Kaggle Competition Project

Severstal: Steel Defect Detection

CZ4041 – MACHINE LEARNING

(Semester 2, AY 2019/2020)

Source Code Repository:
Github Link

LOW YU BENEDICT:	U1821762E
SWA JU XIANG:	U1822040G
TEO BOON SHUAN:	U1821709J
WILSON THURMAN TENG:	U1820540H
KOH TAT YOU @ ARTHUR KOH:	U1821604B

Table of Contents

1. Introduction

The fourth industrial revolution will bring about a monumental increase in data collected and transmitted throughout the world. The world is moving into the entry stages of the fourth industrial revolution, and the amount of data that currently exists in the world is overwhelmingly huge. According to the Internet Center (IDC), the total amount of global data is expected to reach 42ZB in 2020. More than 70% of that global data is expected to be transmitted by image or video, which requires the use of computer vision to extract useful information. The extraction of useful information from the transmitted image or video data worldwide is a problem domain area present in numerous fields. These problem domain areas are frequently represented as competitions on Kaggle, and our group has identified one such competition for our project, which focuses on *image classification and segmentation*.

The roles of our team are as such:

Low Yu Benedict	Developer and Technical Writer
Swa Ju Xiang	Developer and Researcher
Teo Boon Shuan	Technical Writer
Wilson Thurman Teng	Team Lead, Developer and Researcher
Koh Tat You @ Arthur Koh	Technical Writer and Researcher

1.1. Problem Statement

The objective of the Kaggle competition is to detect and classify steel manufacturing defects using the training images provided by Severstal. Each provided image is named and uniquely identified by an *ImageId*. Our task involves two components. The first component involves the detection of defects through segmentation performed on the provided images. The second component involves the classification of defects in the test set of provided images. The competition evaluates the accuracy of our model using the Dice Coefficient:

$$\frac{2 \times |X \cap Y|}{|X| + |Y|}$$

1.2. Challenges of the problem

Imbalanced Dataset

The training dataset provided was imbalanced. The number of defect classes given in the training dataset were distributed as such:

a. No Defects: 5902
b. Defect Class 1: 897
c. Defect Class 2: 247
d. Defect Class 3: 5150
e. Defect Class 4: 801

Due to the imbalanced nature of this dataset, a typical model trained using this dataset might face difficulties with predicting the minority defect classes. In this instance, it would be defect classes 1, 4 and especially 2.

Complex Problem

A typical competition on Kaggle, like the *Airbus Ship Detection Challenge* or the *TGS Salt Identification Challenge* only requires a submitted model to identify pixels required by the competition challenge. This was not the case for the *Severstal Steel Defect Detection Competition*. For our chosen challenge, we were required to not just submit a model to identify the defect pixels, but also the defect class of the those identified defect pixels. This additional requirement thus presents a challenge due to the additional complexity of the problem.

Interpretation of feature vectors

The images provided in this competition contains features with physical meanings that can be interpreted given enough domain knowledge from the field of steel manufacturing. These interpretations are useful for data processing and feature engineering. However, as our team lacks the required domain knowledge, the possible data processing and feature engineering that we could perform was limited.

Nonetheless, there were some generic feature engineering that we attempted. This will be discussed further in Section 3: Pre-Processing.

Competitive leaderboard

At the time of this writing (17th Apr 2020), the competition has attracted a grand total of 2431 submissions. The leaderboard score (LB) between the different submissions vary little as the LB goes up. In fact, the top 126 entries varied only at scores of 0.90X where X would be the difference in the LB. For our initial attempts at this competition, we only managed to achieve a score of 0.7-0.8 on the leaderboard.

2. Data Description

For this competition, we were presented with 12,568 labeled grayscale images (training set), and 5,506 unlabeled grayscale images (testing set) of steel defects. All images were in the resolution of 1,620 × 256.

The training set was labeled with the defect class (ClassId = [1, 2, 3, 4]) that the image falls into. Unlabeled images included in the training set were thus considered to be non-defective images. As such, each image falls into either one of three categories: *(1) No Defects*, *(2) Single Defect* or *(3) Multi Defects*.

In our exploratory data analysis (EDA) the maximum number of defects (Multi Defects images) in a single image was 3. A graph displaying the number of defects per image is shown in Figure 1 below:

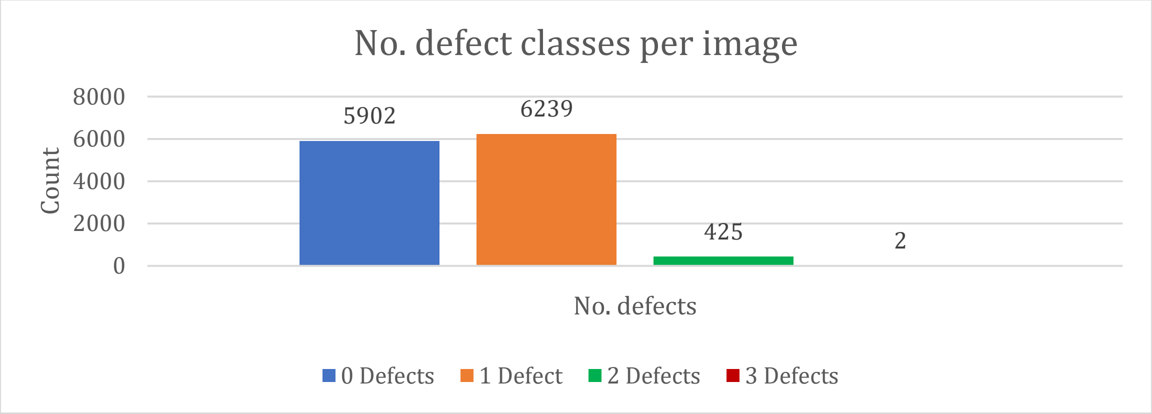


Figure 1: Number of defect classes per image

It was also discovered that pixels identified as defects were not localized and could exist in any segment of the image. This was true for both multi defect images and single defect images.

As mentioned earlier in Section 1, we also discovered that the training set was heavily imbalanced; with Class 3 defects accounting for almost 40% of the total training set, and images with no defect accounting for 45% of the total training set. Conversely, Class 2 defects only accounted for 1.9% of the training set. This imbalance indicated a high possibility that our model might struggle with classifying Class 2 defects – or for the matter, other classes of defects as well. A graph displaying the number of defects for each class is shown in Figure 2.

