



UNIVERSITÀ DEGLI STUDI DI MESSINA

DEPARTMENT OF ENGINEERING

MASTER'S DEGREE COURSE IN  
ENGINEERING AND COMPUTER SCIENCE

---

COMPUTER VISION AND DEEP LEARNING

IN DEFECT DETECTION

*Student:*

**Le Van Huy**

*Student Id:*

**492419**

*Advisor:*

**Prof. Dario Bruneo**

*Co – Advisor:*

**Fabrizio De Vita**

---

ACADEMY YEAR 2020 - 2021

# Table of Contents

<b>Abbreviations .....</b>	<b>I</b>
<b>Acknowledgment.....</b>	<b>II</b>
<b>Abstract.....</b>	<b>III</b>
<b>Chapter 1: Introduction .....</b>	<b>1</b>
<b>Chapter 2: Computer Vision (CV).....</b>	<b>5</b>
<b>Chapter 3: Background Knowledge .....</b>	<b>8</b>
3.1.    Overview of Machine Learning .....	8
3.2.    Overview of Deep Learning.....	10
3.3.    Artificial Neural Networks (ANNs).....	10
3.3.1.    Neuron .....	11
3.3.2.    Activation Functions .....	12
3.3.3.    Layer.....	22
3.3.4.    Forward and Backward (back propagation) steps.....	24
3.4.    Gradient Descent Optimization Algorithms (Optimizers).....	25
3.4.1.    Gradient Descent (GD): .....	26
3.4.2.    Batch Gradient Descent (BGD) .....	28
3.4.3.    Stochastic Gradient Descent.....	28
3.4.4.    Mini-batch Gradient Descent .....	28
3.4.5.    Stochastic Gradient Descent with Momentum.....	29
3.4.6.    Adaptive Learning Rates Methods:.....	31
3.5.    Convolutional Neural Network.....	33
3.5.1.    Convolution Layer .....	35
3.5.2.    Pooling Layer .....	42
3.5.3.    Batch Normalization (BN).....	43
3.5.4.    Fully Connected Layer .....	45
3.6.    Residual Network (ResNet).....	46
3.6.1.    Residual Blocks (RBs).....	48
3.6.2.    ResNet Architecture .....	50
3.7.    EfficientNet.....	52
3.7.1.    Model Scaling .....	53
3.7.2.    Compound Scaling Method – EfficientNet .....	54

3.7.3. EfficientNet Architecture.....	57
3.8. Fully Convolutional Networks (FCNs) .....	60
3.9. U-Net .....	65
3.9.1. U-Net Architecture .....	65
<b>Chapter 4: Methodology Approaches .....</b>	<b>68</b>
4.1. The U-Net Model Approach.....	68
4.2. Combination of the ResNet and U-Net Model Approach.....	70
4.3. Combination of the EfficientNet and U-Net Model Approach.....	71
4.4. Data augmentation and K-Fold Approach .....	72
4.5. Comparison among the approaches .....	73
<b>Chapter 5: Implementation.....</b>	<b>74</b>
5.1. Dataset – Image Processing.....	74
5.1.1. Data Augmentation.....	78
5.1.2. K-fold cross-validation .....	79
5.1.3. Data Generation .....	80
5.2. Model.....	81
5.2.1. Evaluation Metrics .....	82
5.2.2. Loss function .....	83
5.2.3. Execution Environment .....	83
5.2.4. Model Training Process.....	84
<b>Chapter 6: Results .....</b>	<b>86</b>
<b>Chapter 7: Conclusion and Future Works.....</b>	<b>99</b>
<b>Bibliography.....</b>	<b>100</b>

## Abbreviations

AF(s)	Activation Function(s)
AI	Artificial Intelligence
ANN(s)	Artificial Neural Network(s)
API	Application Programming Interface
BGD	Batch Gradient Decent
BN	Batch Normalization
CNN(s)	Convolutional Neural Network(s)
CV	Computer Vision
DL	Deep learning
DNN(s)	Deep Neural Network(s)
FCN	Fully Convolutional Network
FN	False Negative
FP	False Positive
FPD	Function of Probability Density
GD	Gradient Descent
ICS	Internal Covariate Shift
IOTs	Internet of Things
LReLU	Leaky Rectified Linear Unit
ML	Machine Learning
NN	Neural Network
OD	Object Detection
RB(s)	Residual Block(s)
ReLU	Rectified Linear Unit
ResNet	Residual Network
SF	Sigmoid Function

SGD	Stochastic Gradient Decent
SR	Speech Recognition
Tanh	The Hyperbolic Tangent
TN	True Negative
TP	True Positive

## **Acknowledgment**

*I would like to give my deep gratitude to Professor Dario Bruneo and Ph.D. Fabrizio De Vita who are advisors enthusiastically supported and gave me instructions and valuable feedbacks to complete this thesis. And I would like to give thank Kaggle and Severstal corporation for supplying the data of the image and a project requirement that is a base of the thesis. I would like to give a thank Giang Vu who is one of my juniors who joined me in researching and supporting me learn about deep learning and the thesis' project problems in the first days. Besides, I would like to give thank my family, and friends who have always been by my side for encouraging and creating a source of positive energy to help me firmly on the path of completing this thesis. I am very grateful to all of you because this accomplishment would not have been possible without you. Thank you*

## ***Abstract***

*The defects of a product or materials in the manufacturing industries are one of the problems that cause headaches and costs for manufacturers as well as companies around the world so far. It is costly in recalling defective products, costly in refurbishing or remanufacturing them, and also in human resources in monitoring and evaluating products before they are put on the market. Because of that, the application of research in artificial intelligence, as well as computer vision for defect detection in the manufacturing industries, was started a few years ago, to reduce defective products as well as minimize costs such as recalling products, remaking them, or consuming human resources for product quality assurance. More than that, it is aimed at helping the product have a better and perfect level when it reaches the user. Therefore, in this thesis, it has been researched and applied the research results of the forerunners in the field of Deep learning as well as computer vision. Specifically, it has been researched on the topic of defect detection on steel surfaces, it was taken from Kaggle's contest titled "Severstal: Steel Defect Detection", which was a quite big contest held in 2019. In this thesis, there has been a solution for the requirements of detecting and classifying defects on the steel surface through the image data set provided by the Russian corporation Severstal. More specifically, there has been an implementation of semantic segmentation and accompanied by some other techniques in deep learning to solve this problem.*

## **Chapter 1: Introduction**

In the era of industry 4.0, with technological achievements that the world has achieved through research on artificial intelligence (AI) from the last century, accompanied by well-developed infrastructure, machinery, and equipment applying cutting-edge technology, which help IoT systems are gradually growing stronger and more popular, they have such high applicability in most aspects of life. Currently, there is a rapid development of smart health care systems, smart agriculture, smart cities, smart homes, smart industries, and so on... all these "smart things" are built on top of the distributed systems and physical infrastructures of each field. If the distributed system, which is considered as the "backbone", is the main architecture building on the internet of things (IoTs) system, helps the IoTs system operates smoothly, then artificial intelligence acts as a "cognitive area of the brain" in the IoTs system. It helps the system to collect and process information to make decisions, more specifically here is computer vision of deep learning, which is one of the indispensable components in the IoTs system. Using cameras along with sensors such as temperature, light, sound, atmosphere, or GPS in most popular IoTs, they are one of the important devices used to capture collect data about the system's cloud and use trained deep learning models to make decisions on these data or use these data to train and improve the "brain" of the system IoTs are getting better and better for specific purposes of the system.

