for each iteration to calculate loss which means setting the batch size * parameter to one in a Neural Network context where a sample is selected randomly from the training set for each iteration. More information about Stochastic gradient descent can be found here [51] which exceeds the scope of this thesis. Retina Net uses an exotic loss function called Focal Loss [8]. It uses a loss function which adds the regression loss and classification loss together. Regression loss is the loss calculated for the location of the predicted bounding box and classification loss is the loss calculated for the category of the detected defect. Focal loss can be formulated as below:

$$FL(\rho t) = -(1 - \rho t)^{\gamma} log(\rho t) \tag{3.1}$$

where FL stands for Focal Loss, γ is the focusing parameter which adjusts the decrease level of the easy example's effect on weight updates and is greater than zero. $(1 - \rho t)$ is the modulating factor where ρt is the class probabilities (e.g. background and object1 for the binary case) for each anchor box predicted by the model [8]. If the anchor boxes consists of mostly background which is considered as the easy class, then it will be downweighted by subtracting it from 1 to limit the effect of it on parameters.

This loss function takes it roots from the plain Cross Entropy loss [52]. Focal Loss concept added on top of Cross Entropy loss ensures giving more weight to penalize false negatives by adding the modulating factor to the equation.

*Batch size is the number of samples the Neural Network uses the calculate the loss for each iteration. Working with different batch sizes can have an effect on the outcome. Working with larger batch sizes are computationally more efficient but it can introduce a convergence bias which might end up in less accurate local minimums

3.3 Keras Retina Net with different Hyper Parameters

Retina Net is implemented with different batch sizes. As the optimizer, stochastic gradient descent is used which is the default version where the batch size is equal to 1. Then, experiments are also conducted with batch size = 8.

3.4 Wavelet Transforms and using Wavelet Transforms with CNNs

Wavelet transforms is an important tool to decompose signals in the time-frequency domain. Most important thing about wavelet analysis is that it enables us to make a "localized" analysis about the signal. Wavelet Transforms has been used in many different areas such as speech-compression and analysis, image compression and enhancement and removing noise from audio and image data [53]. Continuous wavelet transform can be formulated as follows:

$$W(a,b) = \int_{-\infty}^{\infty} \psi_{a,b}(t) f(t) \tag{3.2}$$

whereas $W(a,b)$ is the wavelet transform of continuous variables a (shifting) and b (scaling), f is a continuous function and $\psi(t)$ is the mother(basis) wavelet function, which can be thought of a basis transformation function [54]. In wavelet transformations, our aim is to extract the representations of the signal or a function in terms of the mother wavelet. Thus, it will give us a chance to evaluate the the decomposition of the signal in terms of the mother wavelet patterns.

What we are interested in our case is using wavelet transformations with 2D images, which can be done by Discrete Wavelet Transforms. Before going into that, first some of the main mother wavelet signals will be examined. I, then try to decompose the signals based on the selected mother wavelet and measure it in terms of that wavelet coefficents. This can be done for different scales and frequencies [55]. In Figure 3.9 below, some of the common mother wavelets are shown.

So far, information about an overview of wavelet transformation which was continuous way of doing them were explained. However, in our setting where we work with 2D images we will need to use discrete wavelet transforms because of the nature of our problem (spatial information consisting of pixels are considered discrete), since the images we use are also given in a discrete setting. If we take a closer look how wavelet transforms works, we first need to recognize the main components we use in

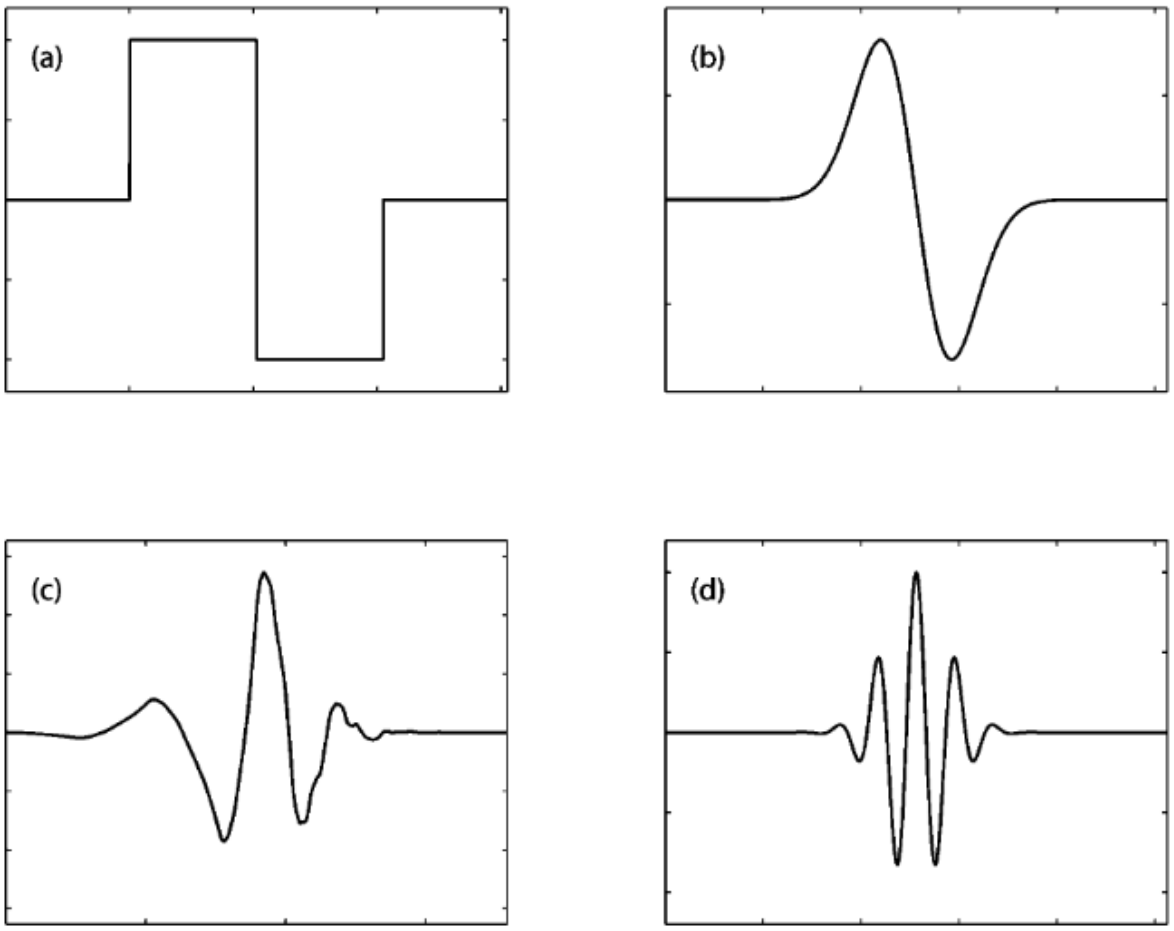

Figure 3.9: Common mother wavelets used for wavelet analysis: (a) Haar wavelet, (b) Gaussian wavelet of order 1, (c) Daubechies wavelet of order 4, and (d) Morlet wavelet. Source [55]

Discrete Wavelet Transforms which are the mother wavelets.

In our task, 3 main mother wavelets are applied to the images from our dataset to see how they work and then, based on the outcomes most promising one is selected and fed to the Neural Network. Below are the wavelets, which have been tested in this work:

- Haar Wavelet
- Daubechies Wavelet
- Discrete approximation of Meyer Wavelet

To start defining them briefly, Haar Wavelets are first introduced by Alfred Haar in 1910 on his thesis [56] and it has been used in many different problems, since Haar transforms works in a straightforward logic when applied to discrete images using high pass and low pass filters. To understand how these high pass and low pass filters behave

in Haar wavelet setting let's assume we have a 1D signal, a vector as follows: [8 4 8 4], which has a dimension of 1x4, and is depicted in Figure 3.10. First level discrete Haar transform for this vector would be calculated as averaging the consecutive elements of the vector and keeping track of the detail coefficients for each scale, which will allow us to construct the original values by using these coefficients when the calculation is finished. Calculation is finished when there is no consecutive number in the coefficient vectors to average. This would be easier to understand by examining the figure below. As it can be seen from the figure 3.10, in the first scale, we end up with a down sampled, averaged version of the original vector which would be [6 6]. This is the approximation coefficients of scale 1 and the detail coefficients of scale 1 are the distance from these averaged values to the original values, [2 2] which can be propagated from output to original input. In this example, detail coefficients are calculated as follows, for the scale 1, ((8-4) / 2) = 2 and ((8-4) / 2) = 2 and for scale 0, ((6 - 6) / 2) = 0. This gives us D0 = [0] and D1 = [2 2]. Wavelet coefficients in this case would be [A0, D0, D1] which corresponds to [6 0 2 2] where A0 means approximation coefficient with scale = 0, D0 is detail coefficient with scale = 0 and D1 is detail coefficients with scale = 1 [57].

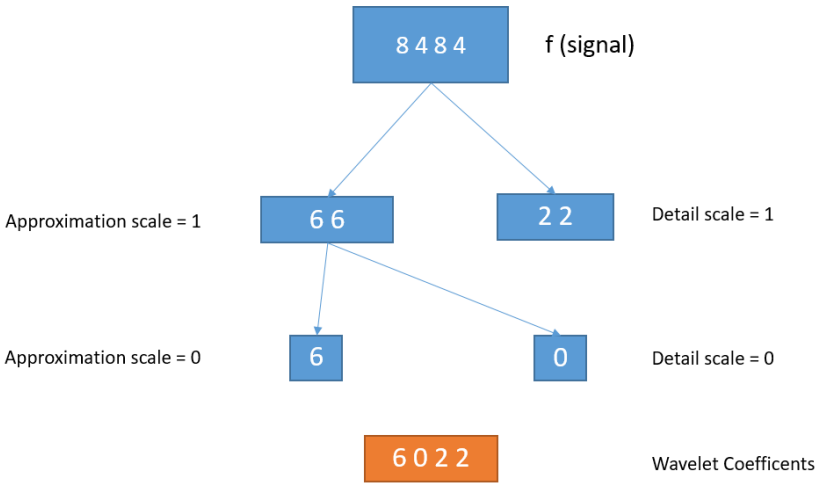

Figure 3.10: Wavelet Coefficient Calculation of 1D 1x4 Wavelet

This simple, computationally efficient Haar function calculation of a signal allows us to decompose it into different wavelets, which gives characteristic information about the original signal. We can think of Discrete Haar Wavelet Transform in 2D setting with the same logic. By passing the low pass and high pass filters it allows us to get characteristic and localized information about the input image. High pass and low pass filters can be thought of convolutional kernels. An example of them, in a 2D image setting can be seen in Figure 3.11. When the original image is convolved with

a high pass filter, the image gets sharpened, whereas the low pass filter, which is the opposite of high pass filter, gives a smoothing effect to the applied image by averaging the adjacent pixels. High pass filters aim to keep the high frequency information and low filters aim to keep the low frequency information [58].