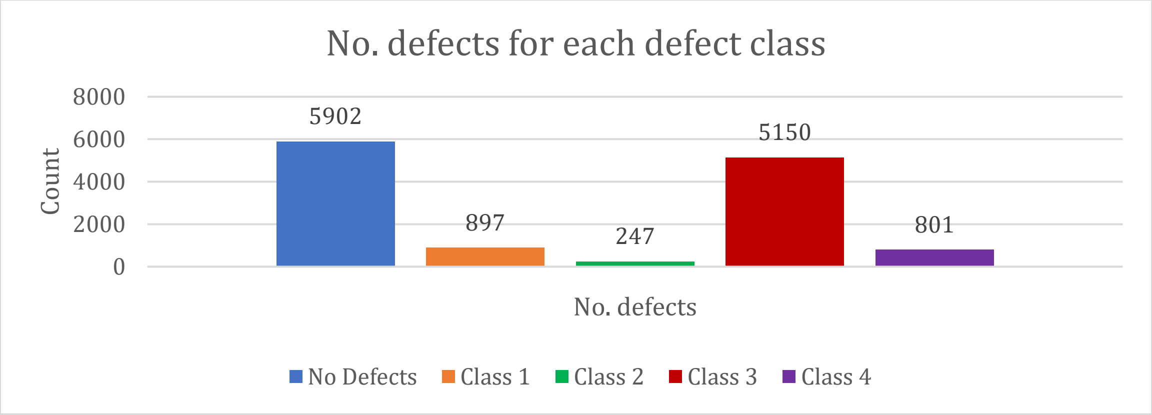


Figure 2: Number of defects for each defect class

3. Image Processing

3.1. Image Data

The image dataset consists of images of the same size, 1,600 × 256 (W x H). Pictured below is an example of the images provided:

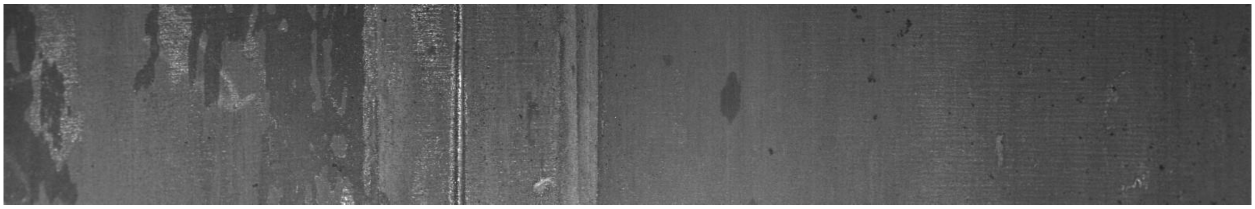


Figure 3: Example of a datapoint – an image taken from the dataset

The objective of our competition is to identify the presence of steel defects in the provided images, before classifying these defects into 1 of 4 defect classes. These defects can occur in any part of the image. As such, the use of a Convolutional Neural Network (CNN) will be a suitable solution for this scenario. To be precise, we eventually settled on using U-Net, a CNN variant that excels at creating segmentation masks.

3.2 Basic Pre-Processing

Pre-processing is normally used in conjunction with CNNs (including U-Net) to improve the quality of data input. As an initial step, we performed two forms of initial pre-processing, namely: *horizontal flip* (with 0.5 probability) and *image normalization* (with 1.0 probability).

Horizontal Flip

A horizontal flip is a data augmentation technique used to synthetically generate additional modified data. This generated data is meant to increase the amount of relevant data, which can directly improve the effectiveness of a CNN. There are two ways of applying horizontal flip: *offline augmentation* and *online augmentation*.

- Offline augmentation on the other hand, involves applying horizontal flip operations on all images in the dataset. This doubles the size of the dataset.
- Online augmentation involves applying horizontal flip operations in mini batches which are fed into a CNN. During each epoch, different image variants will be used for model training, and is usually used for larger datasets to prevent the need for saving augmented images in a storage medium.

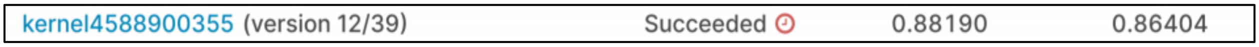
In context to the Severstal Dataset, we opted for the use of *online augmentation*, where a horizontal flip is performed in mini batches that are fed into our U-Net model. The reason is due to memory limitations of the provided Kaggle compute GPUs.

Image Normalization

Image normalization is a process the changes the range of pixel intensity values in any given image. These pixel intensity values are integers that range from 0 to 255. Unfortunately, large value inputs are not suitable for CNNs as these large values slow down the learning process of a CNN. As such, it necessary to change the scale of the range to a smaller range, from 0 to 1 to preserve the learning speed of a CNN.

In context to the Severstal Dataset, the provided images all fit the profile of images with pixel intensity values that range from 0 to 255. As such, we applied image normalization on all the provided images before feeding into our U-Net model for learning.

Using both pre-processing techniques, we managed to train a U-Net model that achieves a private score of **0.88190**.



kernel4588900355 (version 12/39) Succeeded ⊙ 0.88190 0.86404

Figure 4: Score achieved by U-Net Model after submission to Kaggle

We used this score as the baseline metric to evaluate the use of other, more advanced pre-processing techniques. These advanced pre-processing techniques will be discussed in the following sections:

3.3. Advanced Pre-Processing

3.3.1. Contrast-Limited Adaptive Histogram Equalization (CLAHE)

Overview

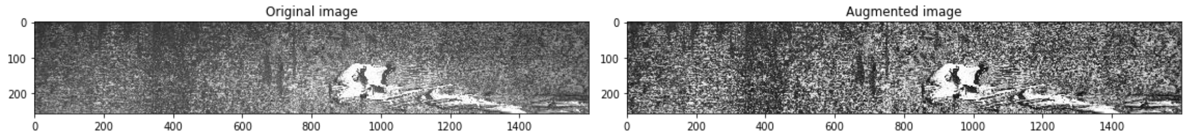
CLAHE is a variant of Adaptive Histogram Equalization (AHE) technique. AHE is an image pre-processing technique used to improve contrast in images. This is achieved through the computation of several histograms, which each correspond to a certain section of an image [1]. These computed histograms are then used to redistribute the brightness values of an image, which improves local image contrast and enhances the definition of edges in an image.

Unfortunately, as there is a tendency for AHE to overamplify the local image contrast in near-constant areas of the image, noise might be amplified in near-constant regions.

To address this problem of undesired noise amplification in near-constant regions, CLAHE was an alternative technique proposed. It works by limiting contrast amplification through the clipping of the histogram at a predefined value before computing the CDF. This limits the slope of the CDF and the transformation function, which limits undesired noise amplification [1].

Rationale

The use of CLAHE was necessitated by the need to properly pre-process data for our U-Net model. In context to the Severstal Dataset, we wanted to improve the contrast in the provided images to improve the images fed into our U-Net model so that defect features could be more accurately detected. An example of this is shown below:


Original image

Augmented image

Figure 5: Augmentation of image with CLAHE

Outcome

With the use of CLAHE, we obtained a private score of 0.85954. Unfortunately, this was a decrease from the initial private score of 0.88190.