With the birth and development since the 1950s of the last century, computer vision, which, is one of the sub-disciplines of AI, has made progress as well as achieved outstanding achievements in the present. Its applications play a very important role in these systems. Looking deeply,

Smart homes often use them in facial recognition, identification tasks such as people, objects, pets, or even identifying and alerting thieves to protect you and your family. Smart cities with applications for developing autonomous vehicles, or traffic monitoring camera systems, smart parking lots that let you know how many spaces are available, or smart airports that help for identification identity. Smart heaths with supports for doctors in diagnosing diseases by predicting phenomena in images from x-ray, endoscopy, ultrasound, or tomography. Smart agriculture with monitoring and diagnosing the health of crops and livestock. Smart industries with camera systems for production or monitoring of the production process of products to ensure quality, to reduce and to avoid costly human resources in tasks that can replace by using machines. Specifically, defect detection on steel surfaces based on computer vision is one of the very useful applications in the industry of manufacturing steel materials or products made from steel, it helps enterprises can identify product defects and evaluate product quality, or help reduce costs and improve product quality.

In the next parts of the thesis, we will briefly cover computer vision and its common applications in technology as well as in life and production in chapter 2. Then we will go to write about the knowledge in deep learning is also such as artificial intelligence used for this project in chapter 3, in detail, we will clarify the basics of Artificial Neural Network (ANN) and important components of the network such as layers and types of layer, activation functions, Convolutional Neural Network (CNN) and some state-of-the-art architectures of CNN like ResNet being built from Residual Blocks, EfficientNet with applying 3D compound scaling method, U-Net with U-shaped structure including encoding part

and decoding part for image segmentation work; or about optimization methods that are commonly used in model training work.

The fourth chapter is methodology approaches to the thesis, where we will present step-by-step the road to finding a viable solution to the defect detection problem on steel surface through the requirements of Kaggle's contest with the accompanying dataset of Severstal corporation. The initial approach is based on understanding the CNNs and their achievements by the utilization of the U-Net model for image segmentation to get a relatively optimistic but not the best result as the first step. we continue with other approaches like learning about the combination of some state-of-the-art models with the U-Net model and techniques of data augmentation as well as K-fold strategy in the training model to achieve impressive optimal results.

Continuously, relying on the achievements of chapter 4, we will give out a solution by applying the state-of-the-art CNN models with a combination of them to give positive results. It is the method of semantic segmentation on the image dataset by a combination of U-Net as a master architecture and versions of ResNet as the backbone architecture for U-Net, or another combination of U-Net and versions of Efficient, is respectively the overall architecture and the corresponding backbone in the model. Furthermore, given the data set was provided with an imbalance in defect type, so we went to make them balance in the data type by augmentation method and using the K-fold method to evenly divide the types of data and the quantity of data for the training data set and validating data set to help the learning model give an objective result and improve the prediction accuracy at the testing data set. In addition, applying an optimization method as Adam with default values for parameters to help the model learn faster and more accurately.

Using 5 folds and going through over 20 - 30 epochs for each fold in the training model, we got good results with accuracy ranging from 90% - 94% with validating data set and around 68% - 71% with testing data set through model evaluation. The following step was to make an evaluation of the model's performance with techniques of Accuracy Classification Score and Multi-label Confusion Matrix. The Accuracy Classification Score method gave results ranging from 72% - 82% in total prediction of defect types and results from 90% - 95% in predicting each defect type. Then, the Confusion Matrix method also gave quite optimistic results of classifying labels between ground-truth masks and predicted masks of test data set. In the final step, we went into re-checking the results with the naked eye by rendering the original image with ground-truth masks and the original image supplying predictive masks based on the threshold number of pixels that the model makes predictions for each defect type. To understand the solution in more detail, we will cover them in chapter 5 - Implementation and chapter 6 - Results.

The final chapter is the conclusion and future works, which is with an overview of the thesis as well as the ideas to reach a new better achievement in the future.

## **Chapter 2: Computer Vision (CV)**

Computer Vision is an Artificial Intelligence (AI) area which deals with computational approaches for assisting computers in understanding and interpreting the content of digital images [38]. As a result, computer vision seeks to enable computers to view and comprehend visual data received through cameras or sensors to identify and understand the visual world automatically by emulating the vision of human beings. In particular, the goal of computer vision is to artificially mimic human vision by allowing computers to comprehend visual stimuli meaningfully [39]. As a result, it is also known as machine perception. CV remains one of the most difficult subjects in computer science owing to the vast complexity of the varying physical world. In fact, human sight is the result of a lifetime of learning with context to train how to detect certain things or human faces or individuals in visual settings.

CV is not a newly discovered technology nowadays because the earliest attempts with it began in the 1950s, and it was used to analyze typewritten and handwritten text after that [39]. At that period of time, computer vision analysis processes were very straightforward, but they needed a significant amount of work from human operators who had to manually input data samples for analysis. So, it was difficult to supply a large quantity of data manually. Furthermore, the computer capacity was insufficient making the error margin for this analysis was quite large. There are numerous supports for CV nowadays. Cloud computing, when combined with powerful algorithms, has the potential to help us tackle even the most difficult issues. The combination of new hardware, clever algorithms, and the massive amount of publicly available visual data that we generate daily is responsible for propelling computer vision

technology ahead. Modern artificial vision technology uses machine learning and deep learning approaches to train robots to recognize objects, faces, or people in visual settings based on a large pool of picture and video data. As a result, computer vision systems employ image processing methods to enable computers to locate, identify, and evaluate objects and their surroundings using data from a camera.

Moreover, CV systems are trained to examine products, monitor infrastructure, or a manufacturing asset in order to detect faults or difficulties in thousands of products or processes in real-time. It can swiftly outperform human capabilities because of its speed, continuity, objectivity, correctness, and scalability. Applications of CV are employed in numerous industries, including security and medical imaging, as well as manufacturing, automotive, agricultural, construction, transportation, smart cities, and many more. More use cases become conceivable as technology progresses and becomes more versatile and scalable. In addition, the most recent deep learning models outperform humans in real-world image recognition duties such as facial identification, image classification, object detection, and image segmentation.

**Image classification:** Accepting input is an image and outputs the labels for duty of classification along with several measures (loss, accuracy, probability, etc.). For example, a picture containing a duck may be classed as a label of “duck” with some probability, whereas an image contains a chicken being made a defined as a label of “chicken” with several probabilities [43].

**Object Localization:** Helping detect the presence of an object which is bounded in a box within an image, receiving an input of image

and returns the box located in the formation of width, height, and location [40].

**Object Detection:** Doing both classification and localization which receive an input of image and generates one or several bounding boxes, each box contains the attached label of class. These methods are powerful enough to handle localization and classification for multi-class and objects containing numerous occurrences as well [43]. However, the bounding boxes in object detection are always a form of a rectangle. As a result, if the item contains the curvature part, it does not aid in establishing its shape, and object detection does not have the ability to estimate some measurements in high accuracy such as an object's area or perimeter from an image.

**Image Segmentation:** Being an extra extension in the detection of objects. It helps us designate the existence of an object in the image using pixel-wise masks built for every single object. The method is more detailed than producing of bounding box since it allows us to determine the geometry of every single object appearing on the image. This granularity is useful in a variety of applications, which includes medical image processing, satellite imaging, and so forth [43]. Many pictures segmentation approaches have lately been proposed. Mask R-CNN, U-Net are the popular models in segmentation. The segmentation has two essential forms. Firstly, instance segmentation can make an identification for the borders of an item and label its pixels by distinct colors [41]. Secondly, semantic segmentation is the processing of marking each pixel as a label on an image including the background, the pixel is with a distinct color based on its label of class, or its category [41].