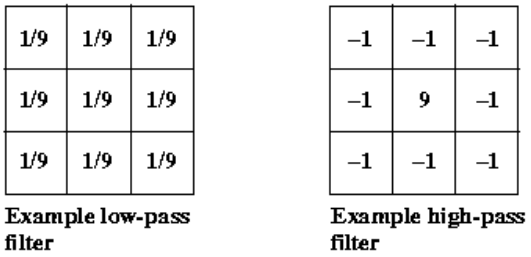

Figure 3.11: High Pass and Low Pass Filter Example [58]

Below in Figure 3.12, we can see a wavelet decomposition tree which shows how Haar Wavelets can give distinguishing information about an image with a computationally low cost calculation.

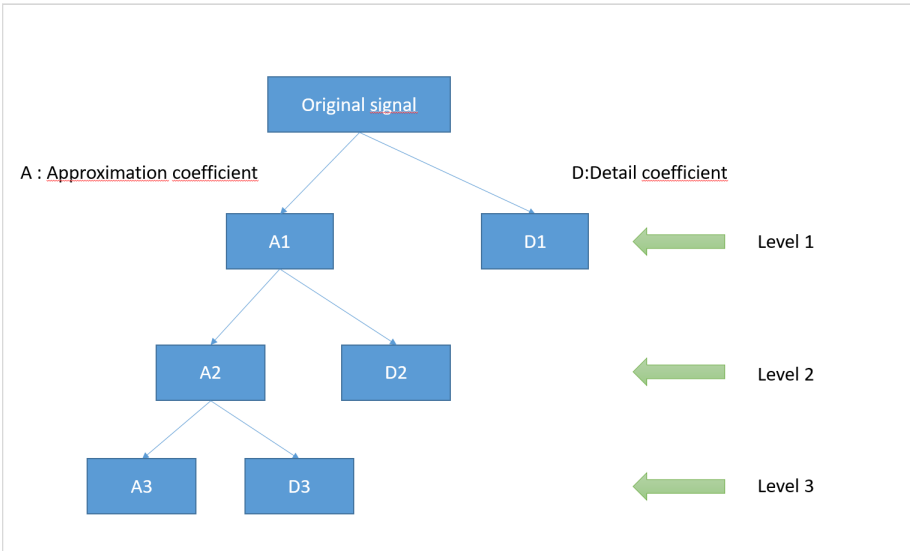

Figure 3.12: Wavelet Decomposition Tree example. A_i, D_i are approximation and detail coefficients consecutively [59]

The basic calculation example above was about 1D signal. When it comes to 2D signals, like images in our case, it is actually the same process, but generalizing the 1D version to 2D as mentioned with the high pass and low pass filtering above. Every row is treated as a 1D signal and all the pixel values in a row is transformed in the same way we did above in the 1D transformation. After applying the 1D wavelet transform to all the pixel values in a row, we will end up with detail coefficients for

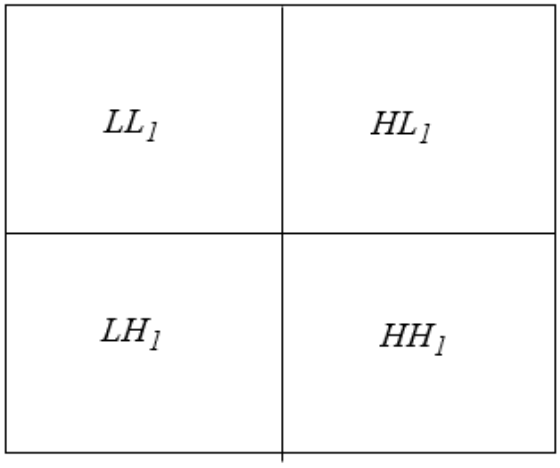

Figure 3.13: a) Single level decomposition scaling = 1. L denotes the low pass filter, H denotes the high pass filter

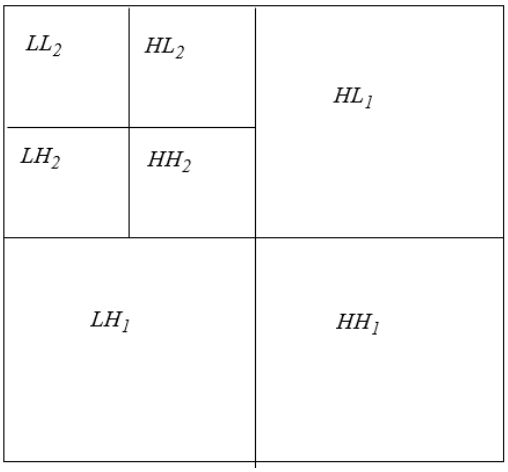

Figure 3.14: b) Two level decomposition scaling = 2. L denotes the low pass filter, H denotes the high pass filter

each row alongside with an average value. Subsequently, these transformed rows are treated as an image and in a second transformation pass, each column starts to get 1D wavelet transforms. Again, the result would consist of detail coefficients and one value for the average. To complete the transform this process is recursively gets applied to the quarter size image, until it only contains one average.

Discrete Wavelet Transform (DWT) can be in different levels and actually deeper levels might provide semantically more valuable information based on the task. In Figure 3.12, it can be seen how wavelet transformation occurs for 3 levels setting with a graph more clearly before the actual example of an image is shown. In Figure 3.12 above, it can be observed that in each level, a signal is decomposed into detail and approximation coefficients and this goes on until level 4 in this case. In each of the detail and approximation coefficients, the signal decomposed into 4 bands. For each detail decomposition in each level (D) there are 3 coefficients: horizontal, vertical and diagonal detail coefficients. Horizontal is formed by passing a high pass filter followed by a low pass filter (HL), Vertical is formed by passing a low pass filter followed by a high pass filter (LH), diagonal detail coefficient consists of 2 High pass filters (HH) as figures 3.13 and 3.14 shows that more clearly for 2D DWT setting.

For a real image, the results would look like in Figure 3.15. In that figure, we see a 1st level wavelet transformation example. Original image is decomposed to 4 sub bands similar to 4 "squares" in Figure 3.13 and it also shows what filters each square corresponds to in the real image. On the top left side, there is the approximation coefficient, top right is the horizontal detail coefficient, bottom left is the vertical detail coefficient and on the bottom right there is the diagonal detail coefficient. To

use the resources efficiently, our approach was to first apply some different forms of wavelet transforms listed previously, to example images in our problem and see the outputs. If the outputs looked promising in any way we can then start to use them in training processes of the Neural Network. Let's try to look to some of the real images we are going to use and how the different wavelet transformation looks on them. Our aim is to find transformations which can be fed input as the Neural Network and improve the learning task, but we first experiment with some different outputs of the transformations and manually evaluate if they are potentially good to be used as an input.

Figure 3.15: Haar Wavelet Transform of image Lena [59]. Each transformation highlights different features of the image

There are some defects, which are larger than the others, whereas there are some defects that are smaller. We will be trying to see if wavelet transforms are going to help us to distinguish them from each other and from background. As in the Lena Image in Figure 3.15, horizontal and vertical coefficients of the wavelet transform would highlight different details such that in the horizontal detail coefficient (top right) eyes are more apparent, whereas in the vertical detail coefficient (bottom left) nose is highlighted, diagonal coefficient seems to be in between. This is only one level transform and we will try deeper levels of the transform to see which one does better in terms of identifying the defects.

Before proceeding to the actual experiment results, a pre evaluation step will be applied to see which one of the wavelets will have a potential to perform better. Based on the example transformations it will be decided which type of wavelet should be fed to the Neural Network. Let's look at some of our images and wavelet transformed versions of it. We used MatLab Wavelet ToolBox to generate these transformations. In Figure 3.18, we can see Dmey (Discrete Meyer) wavelet is not useful because it has some unusual patterns which spoils the image. The cause might be due to the boundary effect, but we did not study through that because it is out of the scope of this thesis. On the other hand, Haar and Daubechies wavelets provide promising results. Here in the image below (3.16), there are 2 defects, a bigger one on the top and a tiny one on the middle right. Both Haar and Daubechies transforms seems like they are able to highlight the bigger defect quite well in their Horizontal detail coefficient of level 3. They almost give the same results and we decided to move on with only one of them, which is the computationally efficient Haar Wavelet Transform. Deeper levels especially the 3rd level transforms are too small in Figure 3.16 , 3.17 and 3.18 and hard to see in detail. Larger versions of 3rd level coefficients can be found in the Appendix section.

As it can be observed, the bigger defect is highlighted better, but small one is not that easy to distinguish. It can bee seen that there is a lot of background noise, which gives similar pixels to the small defect, but in fact they are not actually defects. This problem is tried to be tackled by using Total Variation (TV) Denoising [60] in the pre-evaluation step and if the transformation would have seemed promising we could use them in our Neural Network training. With a certain threshold, TV Denoising might help us to eliminate some of these pixels and give a better highlighted areas of the defects. Next, TV Denoising will be discussed, then finally implementation details of our methods defined in this section and corresponding results will be presented.