3.3.2. Detrending

Overview

Detrending is commonly used to remove a trend from a time series. This is a temporal detrending operation used to remove temporal bias. In our case, we used a variant called *spatial detrending*, which is used to remove spatial bias from images. This is accomplished using an image's spatial coordinates to remove spatial trends.

Rationale

Spatial bias in images occur often due to differences in depth from certain parts of a subject to the measurement instrument used to capture an image [2]. In context to Severstal Dataset, we wanted to minimize the possible spatial bias in the provided images due to these variations (E.g. certain defects might be located physically deeper within a piece of steel, which would result in would result in darker defect pixels). To model this spatial bias and remove it, we will assume that changes across the entire extent of the provided images are due to this bias [2].

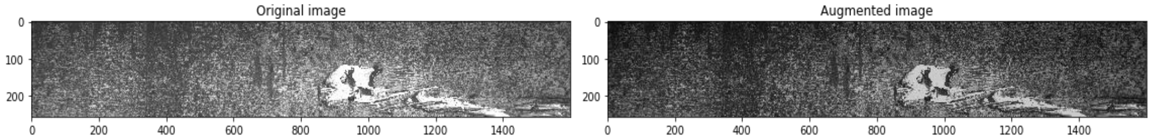
Outcome

With the use of detrending, we obtained a private score of 0.86551. Unfortunately, this was a decrease from the initial private score of 0.88190.

3.3.3. Random Brightness and Contrast

Overview

Adjusting the brightness and contrast of images are common data augmentation techniques. By adjusting the brightness and contrast of the input images it encourages our model to be trained to identify steel defects in all conditions and not just in optimal lighting. As such, the randomizing of brightness and contrast was performed with the intention of training a more robust and versatile model. An example of this is shown below:


Original image

Augmented image

Figure 6: Augmentation of image by adjusting brightness and contrast

Rationale

As this Kaggle competition does not disclose the private test datasets used, we had to consider the possibility that the brightness and contrast of the private test datasets may not be similar to the training dataset provided to us. Randomizing the brightness and contrast would allow us to prepare for such a scenario.

Outcome

Using random brightness and contrast resulted in a private score of 0.86359. Despite our expectations, this was lower than the initial private score of 0.88190. As such this pre-processing technique was not applied in our submitted code.

3.3.4. Random Grid Shuffle

Overview

As the name implies, random grid shuffling segments each image by a grid of a fixed length and shuffles these grids. Random grid shuffling is done with the intention of allowing the trained model to predict more accurately as these grid sizes would ideally encapsulate only the wanted pixels for the predicted label, thus allowing the model to easily identify features unique to the wanted predicted labels. An example of this is shown below:

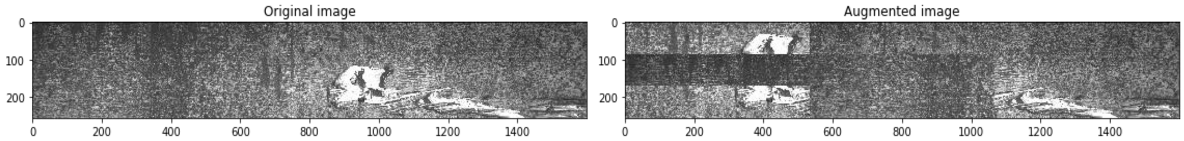


Figure 7: Augmentation of image by applying Random Grid Shuffle

Rationale

In the case of this competition, the Random Grid Shuffle was adopted as defects tended to exist in clusters. As such it was thought that perhaps the random grid shuffle could allow for these clusters to be placed in grids, thus presenting more clearly to our model the features it needs to learn.

Outcome

The random grid shuffle resulted in a private score of 0.86895. This was lower than the initial private score of 0.88190. One likely reason could be that the grid sizes were suboptimal, especially since the various defect classes were visually different. While adjusting the grid sizes for each defect class might result in a better performing model, we lacked the computational and time resources to investigate this possibility further. As such, we did not use this in our submitted code.

3.3.5. Random Rain

Overview

As the name suggests, random rain performs image pre-processing by adding rain effects to the image. We got this idea from a GitHub user, UjjwalSaxena (https://github.com/UjjwalSaxena/Automold--Road-Augmentation-Library), who used these weather effects to train a convolutional neural network that helps with driving on the road. [3]

Rationale

This technique was used as the training images seemed to be rather similar to rain – long thin strokes of lines on each image. As such, it was considered that perhaps adding rain effects to the input images could train the model to be more robust in identifying defects as only the most important features in a defect would stand out, amongst the lines and strokes of the steel and rain.

Outcome

Using random rain resulted in a private score of 0.86895. This was lower than the initial private score of 0.87346. As such, this pre-processing technique was not applied in our submitted code.

3.3.6. Mask Dropout

Overview

Mask Dropout is an image pre-processing technique that works using the same principles of the dropout regularizations used in neural network. Essentially for this pre-processing technique, regions where a mask exists are sometimes randomly removed prior to passing the image into the network as an input.

The goal of performing mask dropout is to create a more robust model able to identify the defect features with minimal reliance on the surrounding pixels of a defect. Essentially a model can be trained that is less sensitive to specific contextual information (similar to the notion of avoiding overfitting), and better at generalizing the behaviors of the expected predicted masks.

The mask dropout technique was inspired by Kaggle user R Guo (https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/114254) who won the first-place submission in the Severstal Steel Defect Detection Competition.

Rationale

In the case of this competition, mask dropout is an image pre-processing technique that seemed very viable. As steel defects in the training images often shows up in clusters, dropping the masks encourages the model to better understand the underlying features of each defect class, rather than the contextual surrounding pixels.

Outcome

Using the mask dropout technique resulted in a more consistent private score (0.87079) and public score (0.86860). However, the private score was still lower than the initial private score of 0.88190. As such, it was not used in our submitted code.

3.4. Summary of Pre-processing Techniques

Despite the use of 6 different advanced pre-processing techniques, including combinations of said techniques, we were not able to improve our private score for our submitted code.

One reason why this might be the case could be because the provided images by Severstal were already captured using high quality cameras and stringent capturing parameters that eliminate the need for pre-processing, as such techniques are meant to only be useful for suboptimal image datasets.

To explore other options that could improve our score, we decided to focus our efforts on post-processing techniques. This will be discussed in the next section.

3.5. Post-Processing

3.5.1 Adjusting Threshold Value

For our model, a threshold value (ranging from 0 to 100) is used to determine if an image should be considered a defect or not. An image with a score greater than the threshold value would be considered not defective, while an image with a score lower than the threshold value would be considered defective.