## **Chapter 3: Background Knowledge**

### **3.1. Overview of Machine Learning**

Machine learning (ML) is AI technology allowing computers to make automatic learning and improvement based on experience without flagrant programming. ML is focused on the development of computer programs being able to make access and utilization of data to learn themselves. Its progress of learning starts from data or observations, for example, a direction of finding out data patterns to make decisions with a better quality relied on instances being supplied in the future [37]. The fundamental purpose of helping computers have an ability to learn autonomously, which does not contain any participation or assistance of people and then adjust activities of them suitably. Then, ML allows for the examination of enormous amounts of data, although it generally produces quicker and more exacting results in identifying valuable possibilities or risky hazards, it probably necessitates more time and resources to train it precisely. In addition, the combination of ML with AI and technologies of cognition can increase its efficiency in processing massive amounts of data. Algorithms of ML are usually classified into supervised or unsupervised [37].

- **Supervised** ML algorithms have the potentiality for predicting events of the future by applying what they learned previously on new data by utilizing labeled examples. The learning method generates a function of inference to predict output results based on a study of a known set of training data. After going through an adequate process of learning, the system may supply goals for any input of new data. Moreover, the

algorithm also has the function of making a comparison between its output and output of expectation or correct output and gives out an error detection to make a suitable adjustment for the model [37].

- On the other hand, techniques of **unsupervised** are utilized when the material required to train is both unclassed and unlabeled. These algorithms learn about the mechanism of computers for producing function of inference from data without being labeled, which is to express an unseeable construction behind. Although the system does not determine the correct output, it investigates the data and is able to make conclusions from sets of data to characterize concealed constitutions of unlabeled data [37].
- **Semi-supervised** methods sit midway between supervised and unsupervised learning because they train with both unlabeled and labeled data, which is often a little of labeled data amount and a huge amount of unlabeled one. This strategy allows systems to significantly enhance learning accuracy. Semi-supervised is particularly utilized when the collected labeled data requires experienced and agreeable resources for training from. Besides, obtaining unlabeled data usually does not necessitate the use of supplemental resources [37].
- **Reinforcement** ML algorithms are a type of learning system which is several interactions with its surrounding environment by performing executions and exploring rewards or errors. The most essential aspects of these algorithms are making several searches of trials and errors and postponed rewards, which enable devices and agents of software to find an optimal

behavior at a specific scenario to achieve an improvement for their performance to the maximum point. For the agent to know what action is the best, there is a requirement of a reward's simple feedback, which is called the signal of reinforcement [37].

### 3.2. Overview of Deep Learning

Deep learning (DL) is a small field belonging to ML and AI which utilizes a bunch of layers of ANNs to reach achievements of "state-of-the-art" accuracy in tasks like speech recognition (SR), language translation, object detection (OD), and others. There is a difference between techniques of DL and typical techniques of ML, DL is able to learn automatically from data such as text, image, or video without the utilization of hand-coded rules or knowledge domain of human beings. When given more data, their architectures with high adaptability are able to learn directly from raw data to achieve an improvement of prediction accuracy [42].

DL is widely employed in applications such as CV, systems of recommendation, and AI in conversation. DL is applied for CV software to learn from digital productions such as photos and videos. The applications of Conversational AI assist devices in understanding and communicating by natural language. Besides, language, user's behaviors, and digital productions are used by recommendation systems to provide searching results and services which are suitable and significant [42].

### 3.3. Artificial Neural Networks (ANNs)

The concept of ANNs was built on the inspire of human brain biology characteristics, Cybenko [1] gave out a substantiation which

there is a hidden layer including neurons with several specific limitations, having the ability to make an approximation for any unstoping function to achieve any desire of precision in a neural network. It included a group of ML models having the potentiality to “learn” to execute or perform some specific tasks based on supplied examples. In general, these networks were created by a group of units or nodes connected to each other like a simulating version of neurons in the human brain, which are called artificial neurons.

### **3.3.1. Neuron**

Analog electrical signals flow inside biological neurons, while digital values are present as the signals for artificial neurons. Each artificial neuron has an output being computed by using a function of the sum of the input signal values with generally added bias, the connection between the neurons as known as edges, in each connection contains an adjustable model parameter named weight that can be “learned” through a process called “model training”. The strength of the synaptic will determines the interaction’s level between the neurons, this function combines the neurons, bias, and weights in the connections called “activation function”, and there are several activation functions that mean there is not only one fixed activation function for ANN, and we can choose the activation function for the network’s output. These functions will be mentioned later in this thesis.

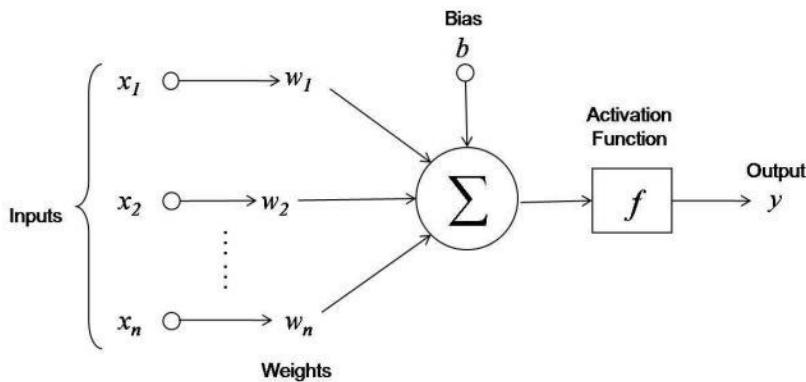


Figure 1: Artificial Neuron Structure [46].

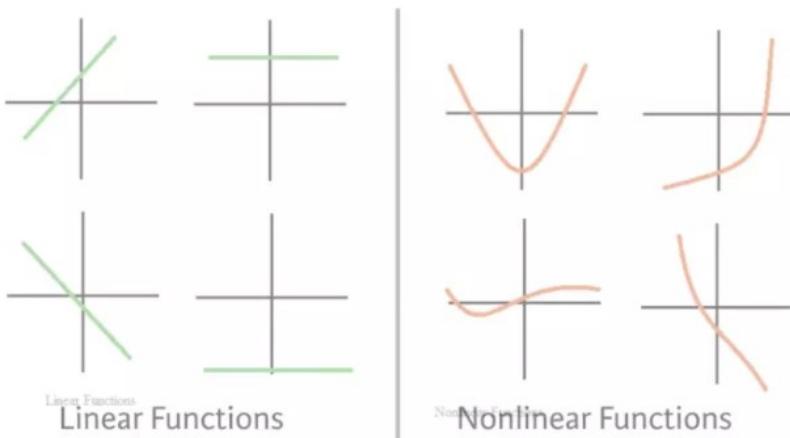
### 3.3.2. Activation Functions

Activation Functions (AFs) in Neural Network (NN) are a component which is very essential in Deep Learning. AFs help gives out a determination for a Deep Learning model's output, the accuracy, and effectiveness of computation of the training model making a decision on the success or failure of a NN system. AFs also have a considerable influence on the convergence and convergence speed of NNs, or in some cases, AFs can prevent NNs from assembling right from the start.