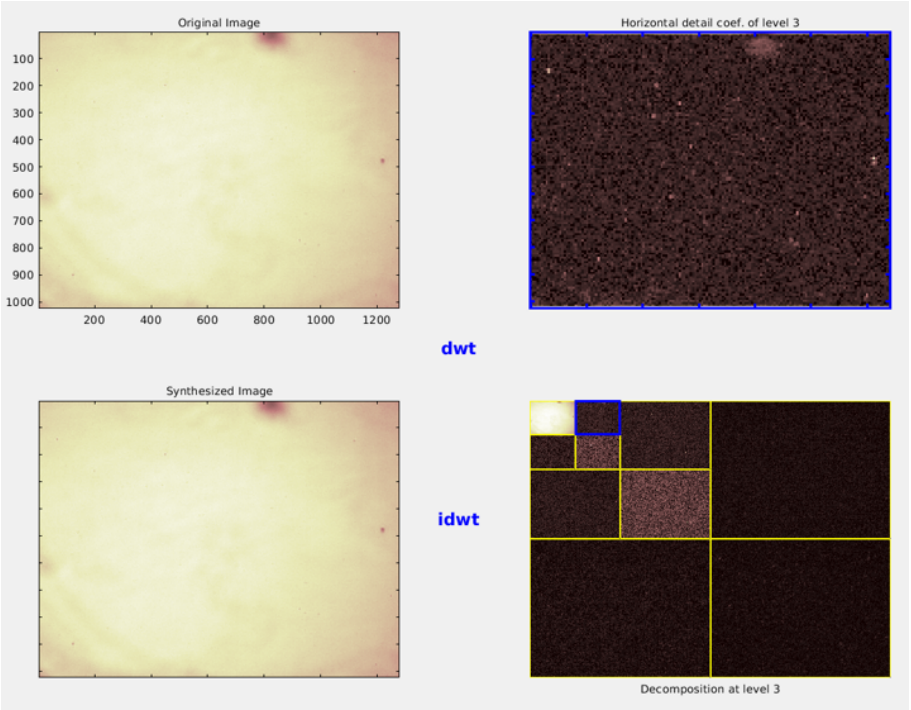

Figure 3.16: Haar Transform of an example image [59]

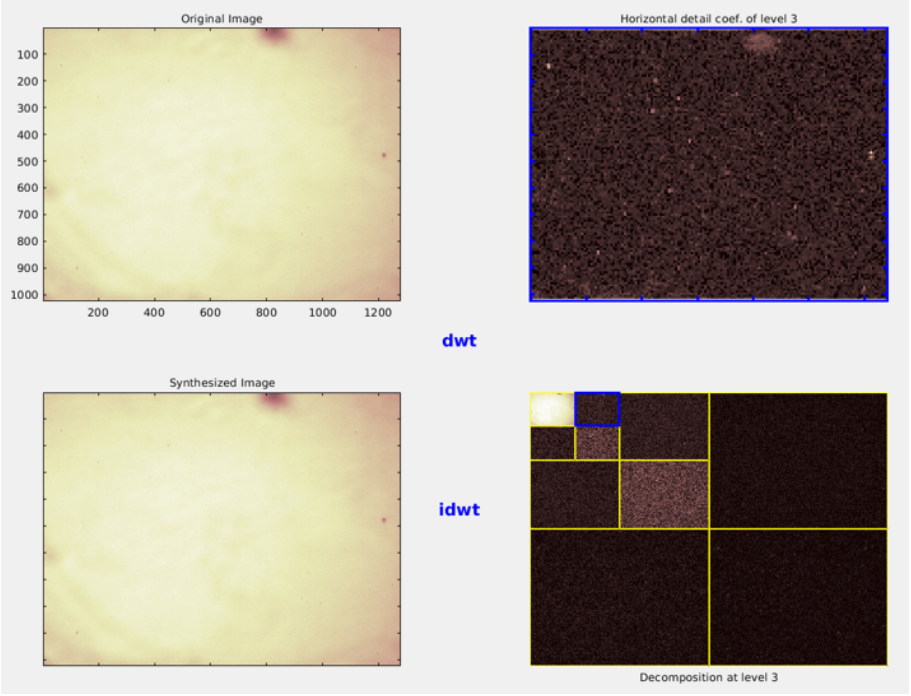

Figure 3.17: Deabuchies Transform of an example image [59]

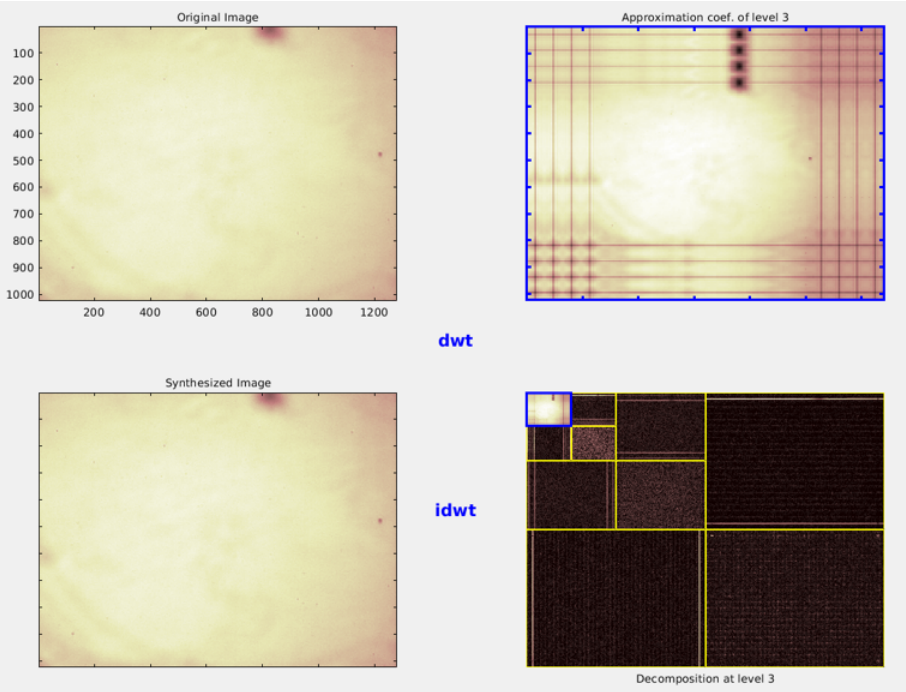

Figure 3.18: Dmey Transform of an example image [59]

3.4.1 Total Variation Denoising

Total Variation Denoising is one of the most commonly used denoising algorithms with the purpose of keeping the sharp edges in the image. The one we use is based on the Rudin-Osher-Fatem (ROF) [60] image denoising model, which is a minimization problem of the function below:

$$\arg\min_{u\epsilon BV(\Omega)} ||u||_{TV(\Omega)} + \lambda/2 \int_{\Omega} (f(x) - u(x))^2 \, dx, \tag{3.3}$$

where u is the restored image, λ is the regularization parameter and f is the noisy original signal. $TV(\Omega)$ represents the Total Variation Domain and $BV(\Omega)$ represents the Bounding Variation domain, more detail about the algorithm can be found here [60].

After examining our target images which will be used in the experiment, in some images it can be seen that there are background noise which is roughly similar to defects but in fact they are only measurement errors or background pixels (recall the images in 3.1 section). It is not wanted to have background pixels to be labeled as defects. TV denoising is a promising technique to apply to these kind of problems in pre-processing step, because it has the smoothing effect, which can eliminate the noise, hence the false positives, before the prediction process starts. Here in the figures below, there is an example of TV Denoising applied to a natural image first to see how it behaves more clearly followed by the images from our dataset.

In Figure 3.19 & 3.20, we can see the effect of TV denoising more clearly on a real image before we apply it to our images from the dataset. In these figures, it can be seen that denoised image highlights the sharp edges more and gives a smoothed, "cartooned", look to the original image. Background noise is reduced significantly and the image looks a bit blurred.

Now, the same denoising will be applied to one of the original images from our dataset and to the Haar coefficients of it. When TV Denoising applied to the original image the results are not too promising, however when we apply it to the Horizontal level 3 coefficient we can see that it significantly reduces the noise.

Here in Figure 3.21, the difference between the H3 (Horizontal Level3 Haar coefficient) and Total Variation Denoised H3 can be seen. Background noise is decreased significantly and 2 defects is highlighted better than the H3 coefficient before TV Denoising. What can be don with this denoised image and the Haar coefficients is that, they can be used in our deep learning model to help the model in making better feature extraction. This can be thought as an additional data pre-processing step as some studies showed when the Neural Network is fed with transformed images rather than original image only, it could possibly contribute to the learning [62]. In addition, Fig-

Figure 3.19: a)Original Image [61]

Figure 3.20: b)Tv Denoised [61]

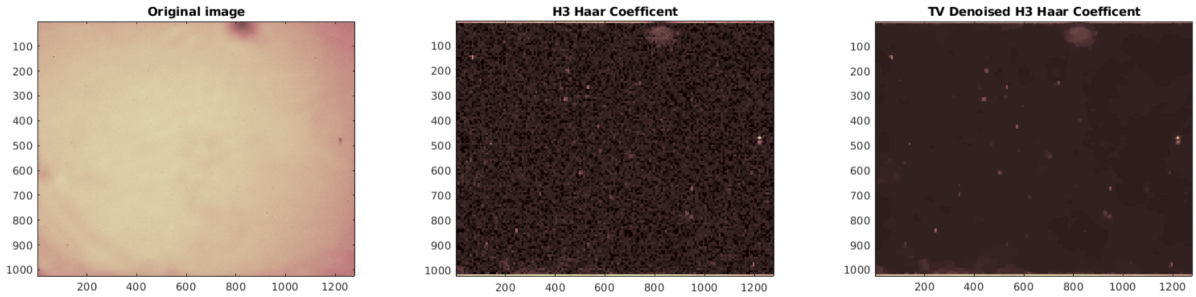

Figure 3.21: TV denoising effect on Haar coefficient On the left original Image, in the middle Haar 3rd level coefficient and on the right TV denoised Haar 3rd level coefficient

ure 3.21 shows us the singular, large noise spots can be better highlighted when 3rd level Horizontal Haar coefficient is combined with TV denoising. In this work, it will be examined if that can be generalized for all cases in our task to further improve the learning.

As mentioned previously, when it comes to Convolutional Neural Networks main advantage of them is their ability of doing automated feature extraction. However, what kind of initial input is best for a CNN is an important question in our task. In some cases adding hand crafted features to the CNN or applying transformations to the raw input image in the pre-processing step could improve the performance [16] [63]. Before moving to the Results and Discussion chapter, to remind the simple example mentioned previously about this phenomenon, we can think of a Neural Network which takes the date as input variable. In which format the date should be fed to the Neural Network? In theory, neural network would be able to understand the encodings of unformatted date after some iterations by itself (such that which digit corresponds to hour, day, month, year etc.) and learn which column corresponds to month, day or year. However, common usage in Neural Network applications is to encode the date into some formatted feature representation before feeding it to the Neural Network because it performs better than raw unformatted version of it [64] [14]. As discussed in the papers, higher-level variables for date and time are constructed to better capture their effect such that quarters of hour, day of the week, week of the year. Same logic applies to the input images we have. We want to see if higher-level variables of our input image also improve the learning performance. This time, instead of date we have the original image and as the date gets formatted before fed to the network, we try to pre-process the original input with TV Denosing and Wavelet Transformations to see it's effects on learning.

One of the things in the next section which will be tested is, whether this assumption holds in our problem setting or not. It will be compared how the performance of Neural Network differs when we use only the original image, one of the coefficients, TV denoised version of the H3 coefficient and finally feeding all the coefficients alongside with the TV Denoised version of the H3 coefficient. To see the differences, all the other parameters of the network is kept constant.