While the threshold value was initially set to 50, we decided to investigate the possibility of discerning a threshold value that would better differentiate the defective and non-defective images.

To access the effectiveness of the threshold value, we compared the threshold value to the metrics (i) Accuracy, (ii) Recall, (iii) Precision and (iv) F1 score. The formula for each metric is given as such:

Accuracy	**Recall**	**Precision**	**F1 Score**
$\frac{TP + TN}{TP + TN + FP + FN}$	$\frac{TP}{TP + FP}$	$\frac{TP}{TP + FP}$	$\frac{2 \times Recall \times Precision}{Recall + Precision}$

Table 1: Four metrics and the respective formulas

We then plotted a graph of the various metrics against the threshold value, as seen in Figure 8.

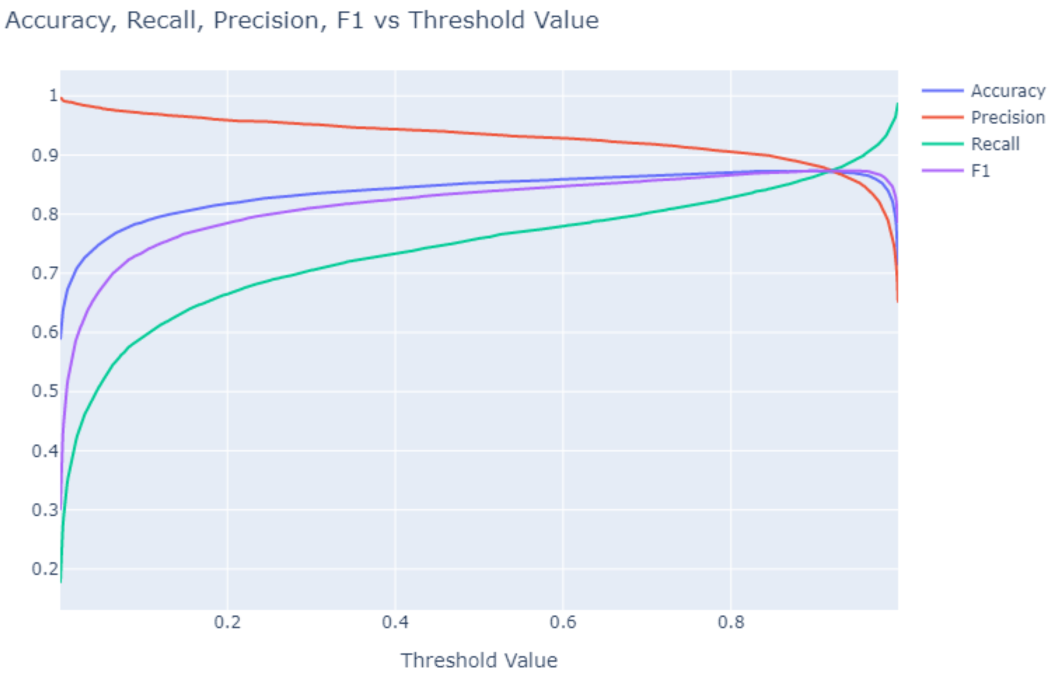


Figure 8: A plot of the different metrics against the Threshold Value

Subsequently, it was discovered that the threshold value that resulted in the most consistent score for all four metrics was 0.919. The table below shows the respective scores and the threshold value.

Threshold Value	Non-Defect Images (%) Above Threshold	Defective Images (%) Below Threshold	Accuracy	Recall	Precision	F1 Score
0.919	0.849373	0.873714	0.872594	0.873714	0.874577	0.874145

Table 2: Threshold value along with the various metrics

3.5.2 Reducing False Positives

We discovered that false positives were heavily penalized when images without defects were classified as images with defect(s). In fact, should a non-defective image classify a single pixel as defective (and thus classify the image as defective), this image would receive a score of 0. Conversely, if the same image hadn't misclassified that single pixel as defective, the image would have scored a 1.

As such it was evident that minimizing False Positives would enable the submission to score better. With that understanding in mind, we set a value of 3500 pixels as a threshold value. Any images classified as a defect with a total pixel count of less than 3500 pixels in the mask would thus be classified as non-defective.

After using this post processing technique, the private score improved from an initial score of 0.88190 to 0.88536. Moreover, this technique resulted in an increase in private and public scores for every other previous attempt we had. With such promise this post processing technique used, we adopted it into our submitted code.

4. Models

Choice of Model

For the provided Severstal image dataset, we have decided to use a Convolutional Neural Network (CNN). The reason for this choice is due to the 2 objectives of the Kaggle competition:

1. Defect identification of steel defect pixels in the provided image dataset
2. Defect classification of previously identified steel defect pixels

Tackling the two above-mentioned objectives effectively requires a model that can develop an *internal representation* of the provided image dataset. In this regard, a CNN would be uniquely suited for this task. To explain why, we will be explaining the reason why in two sections. These two sections are generally outlined as:

1. Demands of the Severstal Dataset
2. Feature Learning: MLPs vs CNNs

Demands of the Severstal Dataset

The provided Severstal Dataset consists of images which require identification and classification of steel defect pixels. This means that *steel defect features*, which are specific structures in the provided images from Severstal (e.g. points, edges or objects), must be identified. In order to identify these steel defect features, there are 2 ways of doing this:

Feature Engineering

Is the process of using domain knowledge to identify features from raw data through data mining. In context to the Severstal dataset, this would usually require a person to manually use computer vision techniques and domain knowledge (from a steel industry domain expert) to create a detector for identifying the steel defect features. [4]

Feature Learning

Is the process of using a set of techniques to automatically discover internal representations from raw data through a mathematical transformation. These representations are meant to be more suitable for machine learning and are needed to create detectors for feature identification and classification. In context to the Severstal dataset, supervised feature learning is used to create a detector by training a model for identification of the steel defect features. [5]

While both ways can be used to identify steel defect features, feature engineering is less effective than feature learning. This is because feature engineering relies on explicit algorithms. These explicit algorithms serve as the basis for feature engineering detectors, which suffer from 2 problems: they tend to be either *too general* (too simple), or *too over-engineered* (overfitted). Both problems mean that feature engineering will have issues with effectively identifying steel defect features in the provided Severstal dataset.

The reason for this is because steel defect features cannot be explicitly defined and discovered through algorithms, which would have been effectively tackled by feature engineering detectors. Addressing this requires a way to discover these steel defect features *without* relying on explicit algorithms. In this regard, feature learning detectors are particularly well suited as they can discover features without the use of explicit algorithms.