AFs are presented in equations of math and this function is attached to every single neuron in the network and produces a determination of being activated or not, relying on the input data fits prediction of the model or not, which also help manufacture a normalization for every single neuron's output with a range of zero to one or minus one to one, depending on the type of AF being used in the layer or network. Additionally, another aspect of AFs is they have to have efficiency in computation because of being computed over millions or even billions of neurons for every single instance of data. And modern NNs utilize a technique of backpropagation for training the model.

The main reason that NN models stand out from machine learning models is their ability to solve non-linear data problems through AFs of hidden layers which execute the task of solving problems for complex nonlinear relationships between data features and model outputs.

As mentioned earlier, in a NN, the input data is fed to the neurons which are located in the input layer. Every individual neuron has the ability to produce an outcome as an output from the multiplication of its input and its weight, this output is then passed on to the following layer. So, AFs are applied in NNs to be on duty of computation for the weight input's sum and biases, being utilized to make a decision on a neuron be activated or not. It executes a manipulation of the supplied data via several handlings of gradient, or gradient descent (GD) in typical and thereupon gives out production for a NN's output, including the parameters. These AFs are usually known as a reference for a transfer function in some documentation or research papers. Furthermore, relying on the function they deputize, AFs have the potentiality of being either linear or non-linear, and they are used to modulate the outputs of NNs in a diversity of areas ranging as segmentation, object recognition, classification, speech recognition (SR), cancer detection systems, fingerprint detection, and so on... [12]



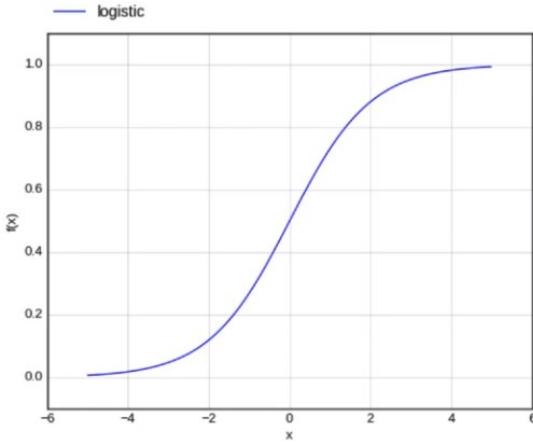
*Figure 2: Linear and Non-linear functions [47].*

Although in fact that there are many types of AFs, in this article, we will only mention some of the AFs that are commonly used in current CNNs architectures like sigmoid, softmax, hyperbolic tangent (Tanh), rectified linear unit (ReLU) and Leaky rectified linear unit (LReLU).

### 3.3.2.1. Sigmoid function (SF)

Sigmoid Function is also called the Sigmoid curve. This is built relied on a mathematical function characterized by a curve with S-shaped, which is one of the non-linear activation functions most widely used. If being familiar with some machine learning models, probably still remembering Logistic Regression - as one of the simple algorithms for binary classification problems, which is quite effective. The "soul" of Regression is this Sigmoid function. The sigmoid is a continuous nonlinear function, allows to pass real numbers as its input and gives a production of results in the range 0 to 1, which is considered probabilistic in some problems. Suppose you trained a NN to image classification of birds and dogs, what a classic problem, where the bird is 1 and the dog is 0. Basically, when your Model returns a value greater

than 0.5 meaning the image is of a bird, or the value less than 0.5 means the image is of a dog.



*Figure 3: Sigmoid function.*

In the Sigmoid function, a small adjustment from the input results in no relative change for output. It gives a smoother and more continuous output than input, so, it has derivative in everywhere [13]. Figure 4 shows the formula of sigmoid function:

$$f(x) = \left( \frac{1}{(1 + \exp^{-x})} \right)$$

*Figure 4: Sigmoid function equation.*

In the formula,  $x$  represents the network's output, and  $f(x)$  is its function, and  $\exp$  stands for the Number of Euler. As a result, if  $x$  goes ahead to the infinity of positive, the value of expectation  $f(x)$  becomes 1. In contrast, if  $x$  approaches the infinity of negative, the predicted value of  $f(x)$  becomes 0. And if the result of the sigmoid function is greater than 0.5, that label is classified as class 1 or positive side, and if it is less than 0.5, classified as class 0 or negative side.

On the other hand, the sigmoid has significant shortcomings such as gradients of sharp damp while doing backpropagation starting from

deeper hidden layers back to layers of input, a saturation of gradient, sluggish convergence, then the output of having non-zero centered, which is the cause of having various passageways for updating of propagation. another AF type, such as Tanh, has been recommended to address some of the shortcomings of the Sigmoid AF.

### 3.3.2.2. SoftMax function

SoftMax Function is an exponential averaging function that calculates the probability of an event occurring. In general, this algorithm calculates the probability of a class occurring out of the total of all possible classes. This probability will then be used to determine the target class for the inputs. Specifically, the SoftMax turns a vector with k - dimensional of any real numbers to become a real-valued k-dimensional vector summing to 1. Its input value might be zero, minus number, or positive number, however, the function will always turn them into a value in the range (0:1]. The SoftMax Function is calculated following this formula:

$$f(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

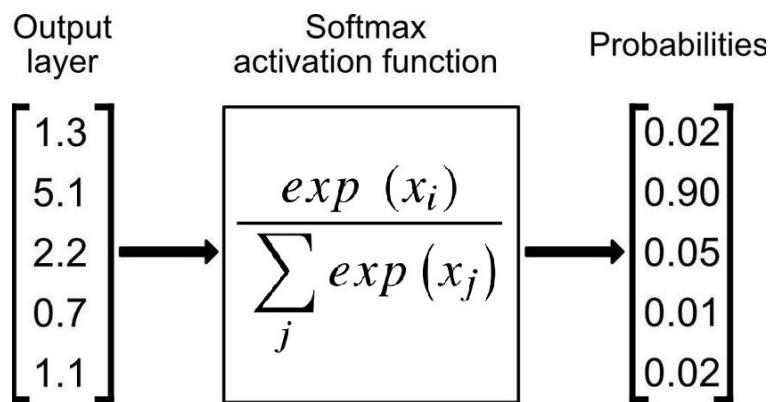
*Figure 5: SoftMax function equation.*

Where ( $x_i$  is the vector of input values for the function from  $x_0$  to  $x_n$ , in which all the values can be any real number, positive number, negative number, or zero. Next,  $\exp(x_i)$  is the standard exponentiation function is applied to each input value returning a positive value greater than 0. This value will be very small if the input is negative, and very

large if the input is positive and it is an uncertain number in the range (0:1] as the requirement of a probability.

As such, they may be called “probabilities”. If there is a negative number or very small number of the input value, the function transforms them into a tiny probability. In contrast, if there is a large input value, it will be converted to a large probability. But the probability is always greater than 0 and less than 1 or equal to 1. So, the sum of all probabilities is always 1 and the function brings a lot of benefits such as:

- Optimized when calculating the maximum probability in the model parameter.
- The property of the SoftMax makes it satisfactory for the interpretation of probability, which is very beneficial in ML.
- Normalization of SoftMax is a method to minimize the effect of extreme values or outliers in data without having to modify the original data.



*Figure 6: Softmax function applied at the output layer of a network for the classification task.*

For the reasons mentioned above, the function of SoftMax returns the value of probabilities for each class in multi-class models, with the

class of intention having the supreme likelihood. The function may be found in practically all of the deep learning's output layers architectures where it is used. Adding, there is a distinctness between SoftMax and Sigmoid, which the Sigmoid is utilized for classification of binary and the utilization of SoftMax for the tasks of diversity classification.