4. Results and Discussion

Before diving into the comparisons and results of different networks, one of the first research questions discussed was to find an answer to when to end training. How many epochs would be sufficient? First, the model is ran with 180 epochs (default) to see if overfitting occurs and when does the model perform the best on the test set. Overfitting is an important phenomenon in Neural Networks as discussed in previous chapters. One of the most crucial metric we need to take into consideration. It means the Neural Network memorizes the data instead of learning, where training error is low but validation error is high. This would be an undesired outcome because it means the generalization power of the neural network is low and it only fits the training set very closely. Neural Networks can easily overfit because of the complex non linear mappings they use. So, overfitting should always be taken into consideration when training Neural Networks. Results showed that after the 50h epoch over fitting starts (this is where training error started to decrease sharply whereas validation error started to increase) and especially after epoch 80 accuracy starts to decrease due to overfitting, which will be discussed next. After analyzing that, our experiments has ran with a maximum of epoch number 80.

Another important thing is the evaluation criteria. Retina Net prediction consists of a class and a regression box. We calculate the losses for both of them separately in each iteration of the Neural Network. Loss values produced by these loss functions are one of the evaluation criterias and the other one is the true count number (accuracy of predicted number of defects disregarding the category of them, correct if defect is detected); which is the count where how many of the defects in a given image are predicted by the model is compared with the actual number of defects which were manually labeled (ground truth). Also, number of false positives are important in our task. A false positive in our task is, for a given image, where the model predicts a defect (coordinates) when it is not. This is another point which will be discussed in this section.

Besides the loss function, in order to actually see the detection patterns and evaluate the quality of the model -also figuring out when to stop training-, a small video tool is developed by the author where prediction of each model for a given image

is displayed in a video for each epoch. Doing manual evaluation with this tool was fairly convenient and clearly showed that especially after the 80th epoch detection quality decreases, which confirms the values coming from the loss functions (training validation). In Figure 4.1 and 4.2, some examples from the video tool and the change in the loss functions after the 80th epoch are shown which will be discussed next.

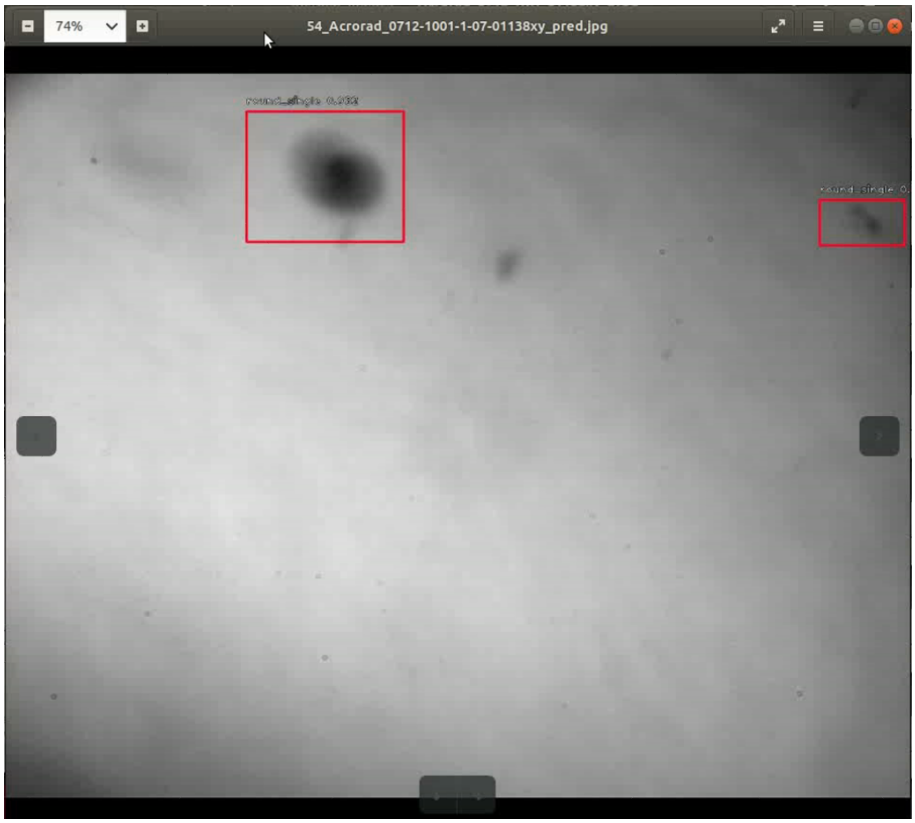

Figure 4.1: Manuel Evaluation Tool at Epoch 54 - predicted defects are marked in red

In Figure 4.1, a screen shot can be seen from the manual evaluation tool which generates a video, where defect predictions for the same image from each epoch is applied to the image (e.g. model after each epoch is saved and applied to the same image) . Number at the top of the image "54" in this case- shows the number of epoch and in different epochs both the shape of the bounding box and location of defects vary. That can be seen in another screenshot of the video tool in Figure 4.2. Here, for the same image, where this time we see the prediction of the model at epoch 67. Here it can be seen that there is one more extra detection compared to the Figure 4.2, which is actually a false positive. Using this tool helped to manually observe the behavior of the network as well as being aware of the false positives, which are important in our task in terms of decreasing the number of them as much as possible and also to show us where to stop.

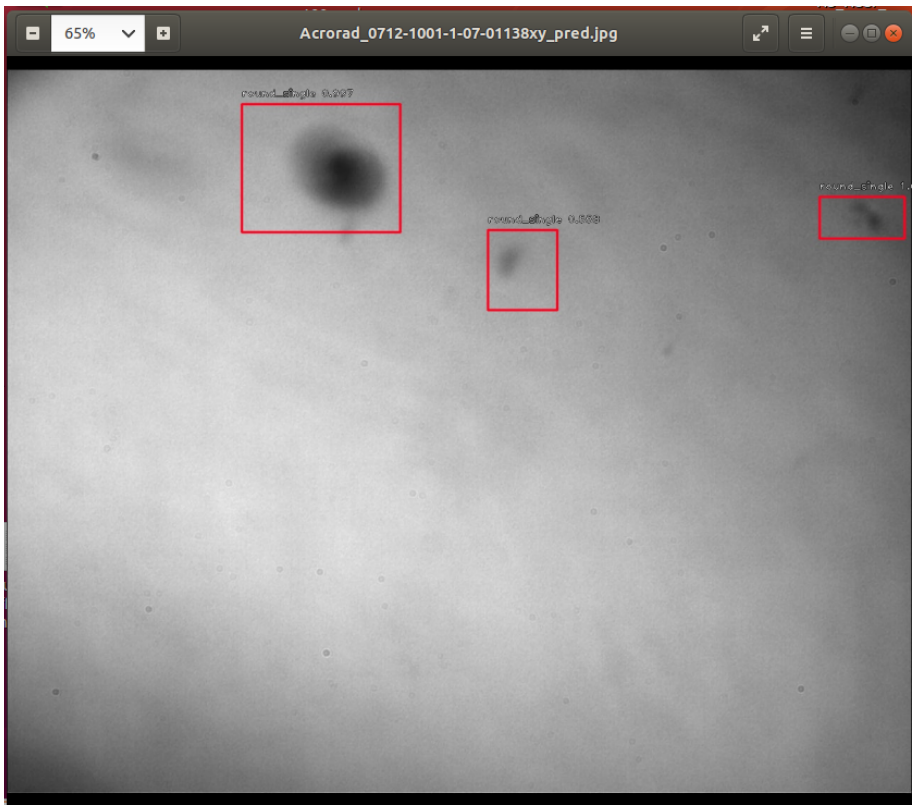

Figure 4.2: Manuel Evaluation Tool at Epoch 67 - predicted defects are marked in red

The loss function in Figure 4.3 displays the training loss which shows the change in the loss function over epochs of the training set. It can be seen that training loss starts to decrease after the 80th epoch, which might indicate an increase in the accuracy. This seems like a good thing at first look. However, the reason it starts to decrease is that due to the overfitting which can be confirmed by looking to the validation loss 4.4.

In Figure 4.3, the training loss figure, x-axis shows the number of epochs and y-axis shows the corresponding loss values. Our aim is to make the loss value as low as possible, while not overfitting. Here we can see a convergence point in Figure 4.3, where the training loss does not get lower any more and the curve becomes asymptotic which indicates there is a possibility of overfitting. To confirm this we also need to look to the validation loss. Validation loss is calculated the same way mathematically as training loss is calculated by using focal loss, explained in 3.2.1. Difference is, there is no weight update occurring in validation steps. It is used to calculate the loss value for the samples which network did not see in training. It is used to tune the hyperparameters, find out when to stop, detect overfitting and see how good is the generalization power of the model until that point in training epochs. Back in our case, in fact, while the training loss doesn't decrease any more and it's curve becomes asymptotic, validation loss starts to increase after 80th epoch, which means after that

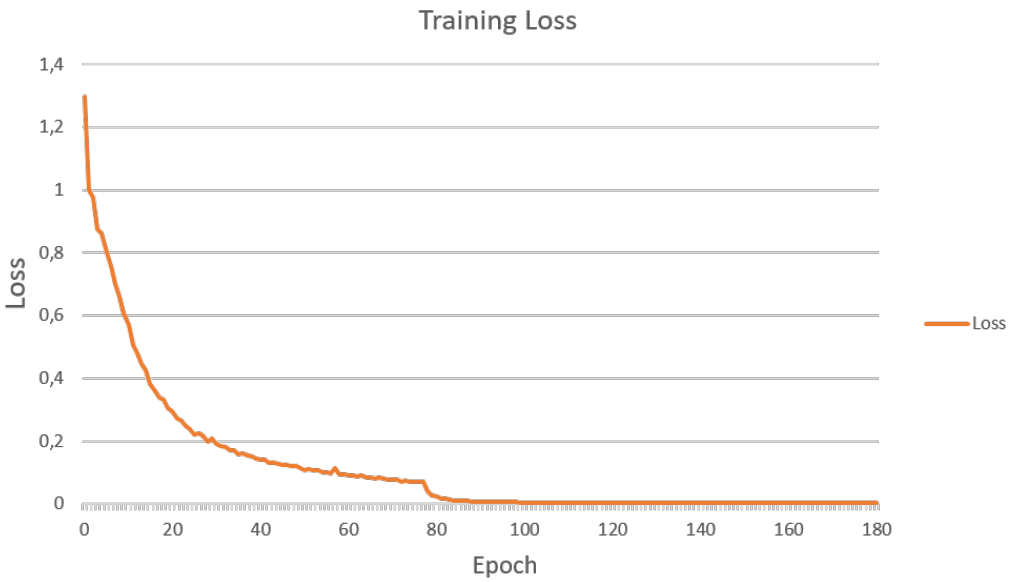

Figure 4.3: Training Loss : Change in Training loss over 180 epochs

point training more epochs is redundant. This can be seen in Figure 4.4 where the validation loss graph is shown. Validation loss starts to increase after the 80th epoch whereas as we examined, the training loss starts to decrease after the same point. This is a strong indication of overfitting and tells us to stop overtraining.