Feature Learning: MLPs vs CNNs

We have previously established that feature learning was crucial for the effective classification and identification of steel defect features. Thus, we will need to implement a neural network that uses feature learning. We will investigate how feature learning is used by 2 types of neural networks for image classification: *Multi-Layer Perceptrons* (MLPs) and *Convolutional Neural Networks* (CNNs). [6]

Feature Learning in MLPs

MLPs consist of at least 3 layers of nodes: an input layer, an output layer, and multiple hidden layers sandwiched between the input and output layers. All these layers *are fully connected*. If we zoomed into a singular hidden layer node, it would be represented as:

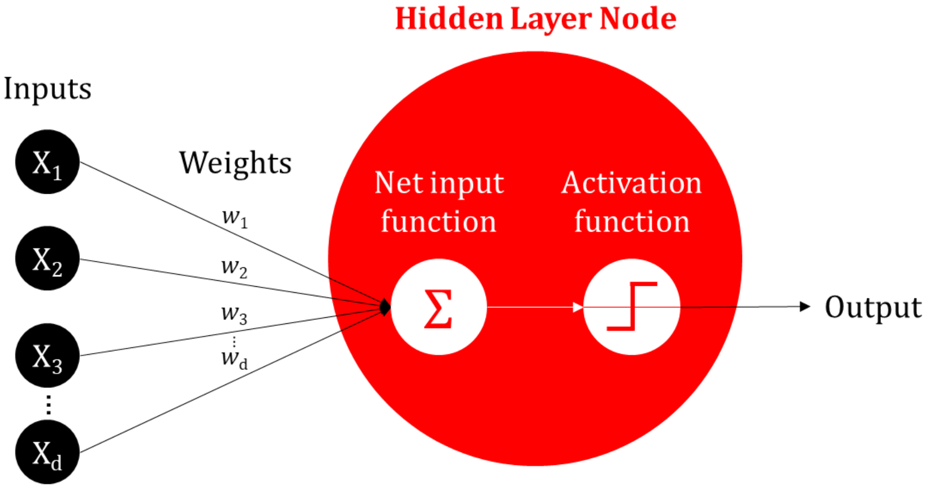


Figure 9: Architecture of a singular hidden layer MLP node

The *activation function* of each hidden layer node is a mathematical equation that determine the output of a node. This is dependant on how relevant an input is for a predicted value. There are many kinds of activation functions, including but not limited to: binary step, sigmoid and Rectified Linear unit function (ReLU). [6]

When expanded out to a network containing layers of nodes, these nodes would be part of larger whole that can be illustrated as such:

Perceptron

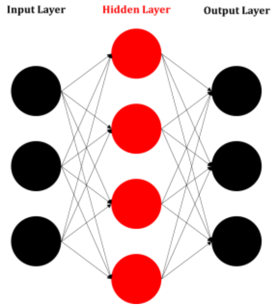


Figure 10: Single Layer Neural Network

Each layer consists of multiple neurons. A single-layer neural network is called a *perceptron*.

Multi-Layer Perceptron

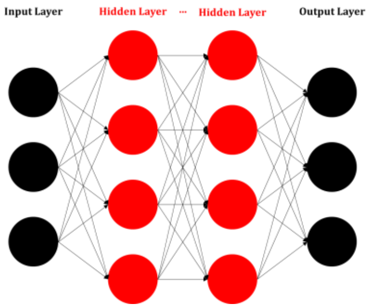


Figure 11: Multi-layer Neural Network

When this model has multiple hidden layers, it is called a *multi-layer perceptron*.

Each input into an MLP must correspond to a pixel in an image, as the weights in an MLP are not shared due to the *fully connected nature* of its hidden layers. In context to the Severstal image dataset, each picture from the dataset has a resolution of 1,600 × 256 pixels. This would mean that processing a singular image would require the training of a staggering 409,600 weights. The training of so many weights results in an exceedingly complex model that is prone to overfitting.

Because the weights in MLPs are not shared, it also means that MLPs are not *invariant*. *Invariance* is a necessary characteristic for effective feature identification in neural networks, as it allows features to be identified even if there are variations in its appearance. Examples of these appearance variances include translation (location), rotation, size and even illumination (brightness) of the features. In context to the Severstal image dataset, *invariance* is important for identifying steel defect features, as steel defects are not necessarily localized to a singular area, and can in fact, appear in any part of any given image.

While MLPs are theoretically able to accurately identify features even without weight sharing, this can only be achieved in a brute force manner with an immensely large dataset. As the provided Severstal image dataset is limited in size, it would not be possible to train such an MLP. In fact, we have already faced memory limitations with training a CNN, which can accurately identify features even with the use of a smaller image dataset. Thus, for the same dataset size, MLPs will struggle with identifying steel defects as they are not invariant.

Feature Learning in CNNs

CNNs are similar in overall structure to MLPs. In fact, CNNs are a variant of MLPs. As such, CNNs also consist of an input layer, an output layer and multiple hidden layers sandwiched between the input and output layers. This can be illustrated as such: [7]

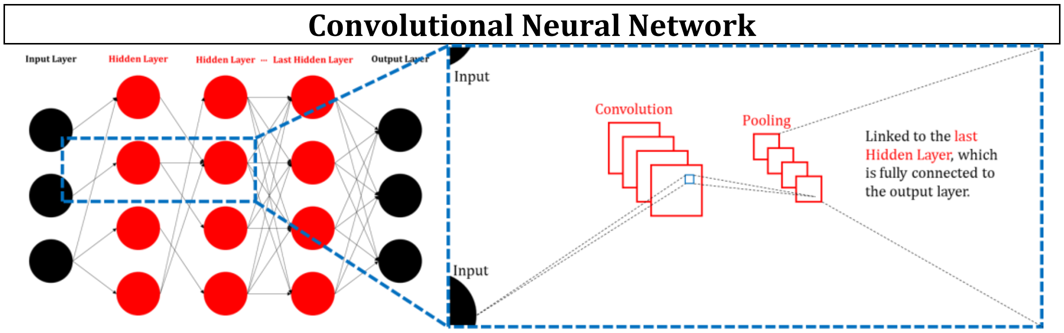

Convolutional Neural Network
Input Layer
Hidden Layer
Hidden Layer ··· Last Hidden Layer
Output Layer
Input
Convolution
Pooling
Linked to the last Hidden Layer, which is fully connected to the output layer.
Input

Figure 12: Architecture of a simple CNN

Aside from the last hidden layer to the output layer, the hidden layers of a CNN are *not fully connected*. CNN hidden layers also typically consist of three other layer types: pooling layers, fully-connected layers and most importantly, *convolutional layers*.