### 3.3.2.3. Rectified Linear Unit (ReLU) Function

In 2010, Nair and Hinton introduced the AF of rectified linear unit (ReLU), which has been the supreme extensively utilized AF for applications in the DL field, bringing a state-of-the-art result till currently [17]. In general, The ReLU AF has an expressive speed of learning AF proven to be the most extensively and successfully applied function [14]. In DL, it outperforms and generalizes the Sigmoid and Tanh activation functions [18]. In particular, there is a representation of a roughly linear function in the ReLU function and hence preserves the features of models in linear that make them easier to ameliorate using the approaches of gradient descent. The mechanism of the ReLU AF, which applies a functioning of threshold to each element of input having values smaller than zero to be replaced to zero, and the formula of the ReLU AF as follows:

$$f(x) = \max(0, x) = \begin{cases} x_i, & \text{if } x_i \geq 0 \\ 0, & \text{if } x_i < 0 \end{cases}$$

*Figure 7: Rectified Linear Unit function equation.*

The ReLU will correct the inputs' values that are smaller than zero, they are driven to 0 and are removed the issue about the gradient of vanishing. Mainly, the ReLU is put to utilize into the hidden units of

deep neural networks (DNNs). Besides, another AF as sigmoid or SoftMax is often applied in the network's output layers with the popular purpose of classification in the image [12] and SR.

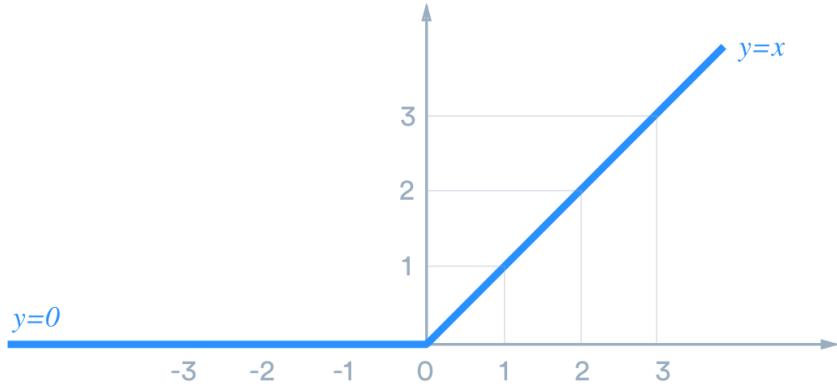


Figure 8: Rectified Linear Unit Function [33].

There are many advantages when using ReLU. In detail, the fundamental advantage of ReLU in computation is that they bring a more expeditious computation because of not needing to quantify divisions and exponentials of them, directing to an increment of achievement of overall computation [18]. creation of sparsity in the hidden units is another feature of the ReLU, which calculates the square value from 0 to the value of maximum. The ReLU exists an issue which it readily becomes overfitted when made a comparison with the function of sigmoid, despite utilizing the approach of dropout to discounting the overfitting's influence of ReLUs and the rectified networks enhanced deep neural network (DNN) performances [19].

There is a notable weakness in the ReLU, which it is occasionally brittle during training, leading to the death of several gradients. This will make neurons to be dead and direct weight updates to inactivate data points in the future, which hampered learning because the dead neurons result in zero activation [3]. So that, the leaky ReLU was offered as a solution to the dead neuron problem.

### 3.3.2.4. Leaky ReLU

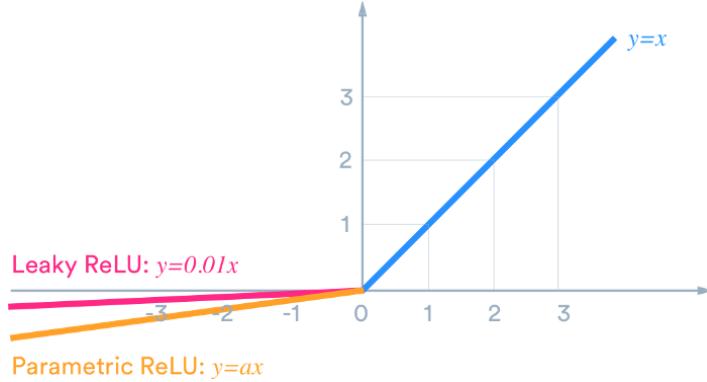


Figure 9: Leaky Rectified Linear Unit Function [33].

The leaky ReLU was released in 2013 that gives out several mini slopes of negative to the ReLU for maintaining and staying surviving the updates for weight throughout the phase of propagation. The building of the alpha parameter serves as the solution to the issue of dead neurons of the ReLUs, which ensure that the gradients are not fixed zero anymore during training. The computation of the LReLU has relied on an unchanged tiny value of gradient for the negative gradient  $\alpha$  in the range of 0.01; hence, the computation of LReLU AF is presented following this formula:

$$f(x) = \alpha x + x = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

Figure 10: Leaky Rectified Linear Unit function equation.

The comparison of the ReLU to an exception of not having zero gradients over the entire time, which reports a similar result. Therefore, except for distribution and sparsity in the comparison of ReLU to Tanh, no having noteworthy enhancement of result. As a result, Leaky ReLU offers two advantages:

- Because it lacks zero-slope portions, it solves the "dying ReLU" issue, so the training process is expedited. And, the "mean activation" near zero is evidence for speeding up training. (It aids in keeping the diagonal entries of the Fisher information matrix short, but it can disregard judiciously.) Unlike ReLU, leaky ReLU is more "balanced," and as a result, it may learn faster.
- Keep in mind that the outcome is not always consistent. Leaky ReLU isn't necessarily better than regular ReLU, and it should only be used as a last resort.

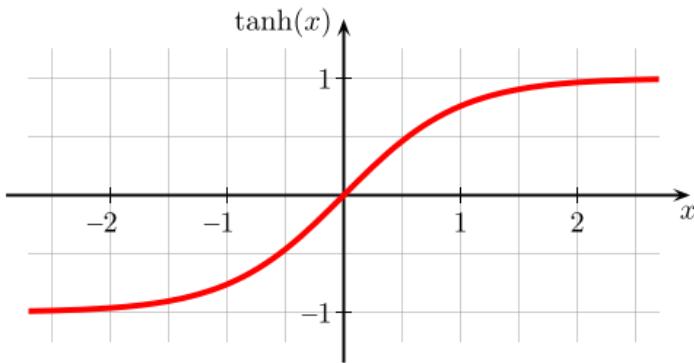
### 3.3.2.5. Hyperbolic tangent function (Tanh)

Another form of AF utilized in DL is the hyperbolic tangent function, which has several versions employed in applications of DL [14]. Tanh is built as a zero-centered function that is silkier with a range of minus one to one, and its output is given by:

$$f(x) = \left( \frac{e^x - e^{-x}}{e^x + e^{-x}} \right)$$

*Figure 11: Tanh equation.*

Because of supplying higher training performance for multilayer NNs than the SF, tanh has become the favored function over the SF [15]. Otherwise, the function of tanh, like the SFs, was unable to overcome the gradient of vanishing problem. The advantage key of this function is producing a generation of output from zero-center support in the operation of backward progress.