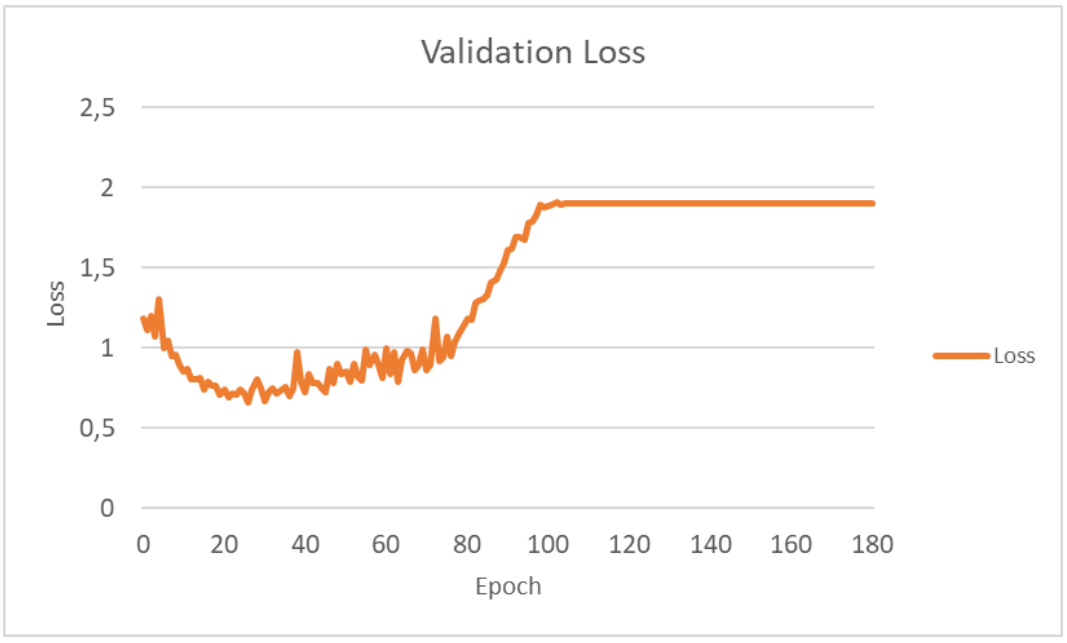

Figure 4.4: Validation loss : Change in Validation loss over 180 epochs

Validation loss starts to increase after the ≈50th epoch and then it converges after the ≈120th epoch. As it can be seen, the validation loss increases while the training loss decreases, which means model is starting to memorize the training set and

generalization power of the model starts to decrease. At the same time, when we check the results of the manual evaluation tool it also confirms our findings about overfitting and detection (also classification) performance starts to decrease especially after the 80th epoch. After these findings (by the loss function and manually evaluating the classification quality) it is concluded that 50 epochs of training would be sufficient but to be certain we kept training for the 80th epochs as it is obsereved that 180 epochs of training is both time and resource consuming. When the performance of 50th and 80th epochs are compared, although there is no big difference in terms of detection accuracy, 50th epoch is slightly better and validation loss is lowest on the 40th epoch for the training of the original image. So far, only original image is used for the results.

When to stop training is an important question when working with Neural Networks in terms of both for the performance of the model and resources used for it. Also, it takes considerable amount of time to train more epochs when working with large data sets. There are 2 basic, yet popular approaches on early stopping. Thresholding the loss value itself and thresholding the decrease rate of the loss. In addition to that, there is an optimization method called patience (patience step), which does not trigger the early stopping until N epochs ensuring at least some particular number of epochs passes before the stopping is triggered. One problem with these approaches is that they might contain some level of bias (by setting the threshold manually), but if we can find a reasonable loss level in our task, which has already found, then we can set it as a threshold for the further experiments alongside with thresholding the decrease rate of the loss. Best time to stop is usually when the training accuracy starts to increase, while the validation accuracy starts to decrease, then overfitting occurs and its time to stop training but we saved all the models in each epoch, to be able to compare their performance manually.

Since the background information about the experiments and the methods which are going to be used in these experiments are explained in previous chapters, now we can run them and see the results. In our experiment setup, 4 different runs of the model with 4 different inputs will be compared. These 4 different inputs which will produce 4 different models can be listed as follows:

- Original Image
- Horizontal Level 3 Coefficient of Haar Wavelet
- TV Denoised Horizontal Level 3 Coefficient of Haar Wavelet
- Concatenated Image

Concatenated image is created by combining the original image, wavelet coefficients (horizontal and vertical) and TV denoised H3 Haar coefficient. Below in Figure 4.5 and 4.6, 4 input images separately fed to the network can be seen.

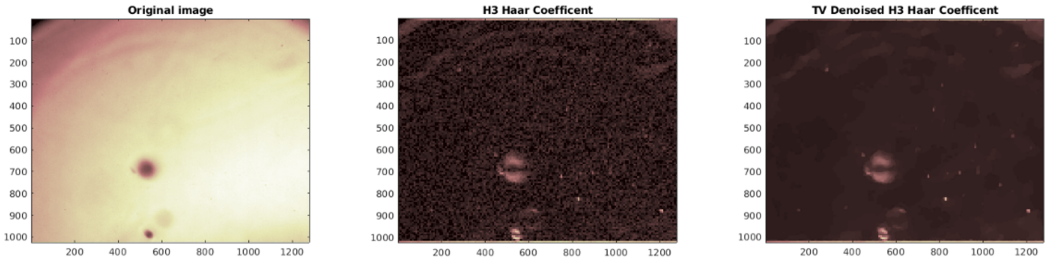

Figure 4.5: Different Inputs for Neural Network. Original image on the left, Horizontal Level 3 Haar Coefficient in the middle, TV Denoised Horizontal Level 3 Haar Coefficient on the right

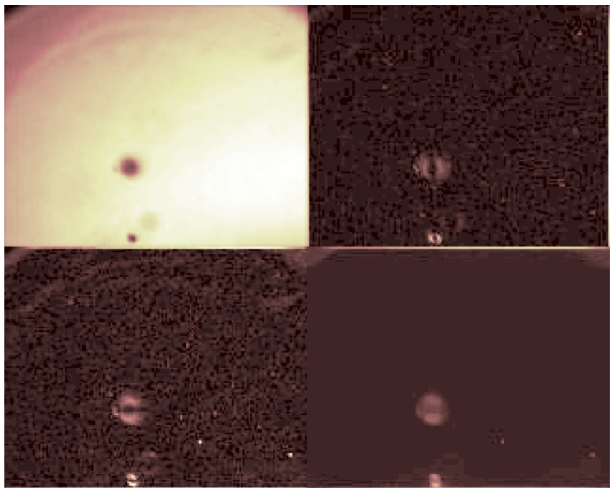

Figure 4.6: Concatenated Image : Original image on top left corner, Vertical H3 Coefficient on top right corner, H3 Haar Coefficient on bottom left corner, TV Denoised H3 Haar coefficient on bottom right corner

Let's take a loot at the results of these different inputs. First, how their loss function behaves will be compared. Then, count based evaluation of the defects will be conducted. Network is ran with stochastic gradient descent which is explained in the loss function section, where batch size is equal to 1. Each of them are trained for 80 epochs with a decaying learning rate.

Here in the figures (4.7 - 4.8) below , we can see the change of the training and validation losses. The lowest validation error is around 40th epoch's, whereas training loss decreases sharply after the 50th epoch. For all of them, also at the same time validation loss starts to rise up significantly after 50th epoch.

It can be said that, based on the performance on the validation set, original model does slightly better than the other 3 models. Around epoch 30 they are the closest. If

Figure 4.7: Change in Training Losses Over Epochs of 4 Trained Models with Different inputs. Inputs are referring to; Original (Raw Image) , H3 (3rd level Haar Coefficient of the Original Image), TV_H3 (TV Denoised 3rd level Haar Coefficient of the Original Image), MAT3 (Concatenated image consisting of Horizontal H3, TV_H3, Vertical H3 and original image)

we sort them based on how they perform on the validation set in 30th epoch, we get this kind of a result consecutively with the validation loss values as 0.42, 0.58, 0.63, 0.81.

1. Original Image (Blue) - 0.42
2. Concatenated Image (Red) - 0.58
3. Horizontal Haar coefficient 3rd level (Dark Orange) - 0.63
4. TV Denoised Horizontal Haar coefficient 3rd level (Grey) - 0.81

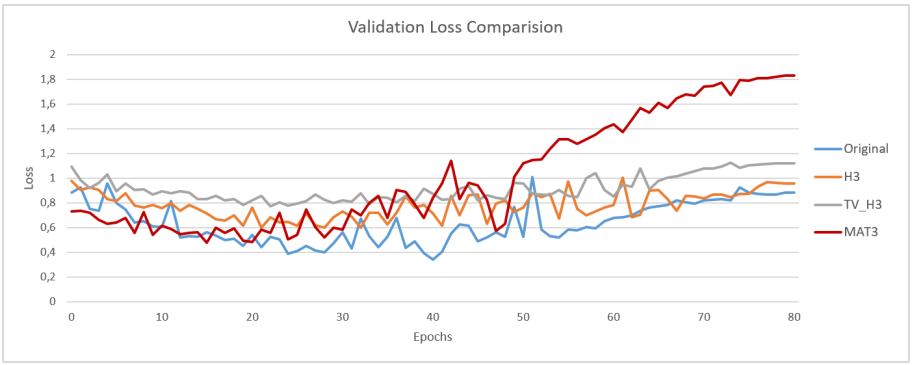

Figure 4.8: Change in Validation Losses Over Epochs of 4 Trained Models with Different inputs

To do the evaluation based on the actual defect counts, 200 random images from the test set are chosen. Then each of the 4 models are run with these inputs. All the models used are from their 30th epoch's. Using the prediction tool made by the author, which can predict one images or several images based on the given folder containing the images to be predicted (which is provided by the user and with a validation set

option). In case there is a validation set, the accuracies would be displayed. Results from the prediction tool are as follows in Table 4.1.

Model Input	Accuracy
Original	0.9685
H3 Haar Coefficent	0.8831
MAT3	0.7012
TV H3	0.5324

Table 4.1: Results based on detected defect counts

Original (Raw Image) , H3 (3rd level Haar Coefficient of the Original Image), TV_H3 (TV Denoised 3rd level Haar Coefficient of the Original Image), MAT3 (Concatenated image consisting of Horizontal H3, TV H3, Vertical H3 and original image)

Here it can be seen that original model works the best, with a quite high count accuracy followed by H3 (Horizontal level 3 Haar transformation of the original image) with a decent accuracy, MAT3 (concatenated image) comes as third and TV H3 (Total Variation denoised version of H3 Haar coefficient) performs the worst. This table confirms the validation loss results, while it also shows us the small differences on the validation set error, in this case can lead to big differences on the actual detection accuracy. This indicates that small differences between the validation set error, in fact, has larger effect on the count based evaluation. For instance, validation loss of the 30th epoch for the original image and TV H3 is 0.42 and 0.81, respectively. While difference is 0.40 on the validation losses, the count accuracy of the original image is %96 and TV_H3 is %53.