Pooling layers are used to progressively reduce the spatial size of the represented image. This reduces the number of parameters that must be processed by the CNN, and in turn reduces computation amount that needs to be performed. To note, CNNs are already considered to be computationally intensive even with the use of pooling layers (in comparison to MLPs). [7]

Fully connected layers appear as the last layer in the hidden layers of a CNN. This last layer is fully connected to previous hidden layer, as well as the output layer. The role of a fully connected layer is to use the results generated from a prior convolution and/or pooling process to classify an image according to a label. [7]

Convolutional layers are used to perform convolution operations. A convolution operation in context to deep learning, involves the multiplication of a set of weights with an input. [7] This multiplication is performed by an array of inputs (representing an image) and a 2D array of weights. An array of weights is called a kernel, and it is applied systematically to each overlapping part of an image. In the process of doing so, a key distinction between MLPs and CNNs is apparent: the systematic kernel application leads to *weight sharing*, unlike MLPs.

This means that unlike MLPs, CNNs are *invariant*. Invariance is the characteristic that allows neural networks to discover features in any part of an image, as previously mentioned. In context to the Severstal dataset, this means that a CNN would be well suited for identifying steel defect features, as steel defects can appear in any part of any given image.

Since the inception of CNNs, there are variants of CNNs that have been developed to be better at certain tasks (e.g. segmentation). One such example is U-Net, which is the CNN variant we have decided to use for the Severstal Steel Defect Detection competition. The details of it will be elaborated in the next section.

U-NET

Rationale for choosing U-Net

U-Nets are a variant of CNNs that were developed for biomedical image segmentation. Traditional image processing models suited for identifying an output with localization worked by creating a network with sliding-window approach, predicting the class label by individual pixels, as well as the surrounding pixels as context (termed as a patch). This results in regions of labeled pixels that is considered the labeled output. [8]

This method utilized by traditional image process models unsurprisingly resulted in long computational time, while experiencing many redundant computations as a result of patches overlapping. Additionally, the patching size (for predictions of localization) had the problem where large patches reduced the localization accuracy as a result of requiring more layers. On the other hand, small patches lacked the context to accurately predict the localization of the expected output. [8]

Due to the reasons mentioned above, we decided to adopt the U-Net architecture instead as our team faced a lack of computational resources. Moreover, U-Net has already established itself as one of the best networks for localizing predictions in images. Since its inception in 2015, U-Net has been widely adopted for any prediction tasks requiring network localization.

U-Nets are highly effective when working on tasks where the output and input are of similar sizes, and the output can utilize the spatial resolution provided. As a result, this enables U-Nets to perform astonishingly well in the creation of segmentation masks. In context to the Severstal Steel Defect competition, these segmentation masks are essential as contestants are required to not just predict the defect pixels, but also label these pixels as one of the four given defects.

U-Net Architecture

U-Net derives its name from the architecture of this model. The architecture of the U-Net model consists of a contracting path, and subsequently an expansive path, thus an architecture in the shape of a "U". The goal of the *contracting path* is to capture the context of the input image to be used for segmentation further down the architecture. On the other hand, the goal of the *expansive path* is to allow for the precise localization and relevant contextual information provided from the contracting path [8].

Contracting Path (Encoder)

In the contracting path, each image in the Severstal Steel Defect Competition will be modeled repeatedly using 3 x 3 convolutions before reaching a rectified linear unit (ReLU) and then downsampled to a smaller resolution using a 2 x 2 max pooling operation with stride 2.

Whenever an image is downsampled (i.e. reduced in resolution using max pooling), the number of feature channels will be doubled [8]. This allows for U-Net to learn more complex features in the deeper layers, especially since a downsampled image implies an increase in the receptive field (with respect to the image size) for any feature channel.

Essentially, the contracting path aims to reduce the spatial information through downsampling, while increasing the feature information by doubling the feature channels.

Expanding Path (Decoder)

The expanding path can be thought of as the opposite of a contracting path. Where a contracting path downsamples and increase the feature information, the expanding path upsamples and reduces the feature information.

After each step of upsampling (a 2 x 2 up-convolution) is performed the number of feature channels is halved. This ensures that the output along the expanding path is an image with increased resolution but as a result would naturally lack some feature information (halved to be exact) [8].

To improve the performance loss as a result of this, the expanding path starts with a large number of feature channels during the upsampling process [8]. This done to preserve as much feature information as possible. Furthermore, the upsampling path also uses skip connections, which concatenate the output of the transposed convolution layers with that of the earlier mentioned contracting path's feature maps at the same level. After each concatenation, two regular convolutions are applied to allow for a more precise output.

The diagram below taken from this research paper [8] summarizes the U-Net architecture succinctly.

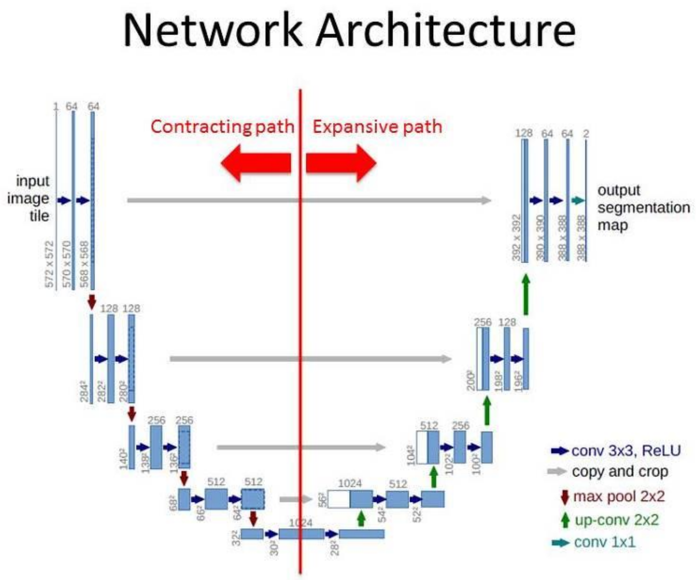


Figure 13: U-Net Architecture – Downsampling on the left and Upsampling on the right

Residual Networks (ResNets)

There are many variations of U-Net as researchers investigate various methods to enhance the performance of U-Net models. A variation we have adopted uses ResNet in the encoder of the U-Net architecture. In this section, we will discuss and explain the pros of using ResNets as well as the challenges that ResNets mitigates.

The Residual Networks (ResNets) convolutional neural network architecture first came into the spotlight in 2015, after winning 1st place in the ILSVRC 2015 Classification Competition, as well as the COCO 2015 Competition in ImageNet Detection, ImageNet Localization, Coco Detection and Coco Segmentation.

The ResNet architecture enabled the model to train networks with more layers without being hamstringed by the vanishing gradient problem – as the network depth increases, performance eventually plateaus and then starts degrading rapidly [9]. The vanishing gradient problem is caused when deep networks are trained, and since backpropagation uses the derivatives of each layer from the last to the first, a deep network results in a large number of derivatives of sigmoid functions multiplied together, and the gradient decreases exponentially (Figure 14) [10].