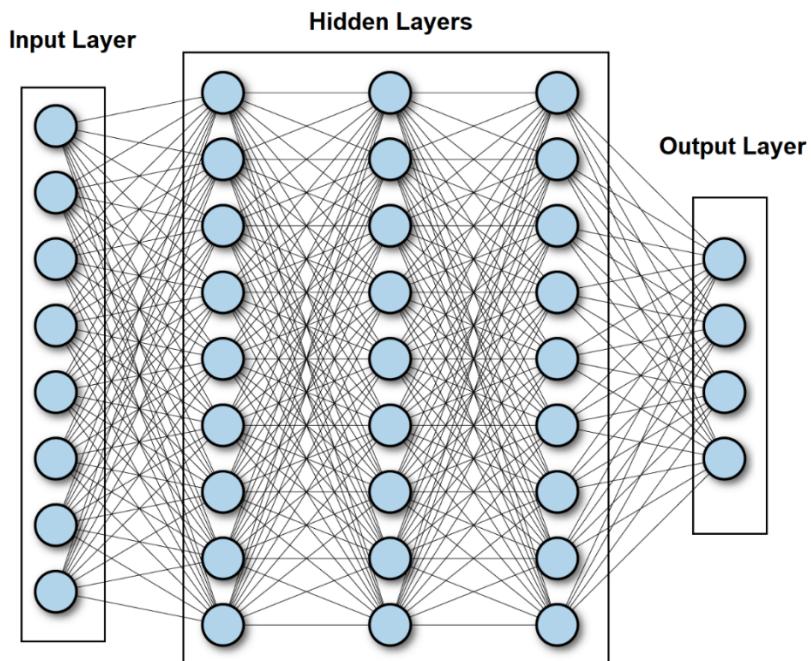
*Figure 12: Function of Tanh with values moving in the range [-1: 1].*

The tanh has a specialty which is capable of an accomplishment of gradient value equal to 1 in case there is only zero value of the input, that equivalent to  $x$  equal zero. As a result, the tanh generates several inactivated neurons within the progress of computing. The unworked neuron is a situation in which its weight of activation occasionally is employed similar to a consequence of gradient of zero. In addition, this obstruction of the tanh prompted an additional study in AFs to remedy the issue, giving rise to the ReLU AF. Furthermore, the tanh function has mostly been employed for SR and natural language processing in recurrent neural networks [16].

### 3.3.3. Layer

All neurons are linked directly in the network, on the other hand, they are grouped into subsets of neurons known as layers, which are generally connected in a sequential way based on the activation function in every node to compute nodes' outputs from node's inputs. In detail, the neurons in a layer are made a connection to a single set of neurons in both the previous layer and to another one in the next layer. Because of the modeling of ANNs as a connection of a neurons' structure in the layer, neurons' outputs at a certain layer become neurons' inputs at the

following layer, which networks are understood as feedforward NNs. There are no loops in the current layer, therefore the inputs are always sent forward. As a result, the neurons in a layer cannot be connected to each other. When all of the neurons in the layer are created a connection to all other ones belonging to the prior layer and the following layer, being called “Fully Connected Layers”, this state of the connection is kept in the whole network as a series of fully connected layers that will be called “Fully Connected Neural Network”.



*Figure 13: Fully Connected Neural Network [45].*

As Figure 13 shows above, there is the existence of 3 kinds of layers inside a network, they are layers of input, hidden, and output. Specifically, the Input layer represents the input of the network, and the number of nodes as known as neurons is the size of the input that we call “input dimension”. The hidden layers lay in the middle of 2 layers of input and output, there is the ability to exist so many hidden layers in a network, however, there is a single input layer and a single output layer

in general, the output layer represents the network's output. In addition, the bias is contained in the input layer, not in the output layer. Although simple decisions are made at the first layer relying on the input, sophisticated judgments are made at the second layer depending on determinations established at the first layer. As one moves into the network more deeply, more difficulty and judgments of abstraction emerge. The phrase of deep neural networks refers to a network with numerous hidden layers.

### **3.3.4. Forward and Backward (back propagation) steps**

As mentioned before, there is a journey of the signals with an order starting from the input layer, hidden layers and till output layer; which operations are performed over the values received from the nodes in previous layers (in all layers but the input layer) and whole activation function results are sent forwards to the next layer till the layer of output. However, it is not the end of the process. After getting the result of the output layer, there is a backward process named back-propagation applied here as known as the training step for ANN. The algorithm is utilized to productively train a NN via a technique being known as chain rule. Simply, the action of backpropagation is a backward journey after every single journey of forwarding movement through a network, while there is an adjustment of the parameters of models (biases and weights), which helps minimize the error for network's result, it is called "loss function" or "cost function" which is the evaluation between predicted and expected output at the final point of forwarding step. Basically, this function calculates the gradient of the loss function to every single parameter in the network respectively means it works from the top of the model (output layer) to the bottom of the model (input layer) for

calculating the contribution of each parameter in that value [2], which updates the weights until reaching the minimum value. Furthermore, the key point of calculating the minimum value for loss function is gradient descent, which is a process of determining the modifications needed to give out the minimum of the loss and optimization of the model becoming a specific evaluating measurement by calculating the derivative of the loss function [3].

### 3.4. Gradient Descent Optimization Algorithms (Optimizers)

During the training phase, we frequently modify and change our model's parameters (weights) in order to try to minimize its loss function and help produce predictions as optimal and accurate as possible. As a result, optimizers were developed to assist in resolving this issue. They make a combination between parameters of the model and function of loss by making updates of the model for replying to the loss function's output. In other terms, optimizers are techniques or approaches which are utilized to help error function (loss function) reach a minimum value or to make production efficiency peaking a maximum value [28]. The optimizers are built from functions for math being influenced by learnable parameters of the model, they are biases and weights. Additionally, the assistance of optimizers in determining the adjustment of learning rate and the weights of a NN for decreasing losses becoming lowest. In a nutshell, the optimizers will mold your model into its most accurate possible form by tinkering with the weights [28]. And the loss function serves as a guide to the terrain, informing the optimizer whether it is traveling in the correct or incorrect path. There are several common optimizers such as “Batch Gradient Decent” (BGD), “Stochastic Gradient Decent” (SGD), “Mini-Batch Gradient Decent”, SGD with

Momentum, Adaptive Learning Rate methods (AdaGrad, RMSprop, Adam).

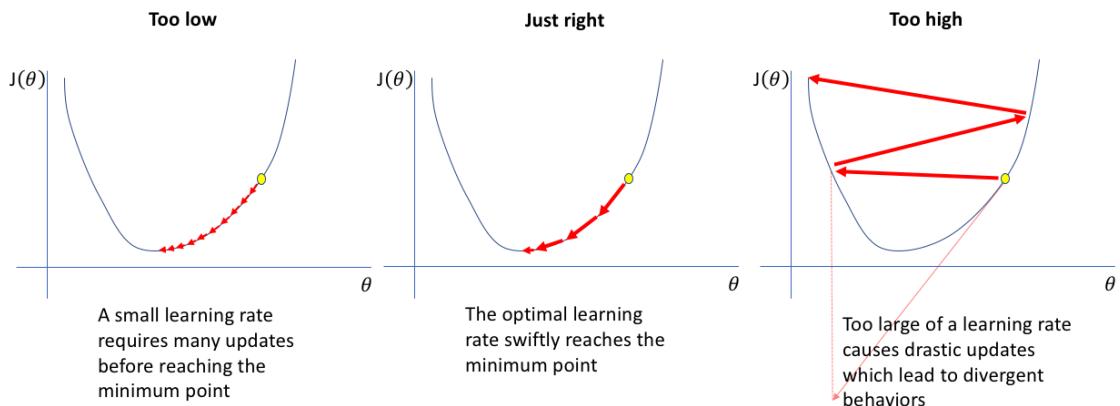
### **3.4.1. Gradient Descent (GD):**

Gradient descent is known as the main algorithm for purpose of optimizing the model, which is commonly utilized for training ML models and NNs [4]. GD is built relied on a function of convex to iteratively change parameters for minimizing a provided function to its local reaching to the lowest value. In the other words, GD gets started by defining the initialized values of the parameter and then customizes iteratively these values for the given cost function to become the lowest one. Deeply, a gradient is a function's derivative with several input variables. In terms of mathematics, a gradient is known as the function's slope, and it is an intelligible measurement of all weights' modification being related to modification of error. Specifically, the steeper the slope and the faster a model can learn, the higher the gradient. However, if the slope is zero, the learning model will be stopped. A gradient is understood as a "partial derivative" regarding inputs of it.