Both H3 Haar coefficient and the original image did quite good on count accuracies separately, it means that they can be used together as well. MAT3 (concatenated image) results are interesting because it contained 4 different images and expected to perform at least as good as the H3 Haar coefficient alone but, in fact it performed worse. The reason of that is perhaps one of the components it contains, which is the TV denoised image. Model with the TV denoised image input has the lowest accuracy among all and including that image in the MAT3 (concatenated image) might as well decreased the overall performance of it. This is important in our experiment because a concatenated image contains all the images in it, which means the information Neural Network can learn is there all together and in theory, it shouldn't perform worse than any of the models with single images as input. However, the opposite is the case. To test this hypothesis, the TV Denoised (worst performed) image from the concatenated image is taken out and one more test is conducted by running the same concatenated

input model this time without the TV denoised component. This did not increase the accuracy performance significantly (Accuracy is 0.6831, slightly below MAT3). It is hard to tell precisely why the concatenated image did not work well but it can be due to the labeling or the way concatenation is applied to the images. When 4 images are concatenated, labeling is done to only to top left image by calibrating the coordinate labels. This means the prediction result we expect would be one label corresponding to the top left image's defect coordinates but not to the other concatenated images'. This might have introduced a bias and possibly making the concatenated images does not perform well. Other concatenation methods will be discussed in the Further Improvements chapter.

Another important outcome to note is that, even though MAT3 and TV H3 performed considerably worse than H3 and the original image. When false positive rates are evaluated it can be seen that they are doing slightly better than these two. This is mostly due to their poor performance on predicting the defects, but at the same time it decreases their false positive rates. This can be seen in Table 4.2.

To sum up the results, when we look to the earlier loss function results which was trained for 180 epochs, they showed us the networks are overtrained and starts to memorize the data points rather than learning from them, starting from 50th epoch. This is getting more severe after 80th epoch. It has been mentioned in the Materials and Methods section that different batch sizes are used in the experiments. Default batch size was 1 and we tried to use 2 different larger bath sizes (8 and 16) which did not proove an increase in the accuracy, however it gave us some gains in performance. Training time for batch size 16 was about x3 times faster compared to the configuration with batch size 1 and batch size 8 was about x2 times faster. This means in our problem setting using Stochastic Gradient Descent performs better than mini batch gradient descent. It is hard to conclude with an exact certainty about the reason of it but mini batch gradient descent might got stuck in local minima whereas SGD has more variance in parameter updates which has a potential to end up in a global minima. This can be one of the reasons SGD works better than mini batch gradient descent in our setting. Exploring the impact on mini batch size in a Deep Learning setting is an important resear ch area [65] which goes beyond the scope of this thesis.

In one of the sample types we consider as hard, where there is both a large defect and a small one, main problem is that detection of the smaller defect is below average. In these hard samples, TV denoised image underperformed significantly. It missed almost 50% of the small defects in this case. Original and H3 performed better on this. The reason TV denosing performs worse in this case might be that using a global denoising parameter is not optimal for this case. With a fixed denoising parameter, TV denoised image highlights the large defect well whereas it might cause small defect

to be considered as noise on the background.

Another interesting findings of our results is that even though H3 coefficient performed slightly worse than the original image, it still performed remarkably well, especially considering that resolution of H3 coefficient is 1/4 of the original image and it still performed quite good on predicting the counts. Training time of the H3 images are ≈ 2.5 times faster than the network with the original image.

All these models can be used in an ensemble learning setting in the prediction phase, which will be explained and discussed in the Further Improvements chapter. A majority voting on a test image can be done by using all or some of 4 models.

When false positives are considered independently, original and H3 coefficient results are very similar which can be seen in Table 4.2.

Model Input	False positive rates
Original	% 8.1
H3 Haar Coefficient	% 7.6
TV H3	% 5.7
MAT3	% 5.2

Table 4.2: False positive rates on randomly sampled test images. Lower false positive rates are desired

These experiments showed us even though there are performance gains from the other image inputs, in terms of general accuracy original image beats the other images. This does not mean that models with different inputs are completely useless. Especially, H3 Haar coefficient is very promising considering that it gives closer accuracy to the original image while having 1/4 of it's resolution. In computationally costly tasks this coefficient can be used instead of the original image, also using original and H3 together in an ensemble learning setting or with using advanced concatenation techniques can improve the performance of the Neural Network. Different ways of concatenation can further improve the performance which will be discussed in the Further Improvements chapter.

In this chapter we discussed different models created from different inputs using the same network architecture. Original model is compared to Wavelet coefficients enhanced models, Total Variation Denoised image inputted model and concatenated image model. The original model performed best in respect of the main task (to count the defects), but the other models provided important results for further improvements.

5. Conclusion

Automated defect detection is an important topic which it's applications are used commonly in manufacturing and material industries. It is one of the areas where Deep Learning algorithms are used actively in industry. This research is aimed to analyze the state of the art defect detection and object detection algorithms, as well as selecting and applying one network architecture to a real world industry problem. In this thesis, Convolutional Neural Networks are discussed in an automated defect detection setting. Research questions are stated in introduction section and before answering the research questions, background information about conventional object detection algorithms and Neural Network powered detection algorithms are discussed. Examples from different studies of how CNNs are used in defect detection is explained. Inverse Mathematics methods were used, such as Wavelet Transformations and TV Denoising. They are explained with comprehensive examples. Then, in the Materials and Methods section, which methods will be utilized for the experiments (pre-evaluation step as it is called) are presented. Based on the results, it is been concluded that our selection of CNN architecture (Retina Net) was well grounded, hence performed well on identifying the number of defects in a given IR image of Cadmium (Zinc) Telluride Cd(Zn)Te. Enhancing the Neural Network with different Inverse Mathematic transformations such as using wavelet transformation coefficients and TV denoising did not beat the original model but gave us important insights. This approach can be further improved with the usage of the methods explained in the Further Improvements chapter. Also, it has been observed that using more advanced Inverse Mathematics transformations has a potential to provide some development of the topic.

To summarize some of the key findings from the research conducted was that, first, usage of an FPN architecture is essential in a problem setting where we have objects/defects which differ in the sizes. Using SGD performed well with a decaying learning rate. Haar Wavelet transformation in 3rd level highlighted the defects successfully, especially the horizontal and vertical coefficients contains useful information about the image. TV denoising the original image did not provide a big difference on the image, but applying TV Denoising to a Haar Wavelet coefficient showed promising results, especially in terms of eliminating the background noise. However, large defects

were isolated successfully but the same can not be said for the smaller ones. It is clear that to benefit more from the TV Denoising some optimizations should be done to the algorithm. Original image performed best on detecting the count of the defect but when it comes to false positive ratio, H3 Haar coefficient performed better than the original model, in addition TV denoised H3 Haar coefficient also performed better than the original model. Determining when to stop was an important hyperparameter to deal with. It is concluded that saving the models only between the 30th and 50th epochs would give the best results.

Based on these conclusion, researchers should consider using Inverse Mathematics transformations (perhaps more advanced methods than the ones used in the thesis) to help the network to learn better. Especially, running the network with different transformed inputs, particularly with transformations which are computationally not expensive, in an ensemble learning setting could be promising.

6. Further Improvements

In this section, possible ways of future development of the experiments conducted in this thesis will be examined. There are 4 main points in this part. Advanced concatenation techniques, adaptive denoising, ensemble learning methods and usage of a segmentation network.

Firstly concatenation: In the experiments, after the model selection is done, alongside with the original image, different transformations of the original image are used. Wavelet transformations and TV Denosing separately applied to the original image, as well as concatenation of both to the transformed images and original image together. MAT3 is the concatenated example. Concatenated image created by combining them in the same dimension. Other approaches for concatenation might be more efficient, one way is to experiment with channelwise concatenation, instead of horizontal concatenation. Instead of horizontally concatenating the images, each additional image can be treated as a new channel, similar to the three RGB channels in a color image. Another way of doing the concatenation is, using an additional network or a layer for concatenation in the network architecture. This has a potential to work better because instead of applying the same filter to 4 images which are concatenated, another learning step would be added where that step would be dedicated for combining the information of 4 different images. Since we are dealing with a black box model, these possible methods all needs to be experimented separately with a try and error strategy to see if they can bring any additional benefits to the learning.

Second: In the results, it is observed that worst performing model was where the TV denoised image is used as input to the model (TV H3). In fact, before running the NN model, in the pre-evaluation step transformed image looked quite promising when TV denoising applied to it, which can be seen in TV Denoising section of Chapter 3. Applying TV denoising to the original image did not provide any major changes on the image, but when it is applied to the Haar coefficient it considerably smoothed down the noise in the background and highlighted the defects. However, main problem here was the usage of the smoothing parameter. When the parameter was low, larger defects were highlighted nicely, whereas same can not be said for the smaller defects. When a large and small defect contained in the same image the performance of TV

denoising decreased considerably. Also with a larger parameter, this time, some noise on the background did not get smoothed enough. To tackle this, an adaptive TV denoising [66] or a localized denoising technique can be used to further benefit from TV denoising. Main idea would be applying denoising with different parameters to different parts of the image. By doing that, both larger and smaller defects could be highlighted at the same time and noise can be reduced while doing this.

Another possible improvement is Ensemble Learning. Ensemble learning in a machine learning setting is combining different models in order to increase the accuracy of predictions [67]. It has 2 popular methods used in machine learning. Bagging and boosting. Bagging means running the same model with changing the dataset each time. This can be done by either taking a random subset of the dataset in each run of the model or changing the weights of the hard examples in the dataset, where hard examples mean the samples which have lower detection accuracy. Boosting, on the other hand, is using various trained models together to make the prediction for a test sample. Since we have 4 different models, it can be suitable to use them in a boosting ensemble learning setting. We have 4 models with different accuracies and weights. They can be used together when a test image will be predicted. 4 models can run separately and their outputs can be combined. This can be done in a majority voting setting such that averaging the predicted coordinates of the defect and the number of detections from each model. For example, if 3 out of 4 models predicts there is a defect and one does not than majority voting will decide that the sample consists a defect with a probability of %75. It would be even better to do the majority voting with a weighting, based on the accuracy levels of each model. In this case, most accurate model will have more impact on the outcome. Especially, the original image and H3 Haar coefficient are suitable to be used in this majority voting scheme. When it comes to bagging, it can be promising to see the outcome of training a model for different datasets or for an additional dataset which weights more the hard samples. This can be done either by modifying the loss function so that it would penalize the hard examples more/easy examples less or removing the number of easy examples in a subset of the whole dataset. These methods can increase the accuracy while compromising from the performance, but needs detailed research to see if they would actually bring any significant benefits to the defect detection problem.