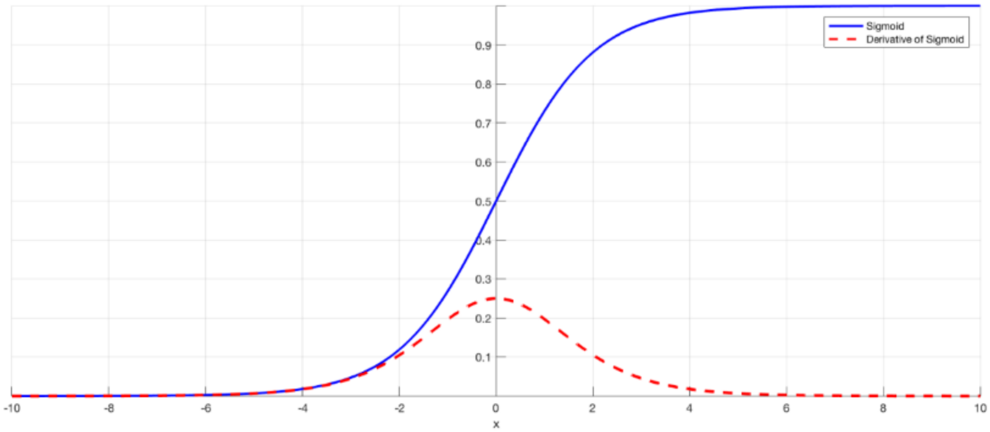


Figure 14: Demonstrating the Sigmoid Function and its Derivative

ResNet Architecture

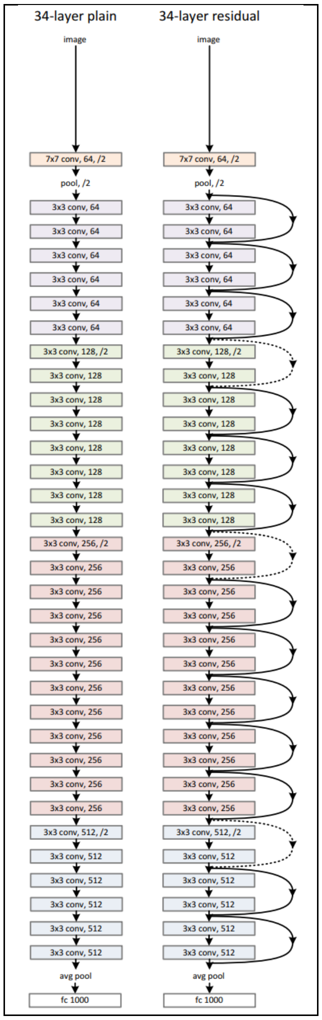


Figure 15: ResNet Architecture []

Figure 15 on the left shows the difference between the architecture of a traditional 34-layer network and a 34-layer residual network [9].

The solid arrows denote that the output and input are of the same dimensions, while the dotted arrows denote the contrary. Arrows essentially describes skip connections, which will be discussed in the next section.

Each building block in a ResNet architecture is defined by the equation [9]:

$$y = F(x, \{Wi\}) + x \qquad (1)$$

Where x and y are the input and output vectors of the layer, and the function F(x, {Wi}) denotes the residual mapping to be learned. This equation introduces no extra parameters or computation complexities, a trait that is extremely helpful in the field of deep learning. In Figure 14 Equation (1) is represented as the solid arrows.

The dotted arrows can be represented in this equation [9]:

$$y = F(x, \{Wi\}) + Wsx \qquad (2)$$

A new variable Ws is introduced to match dimensions by performing a linear projection. The simplicity of Ws also allows for equation (2) to use less computational resources. In Figure 14, Equation (2) is represented as the dotted arrows.

Simply put, ResNet uses simple building blocks that are highly economical yet effective.

Skip Connections

ResNets avoids the vanishing gradient problem entirely by using skip connections. Skip connections allows outputs from one layer to be passed into the input of both the immediate successive layer and some layer further down the network. This allows for information to be passed to successive layers before it becomes too abstract, thus tackling the vanishing gradient problem. Furthermore, the use of skip connections shortens the effective paths, and the use of deep paths are minimized or eliminated altogether. In fact, one study found that a residual network of 110 layers achieved its gradient from paths of only 10 to 34 layers deep [11].

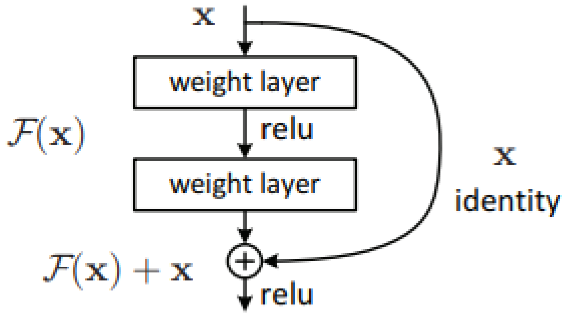
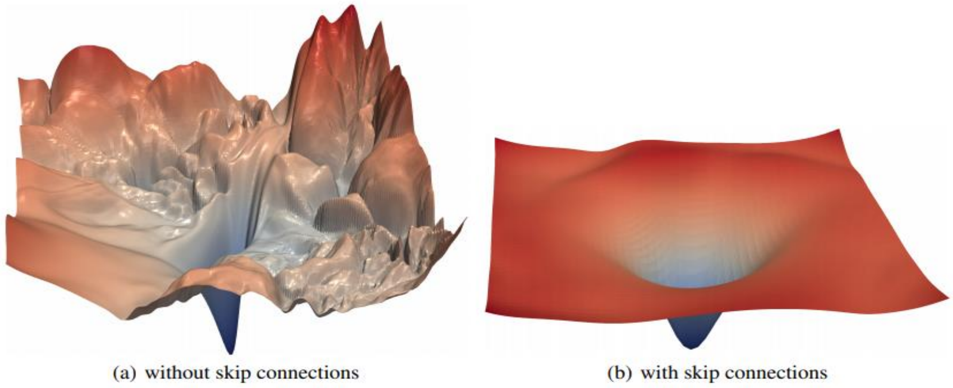


Figure 15: A building block of residual learning.

The study also discovered that the paths used in a residual network did not have strong dependency on each other, much akin to that of ensembles of networks. This suggests that ResNet could also be seen as an ensemble of relatively shallow networks. With this understanding we found the need to use an ensemble learning model to be unnecessary as ResNet already behaves as such.

Skip connections are also used in the encoder-decoder connection in the UNet architecture.