Imagine that a blindfolded man wants to hike down to the bottom of a valley from the top of a hill with as few steps as possible. To go down the hill, he might begin making huge steps in the sharpest direction, which he can do as long as he is not close to the bottom. Nevertheless, the closer he gets to the bottom, the narrower his steps will become to prevent overshooting it. So, the gradient can be utilized to mathematically characterize this process. The learning rate  $\eta$  is a determination for the size of steps that we are able to get an achievement of the lowest local value. Similarly, going ahead the slope direction of

the surface that is produced relied on the downhill of objective function, keeping on as far as reaching to a valley [5].

To achieve the purpose of minimizing the local value for GD, there is a relevant selection for the learning rate value, which is not both excessively low and excessively high. It is very essential because excessively large steps could fail to fall down the lowest local value since it jounces in range of the convex function of GD. Besides, GD will finally fall down the lowest local value if the learning rate is set to an excessively low value, but it may take a lot of time [6]. So, the key point is choosing an optimal learning rate that helps the model converge to the minimum value without spending much time.



*Figure 14: Learning rate choices that decide gradient descent result [44].*

GD has three forms that distinction the number of data being utilized to find the gradient of the objective function. There is an exchange about the exactness of the parameter updates and the time it spends completing an update relied on an amount of data.

### 3.4.2. Batch Gradient Descent (BGD)

Batch gradient descent calculates a sum of error of every single point in the train data set and updates the model after all instances of training being handled an evaluation. This procedure is known as an epoch of training. Though batch supplies an efficient calculation, it still has the ability to take such a much time to execute big sets of training data because it must keep all data in memory [5]. BGD also frequently yields stability of convergence and error gradient, although the convergence point is not always the best, locating the lowest local value rather than the lowest global one.

### 3.4.3. Stochastic Gradient Descent

SGD performs an epoch of training for every single instance in the set of data and makes an update for the parameters of every single instance of training in a period of time. They are easier to remember because of holding an instance of training only [5]. In spite of the fact that the recurring updates provide more specific and rapid, they are able to lead to computational efficiency losses when it is compared to batch gradient descent. Despite having the ability of noisy gradients production, these frequent updates can also aid in escaping the lowest local value and locating the global value.

### 3.4.4. Mini-batch Gradient Descent

A combination of BGD with SGD ideas gives out Mini-batch gradient descent, which divides the data set for training into tiny batches and handles to update every single batch of those. The method reaches a balanced achievement between batch gradient descent's computing efficiency and stochastic gradient descent's speed [5].

Besides, gradient descent has its own set of challenges such as points of the saddle and local minima, gradients of exploding and vanishing.

- Local minima resemble global minima in shape with the slope of the cost function increasing at each side of the present location.
- Points of saddle happen when the minus gradient appears just at a side of the point, the side will fall down to the lowest local value and the other side will peak to the highest local one. Its shape looks like a horse's saddle.
- Vanishing gradients happens when the gradient is too small. The gradient continues to shrink when traveling backward in the progress of backpropagation, leading the learning of the earlier levels of the network will be slower than layers in the following. When it occurs, the parameters of weight are updated till the point when they become inconsequential, which means 0, the consequence of an algorithm does not have the ability to learn anymore.
- Exploding gradients occur in case of achievement of the excessive big gradient, resulting in instability of model. The weights of the model shall get excessively huge and be given out as “NaN” finally. an approach to this problem is the utilization of a technique of dimension reduction, which has the ability to reduce convolutedness of the model.

#### **3.4.5. Stochastic Gradient Descent with Momentum**

SGD gets difficulty in ravines-navigation which are widespread around local optima and are regions where the curving surface is more

considerable in one dimension than the other one. Consequently, there is a swaying of SGD on the slopes of the gorge, which makes only sluggish progress from the bottom toward the lowest local value [28]. That is the reason why momentum was established to reduce extreme volatility in SGD and soften the model's convergence. Moreover, it promotes assembling in the proper leading while decreasing variation in the extraneous direction. This approach employs another hyperparameter known as momentum, denoted by the symbol " $\gamma$ ".

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta) \\ \theta &= \theta - v_t \end{aligned}$$

*Figure 15: Momentum Formula [29]*

Some implementations swap the equations' signs. Typically, the term of momentum  $\gamma$  is initialized to 0.9 or an equivalent number. We basically use momentum to drive a ball down a hill. As it rolls downhill, the ball gains momentum, increasing faster and faster (until it achieves its terminal velocity at the point of being resistant by the air, i.e.  $\gamma < 1$ ) [29]. The term of momentum increment for dimensions has the point of gradients in the equivalent directions and has a decreasing of dimensions having unstable directions in gradients. Consequently, there is an achievement of quicker convergence and less oscillation. Although momentum helps eliminate swaying and high variance in the parameters and congregates quicker than GD, there is a hyper-parameter which must be added and adjusted manually and precisely [28].

### 3.4.6. Adaptive Learning Rates Methods:

The difficulty in employing schedules of learning rate is relied on their hyperparameters being established before and is strongly dependent on the model type and issue. Another issue is that all parameter updates use the same value of learning rate [29]. If the data is sparse, we desire to update the parameters to a different extent. With this cause, AdaGrad, Adadelta, RMSprop, and Adam are adaptive learning rate algorithms that offer an alternative solution for traditional SGD. These per-parameter learning rate approaches give a heuristic approach that does not require costly manual tuning of hyperparameters for the agenda of learning rate.

AdaGrad, in summary, conducts greater changes for more inadequate parameters and more slight changes for less inadequate ones. It performs well with inadequate data and in large-scale NNs of training. Nevertheless, during training deep NNs, its monotone learning rate is frequently excessively aggressive and stops learning too soon, that means AdaGrad makes learning rate change adaptively in iterations and has an ability to train spare data as well, but it will make dead neuron issue when the NN is deep the learning rate becomes very small number [29].

So, AdaDelta is an Adagrad addition that aims to minimize its belligerent, monotonically declining learning rate, when using AdaDelta, it is not essential to set a default learning rate, but the cost for computation is expensive [29].

In addition, RMSprop makes a very modest adjustment to the Adagrad method in trying to minimize falling learning rate critically. Specifically, the RMSprop helps the learning rate automatically be

tweaked and picks a different learning rate for every single parameter, however, it makes the training model very slow [29].

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

*Figure 16: RMSprop function [29]*

There is a division of the learning rate with an exponentially decaying average of squared gradients by RMSprop. It is recommended that  $\gamma$  be set to 0.9, with 0.001 as an acceptable default value for the learning rate  $\eta$  [29].