Finally, an important further development to this research can be usage of a different network structure. The author has explained and worked with CNNs dedicated for object identification and detection throughout this thesis. These networks can be called classification networks. Using classification networks is one approach to the problem and another one is using a segmentation network instead of a classification one. Difference between segmentation and classification network architectures is

that, segmentation networks produce a mask output as the prediction rather than the bounding boxes, which most of the object detection networks produce. Some research are done using segmentation networks to tackle this problem and some used segmentation alongside with classification networks [31]. This approach can be used to test if segmentation networks such as U-Net can bring any improvements to this problem setting. U-Net is a network architecture which consists of 2 sections. Encoder subnetwork and decoder subnetwork. In encoder part of the network, it works like a conventional CNN architecture with stacked convolutions, pooling layers and activation functions. This encoder part does downsampling. Decoder part starts after encoding part ends, which uses transposed convolutions this time to upsample the feature maps outputted from encoder part of the network. Decoding part outputs a segmented prediction mask as the result of the network. More information about U-Net can be found here [68].

Bibliography

[1] J. Walsh, N. O' Mahony, S. Campbell, A. Carvalho, L. Krpalkova, G. Velasco-Hernandez, S. Harapanahalli, and D. Riordan, "Deep learning vs. traditional computer vision," 04 2019.

[2] O. Essid, H. Laga, and C. Samir, "Automatic detection and classification of manufacturing defects in metal boxes using deep neural networks," *PLOS ONE*, vol. 13, pp. 1–17, 11 2018.

[3] D. Yapi, M. Mejri, M. S. Allili, and N. Baaziz, "A learning-based approach for automatic defect detection in textile images," *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 2423 – 2428, 2015. 15th IFAC Symposium onInformation Control Problems inManufacturing.

[4] R. Triboulet, Y. Marfaing, A. Cornet, and P. Siffert, "Undoped highâresistivity cadmium telluride for nuclear radiation detectors," *Journal of Applied Physics*, vol. 45, no. 6, pp. 2759–2765, 1974.

[5] A. Gadda, A. Winkler, J. Ott, J. Harkonen, A. Karadzhinova-Ferrer, P. Koponen, P. Luukka, J. Tikkanen, and S. Vahanen, "Advanced processing of CdTe pixel radiation detectors," *Journal of Instrumentation*, vol. 12, pp. C12031–C12031, dec 2017.

[6] G. A. Carini, A. E. Bolotnikov, G. S. Camarda, G. W. Wright, R. B. James, and L. Li, "Effect of te precipitates on the performance of cdznte detectors," *Applied Physics Letters*, vol. 88, no. 14, p. 143515, 2006.

[7] P. N. Luke, M. Amman, and J. S. Lee, "Factors affecting energy resolution of coplanar-grid cdznte detectors," *IEEE Transactions on Nuclear Science*, vol. 51, pp. 1199–1203, June 2004.

[8] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017.

[9] H. S. Kang, J. Y. Lee, S. Choi, H. Kim, J. H. Park, J. Y. Son, B. H. Kim, and S. D. Noh, "Smart manufacturing: Past research, present findings, and future directions," *International Journal of Precision Engineering and Manufacturing-Green Technology*, vol. 3, pp. 111–128, Jan 2016.

[10] "The mathematics of neural networks." https://medium.com/coinmonks/the-mathematics-of-neural-network-60a112dd3e05.

[11] "Understanding feedforward neural networks." https://www.learnopencv.com/understanding-feedforward-neural-networks/.

[12] "Finnish inverse problems society web page." https://www.fips.fi/.

[13] T. A. Bubba, G. Kutyniok, M. Lassas, M. März, W. Samek, S. Siltanen, and V. Srinivasan, "Learning the invisible: A hybrid deep learning-shearlet framework for limited angle computed tomography," *CoRR*, vol. abs/1811.04602, 2018.

[14] A. de Brébisson, É. Simon, A. Auvolat, P. Vincent, and Y. Bengio, "Artificial neural networks applied to taxi destination prediction," *CoRR*, vol. abs/1508.00021, 2015.

[15] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," *CoRR*, vol. abs/1602.04938, 2016.

[16] Z. Tianyu, M. Zhenjiang, and Z. Jianhu, "Combining cnn with hand-crafted features for image classification," in *2018 14th IEEE International Conference on Signal Processing (ICSP)*, pp. 554–557, Aug 2018.

[17] D. D. Thang and T. Matsui, "Image transformation can make neural networks more robust against adversarial examples," *CoRR*, vol. abs/1901.03037, 2019.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[19] H. Huang, R. He, Z. Sun, and T. Tan, "Wavelet-srnet: A wavelet-based cnn for multi-scale face super resolution," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 1698–1706, Oct 2017.

[20] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I–I, Dec 2001.

[21] "File:vj featuretypes.svg." https://commons.wikimedia.org/wiki/File:VJ_featureTypes.svg.

[22] "Face detection using haar cascades." https://docs.opencv.org/3.4/d2/d99/tutorial_js_face_detection.html.

[23] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893 vol. 1, June 2005.

[24] "Using histogram of oriented gradients (hog) for object detection." https://iq.opengenus.org/object-detection-with-histogram-of-oriented-gradients-hog/.

[25] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov 2004.

[26] A. Mikolajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," in *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, pp. 117–122, May 2018.

[27] A. Gidudu, G. Hulley, and T. Marwala, "Classification of images using support vector machines," *CoRR*, vol. abs/0709.3967, 2007.

[28] "Support vector machine python example." https://towardsdatascience.com/support-vector-machine-python-example-d67d9b63f1c8.

[29] "Understanding the mathematics behind support vector machines." https://shuzhanfan.github.io/2018/05/understanding-mathematics-behind-support-vector-machines/.

[30] "Image recognition and object detection : Part 1." https://www.learnopencv.com/image-recognition-and-object-detection-part1/.

[31] M. Ferguson, R. Ak, Y. T. Lee, and K. H. Law, "Detection and segmentation of manufacturing defects with convolutional neural networks and transfer learning," *CoRR*, vol. abs/1808.02518, 2018.

[32] D. Soukup and R. Huber-Mörk, "Convolutional neural networks for steel surface defect detection from photometric stereo images," in *Advances in Visual Computing* (G. Bebis, R. Boyle, B. Parvin, D. Koracin, R. McMahan, J. Jerald, H. Zhang, S. M. Drucker, C. Kambhamettu, M. El Choubassi, Z. Deng, and M. Carlson, eds.), (Cham), pp. 668–677, Springer International Publishing, 2014.

[33] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, pp. 1798–1828, Aug 2013.

[34] "2d convolution in keras." https://missinglink.ai/guides/keras/keras-conv2d-working-cnn-2d-convolutions-keras/.

[35] "Discrete convolution." https://www.sciencedirect.com/topics/computer-science/discrete-convolution.

[36] "Math behind convolutional neural networks." https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9.

[37] S. Y. Jung, Y. H. Tsai, W. Y. Chiu, J. S. Hu, and C. T. Sun, "Defect detection on randomly textured surfaces by convolutional neural networks," in *2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 1456–1461, July 2018.

[38] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[39] "Illustrated: 10 cnn architectures." https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#e276/.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[41] "Stanford cs class cs231n: Convolutional neural networks for visual recognition." http://cs231n.github.io/.

[42] "Activation functions in neural networks." https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[43] "Ukko2." https://wiki.helsinki.fi/display/it4sci/Ukko2+User+Guide.

[44] "Inverse problems research group." https://www.helsinki.fi/en/researchgroups/inverse-problems.

[45] "An end-to-end open source machine learning platform." https://www.tensorflow.org/.

[46] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.

[47] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[48] "Object detection: speed and accuracy comparison (faster r-cnn, r-fcn, ssd and yolo)." https://mc.ai/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo

[49] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *CoRR*, vol. abs/1612.03144, 2016.

[50] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015.

[51] N. Ketkar, "Stochastic gradient descent," in *Deep learning with Python*, pp. 113–132, Springer, 2017.

[52] Z. Zhang and M. R. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," *CoRR*, vol. abs/1805.07836, 2018.

[53] M. Misiti, Y. Misiti, G. Oppenheim, and J.-M. Poggi, *Wavelets and their Applications*, vol. 330. Wiley Online Library, 2007.

[54] D.Ravichandran and R. Nimmatoori, "Mathematical representations of 1d, 2d and 3d wavelet transform for image coding," *ISSN*, vol. Volume -5, Issue -3, 2016.

[55] J. Baker, "Quantitative classification of near-fault ground motions using wavelet analysis," *Bulletin of The Seismological Society of America - BULL SEISMOL SOC AMER*, vol. 97, pp. 1486–1501, 10 2007.

[56] A. Haar, "Zur Theorie der orthogonalen Funktionensysteme," *Mathematische Annalen*, vol. 69, no. 3, pp. 331–371, 1910.

[57] "Inverse problems 1 course autumn 2018 by tatiana bubba." https://www.helsinki.fi/en/unitube/video/7a8fac34-384f-445e-ad85-d23fe3fba1d8.

[58] "Image enhancement." https://eclipse.github.io/imagen/guide/image-enhance/.

[59] H. Heidary, A. Refahi Oskouei, M. Hajikhani, B. Moosaloo, and M. Ahmadi Najafabadi, "Acoustic emission signal analysis by wavelet method to investigate damage mechanisms during drilling of composite materials," vol. 1, 07 2010.

[60] P. Getreuer, "Rudin-Osher-Fatemi Total Variation Denoising using Split Bregman," *Image Processing On Line*, vol. 2, pp. 74–95, 2012.

[61] "Johdatus valokuvan matematiikkaan." https://courses.helsinki.fi/fi/mat21022/123885328.

[62] T. Williams and R. Li, "Advanced image classification using wavelets and convolutional neural networks," 12 2016.

[63] W. Li, S. Manivannan, S. Akbar, J. Zhang, E. Trucco, and S. J. McKenna, "Gland segmentation in colon histology images using hand-crafted features and convolutional neural networks," in *2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI)*, pp. 1405–1408, April 2016.

[64] B. Ãetiner, M. Sari, and O. Borat, "A neural network based traffic-flow prediction model," *Mathematical and Computational Applications*, vol. 15, p. 269â278, Apr 2010.