Ultimately, the use of skip connections mitigated the vanishing gradient problem and enabled researchers to train a residual network with a depth of 1001 layers that could outperform shallower ResNet models. Figure 16 visualizes the effectiveness of using skip connections in a ResNet-56 model [12].



(a) without skip connections

(b) with skip connections

Figure 16: Visualisation of not using skip connections vs using skip connections

Transfer Learning

Transfer learning is the usage of a pre-trained large-dataset trained model on a smaller dataset. These models are pre-trained using an immense dataset on unrelated categories from *imagenet*, which make them well optimized for images. This is because images share universal low-level features. In context to the Severstal Dataset, we have implemented transfer learning on the ResNet encoder within our U-Net pipeline.

We ran our U-Net model with a variety of pretrained ResNet model variants, namely: ResNet-18, ResNet-34, ResNet-50, ResNet-101 and ResNet-152. The numbers suffixed at the end of the model name indicates the number of layers used. Using deeper pre-trained ResNet models required more computational resources. As a result, we were unable to arrive at results for ResNet-101 and ResNet-152, due to the memory limitations of the GPU compute resources provided by Kaggle. The results are tabled below:

	Batch Size 8	Batch Size 12	Batch Size 14	Batch Size 18	Batch Size 20
ResNet-18					
ResNet-34	N.A.	0.85702	0.86433	0.86461	0.85334
ResNet-50	0.86068	0.86075	N.A	N.A	N.A
ResNet-101	N.A	N.A	N.A	N.A	N.A
ResNet-152	N.A	N.A	N.A	N.A	N.A

Table 1: Results according to No. Of Layers in ResNet and Batch Sizes

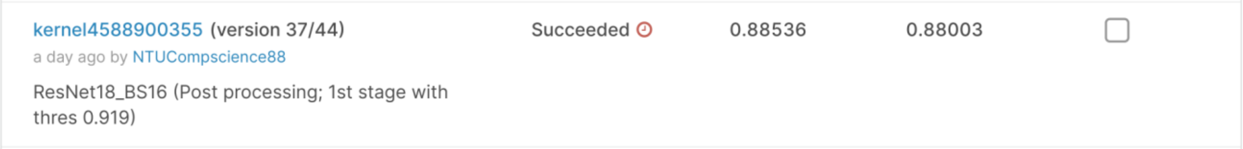
Final Implementation

As a result of the reliability of both U-Net and ResNet, we decided to use a pre-trained ResNet in our U-Net encoder structure. The pre-trained ResNet weights trained on imagenet were acquired from PyTorch.

The use of a pre-trained ResNet model allowed for more time to be spent on fine tuning our final model. Furthermore, as the pre-trained ResNet model was trained for image processing via ImageNet, it results in the network being better at extracting important features (e.g. edges), and hence results in reasonably accurate performance compared to a model trained from scratch.

5. Summary

With the application of a UNet model with ResNet-18 encoder, we achieved a final private score of 0.88536, placing us at the 1153rd position and the top 52.57% on the leaderboard.



kernel4588900355 (version 37/44) Succeeded 0.88536 0.88003 □

a day ago by NTUCompscience88

ResNet18_BS16 (Post processing; 1st stage with thres 0.919)

We have come a long way from an initial score of ~0.78 to ~0.885 by experimenting with various pre-processing techniques, post-processing techniques as well as models.

Through the course of our experiments, we had attempted to use of several other approaches such as the HRNET with OCR model as well as different variations of UNet but these were either too complex for our current understanding, or yielded poorer results.

Going forward, we would like to propose some possible changes to be made that might yield better results.

Firstly, the pixel count threshold value used in Section 3.5.2 (Reducing False Positives) can be optimized through the practice of exploratory data analysis. An optimal threshold value can be found by analyzing the maximum pixel count that results in the maximum discarding of False Positives.

Secondly, it is plausible that use a deeper ResNet model (i.e. ResNet-152) as the encoder might yield better results. We were unable to try a deep ResNet model as a result of a lack of computational resources at hand.

Thirdly, the discarded pre-processing technique, random grid shuffle (Section 3.3.4), might be more applicable if an optimal grid size was used. Due to time constraint we were unable to investigate an optimal grid sizes, and as such only tried using grid sizes of (50,50) and (16,25). As mentioned in Section 3.3.4, the random grid shuffle resulted in a poorer performance.

Lastly, while the post processing technique performed in Section 3.5.1 (Adjusting Threshold Value) was effective, we suspect that we might be able to find an optimal threshold value for each of the 5 classes (Defect Class 1, 2, 3, 4 and No Defects) as well.

6. Bibliography

[1] S. M. A. A. G. Yadav, "Contrast Limited Adaptive Histogram Equalization Based Enhancement For Real Time Video System," 2014. [Online]. Available: https://www.researchgate.net/publication/281406510_Contrast_Limited_Adaptive_Histogram_Equalization_Based_Enhancement_For_Real_Time_Video_System.

[2] Neurohackweek, "Detrending," 2016. [Online]. Available: https://neurohackweek.github.io/image-processing/02-detrending/.

[3] U. Saxena, "Automold--Road-Augmentation-Library," 2018. [Online]. Available: https://github.com/UjjwalSaxena/Automold--Road-Augmentation-Library.

[4] J. Heaton, "An empirical analysis of feature engineering for predictive modeling," 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7506650.

[5] A. C. P. V. Yoshua Bengio, "Representation Learning: A Review and New Perspectives," 2012. [Online]. Available: https://arxiv.org/abs/1206.5538.

[6] V. E. B. L. P.-P. N. E. M. Popescu Marius, "Multilayer perceptron and neural networks," 2009. [Online]. Available: https://www.researchgate.net/publication/228340819_Multilayer_perceptron_and_neural_networks.

[7] R. N. Keiron Teilo O'Shea, "An Introduction to Convolutional Neural Networks," 2015. [Online]. Available: https://www.researchgate.net/publication/285164623_An_Introduction_to_Convolutional_Neural_Networks.

[8] P. F. T. B. Olaf Ronneberger, "U-Net: Convolutional Networks for Biomedical Image Segmentation," 2015. [Online]. Available: https://arxiv.org/abs/1505.04597.

[9] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *Microsoft Research,* 2015.

[10] C.-F. Wang, "The Vanishing Gradient Problem," 08 January 2019. [Online]. Available: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484. [Accessed 27 April 2020].

[11] A. Veit, M. Wilber and S. Belongie, "Residual Networks Behave Like Ensembles of Relatively Shallow Networks," 2016.

[12] H. Li, Z. Xu, G. Taylor, C. Studer and T. Goldstein, "Visualizing the Loss Landscape of Neural Nets," no. 1712.09913v3, 2018.