[65] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," *CoRR*, vol. abs/1804.07612, 2018.

[66] C. Wu, W. Liu, and X. Guo, "Adaptive total variation model for image denoising based on modified orientation information measure," in *2010 3rd International Congress on Image and Signal Processing*, vol. 2, pp. 616–620, Oct 2010.

[67] T. G. Dietterich, "Ensemble methods in machine learning," in *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, (London, UK, UK), pp. 1–15, Springer-Verlag, 2000.

[68] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015.

Appendix A.

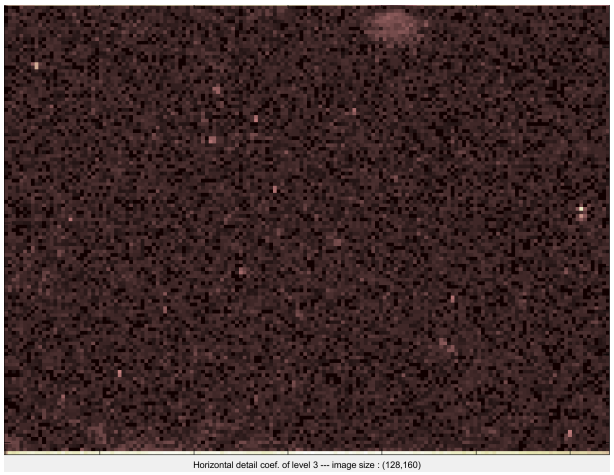

Figure A.1: Haar Wavelet Horizontal 3rd level Coefficient

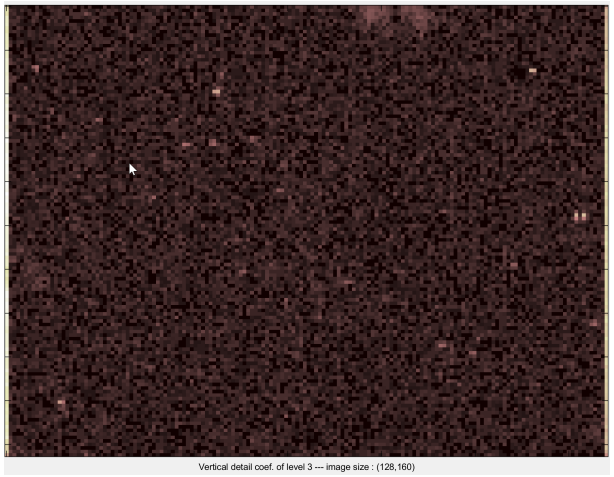

Figure A.2: Haar Wavelet Vertical 3rd level Coefficient

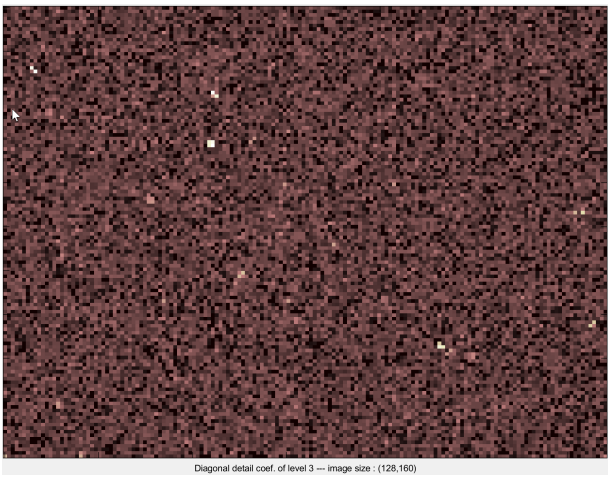

Figure A.3: Haar Wavelet Diagonal 3rd level Coefficient

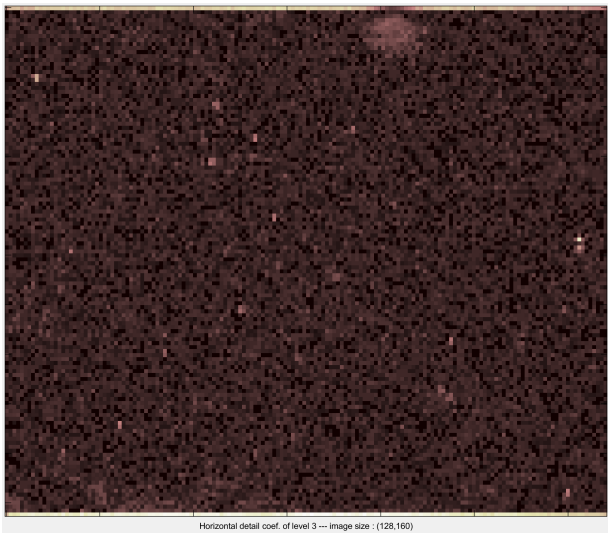

Figure A.4: Daubechies Wavelet Horizontal 3rd level Coefficient

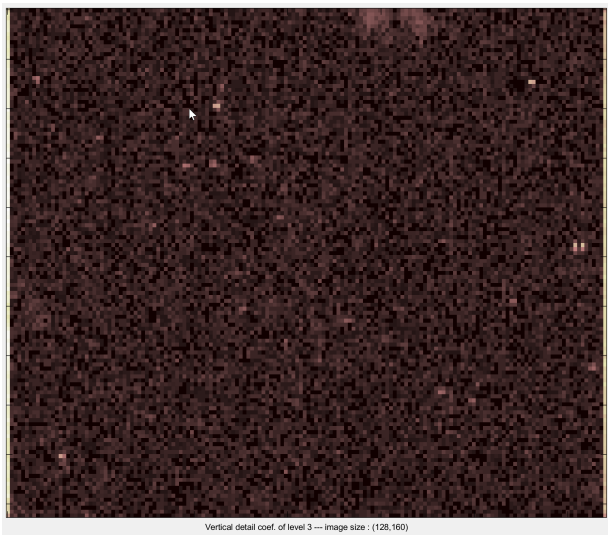

Figure A.5: Daubechies Wavelet Vertical 3rd level Coefficient

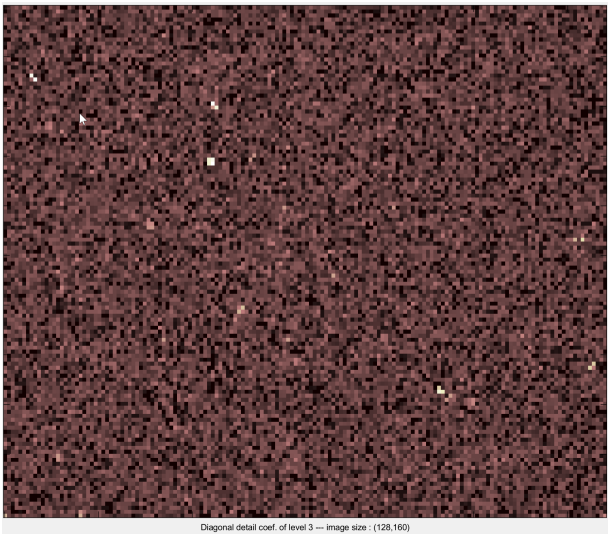

Figure A.6: Daubechies Wavelet Diagonal 3rd level Coefficient

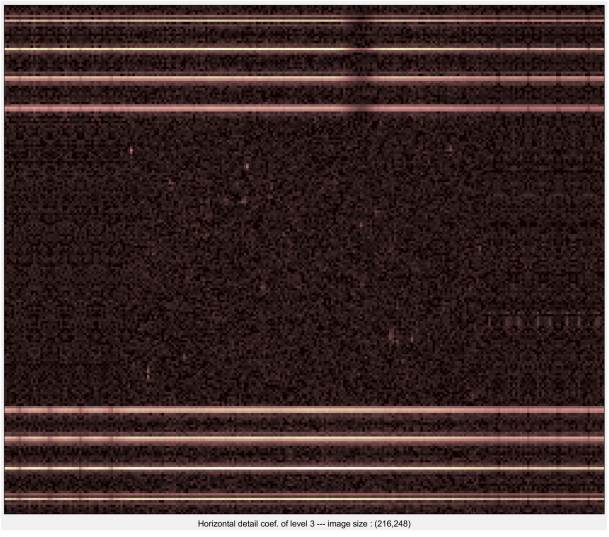

Figure A.7: Discrete Meyer(dmey) Wavelet Horizontal 3rd level Coefficient

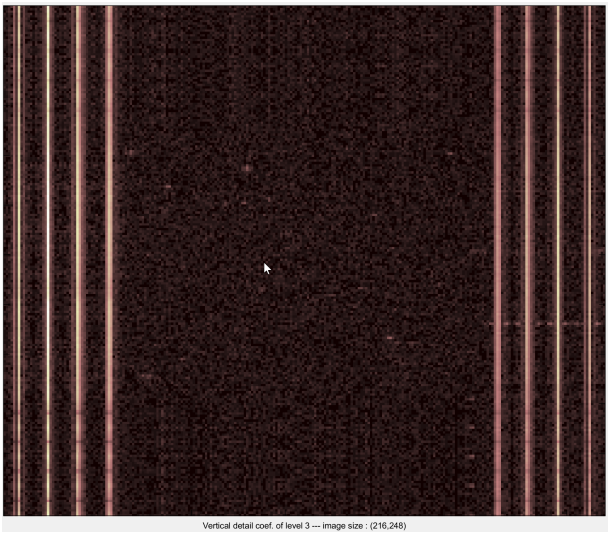

Figure A.8: Discrete Meyer(dmey) Wavelet Vertical 3rd level Coefficient

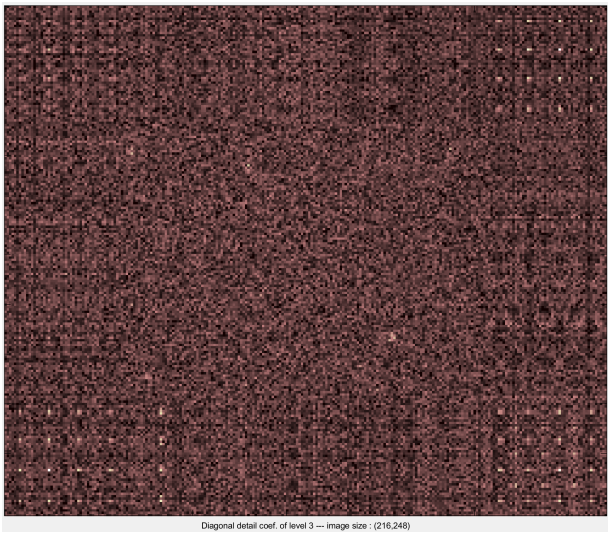

Figure A.9: Discrete Meyer(dmey) Wavelet Diagonal 3rd level Coefficient