Adam stands for Adaptive Moment Estimation, is one of the most favored and usually utilized GD optimization methods, which is a calculating manner for adaptive learning rates for every single parameter, stores the median decay of preceding gradients, comparable with momentum as well as the median of decaying in preceding squared gradients, similar to RMS-Prop and AdaDelta. It incorporates the benefits of both strategies. Furthermore, Adam brings some benefits that are easy to implement, computationally efficient, using little memory requirements, converging rapidly, rectifying vanishing learning rate and high variance, but its computation is costly [28].

$$\begin{aligned}
 m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\
 v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \\
 \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \\
 \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t.
 \end{aligned}$$

Figure 17: Adam function [28]

In Figure 17,  $m(t)$  and  $v(t)$  respectively are numbers of the first moment being the average and the second one being the non-centered variance of the gradients. There is a set of default numbers for  $\beta_1$  equals 0.9,  $\beta_2$  equals 0.999, and  $\epsilon$  equals  $10^{-8}$ .

### 3.5. Convolutional Neural Network

Convolutional Neural Networks (CNNs), which like ANNs, are encouragement from the structure of the human visual cortex [8]. CNNs, which are a type of NN typically utilized for duties in CV in deep learning, have the consistency of convolutional layers, pooling layers, and fully connected layers. These networks are identical to traditional NNs, except that instead of generic matrix multiplication, a kernel of convolution is utilized in a or several layers of them [3]. These networks have the role of image reduction becoming another easier formation for processing, which help not disappearing critical features to achieve an acceptable prediction. Moreover, images are tensors in 3 dimensions of width, height, channels, and CNNs have the ability to take the tensor as

an input, identify which image characteristics or features are essential for classification and its output [1] [7].

Therefore, the basic structure of a Convolutional Network in order starting from a Convolutional Layer or so-called convolution process (do not count the initial input layer), has the effect of picking out the 3-dimensional specificities of the image input. Particularly, the features of the input original image are passed through the “convolutional layer”, at which there is a mechanism built up by matrix multiplication between the input image known as a matrix and one or more matrices with a certain size (usually smaller than the input matrix) being called “filters”, this matrix multiplication will be shifted by rows and columns with a certain rule that will generate one or more new matrices based on the number of filter layers, the product from this process is called feature maps which contain the attributes of the original image’s input.

The pooling layer helps the network synthesize the features of the image in a deeply specific way (max pooling) or more objectively general (average pooling or mean pooling) from the preceding layer’s output. Next, the size of the image has been significantly reduced which is very meaningful for memory in process of the training model. Although the dimensions are decreased, the number of channels or feature maps is still kept unchanged. So, this process is like a composition of a reduction for the size of the image’s input feature in length and width as well as a reduction of the number of learnable parameters. At last, when the input size has been reduced to a reasonable value, the 3-dimensional image will be converted to dimensional vector or known as "flatten layer", at which this process is generally implemented as a Fully Connected Layer, its units will correspond to several classes that defined for the network before, relied on specific AF,

the final network's output will produce a specific final result in a range of the classes.

In general, using CNNs because of their capacity to learn translation-invariant patterns as well as spatial hierarchies utilizing appropriate filters. The invariant patterns of learning translation mean to be once a pattern is learned in an area of a provided image, and the network will have the ability to recognize it anywhere in the image. This improves image processing productivity by requiring fewer samples of training for learning generalizable depictions [1].

Hierarchies by space show that earlier layers in the network can learn tiny patterns of local and grow up to bigger patterns composed of these tiny patterns of next layers, which also improves the network's potentiality of learning complicated and abstract concepts of visualization. In addition, this also allows these networks to deliver class predictions with far higher accuracy than vectorization of a complicated image relied on the reliance of pixels throughout Therefore, CNNs have to be used for any image classification task so as to pursue these crucial properties and correlations between features [1].

### 3.5.1. Convolution Layer

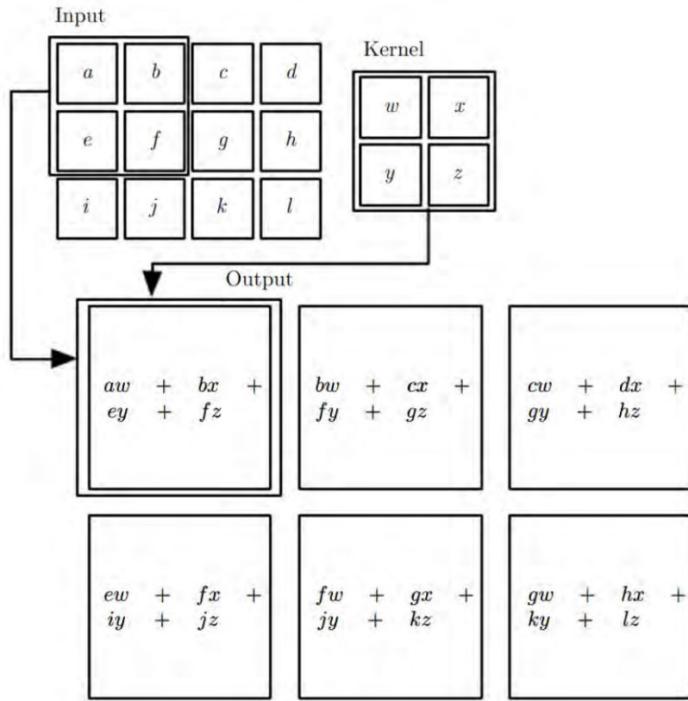
Convolution is an operation that produces a new function by combining two functions of an argument with real value. Let  $s(t)$  is the output estimate function relied on time  $t$ , and  $x(a)$  is the age-based function off input position, and  $w(a)$  is a function of the weight prioritizing measurements in recent. This is the universal formula for convolution utilizing this defined function:

$$s(t) = \int x(a)w(t - a)da.$$

*Figure 18: Convolution Function.*

In detail, the input is the function  $x(a)$ , the kernel is the weighting function  $w(a)$ , and the output  $s(t)$  plays the role of feature map in the CNN. the CNNs use convolutions that are operated over two-dimensional tensors being made up of the input images' width, height, and channels of color. These techniques take out patches and alter the input for building a feature map with varied depth. One of the two critical criteria for defining these convolutions is the extracted patches size. The depth of the output feature map is the second parameter to consider. This is several filters built relied on the layer that is able to encode diverse features of the input data.

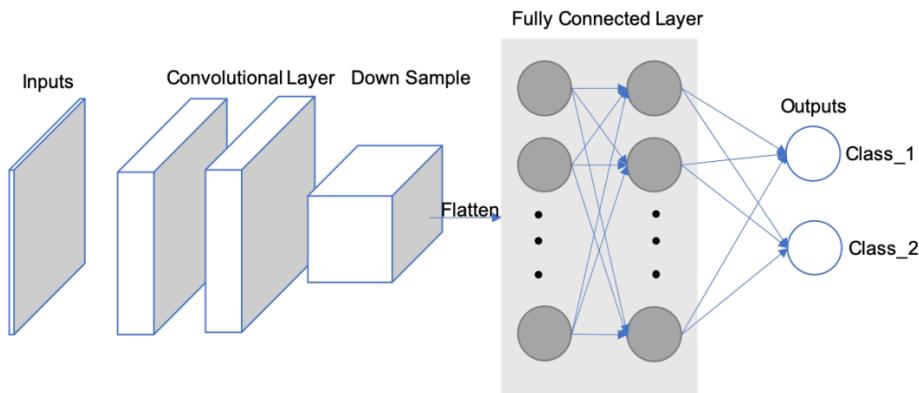
Two – dimensional convolutions are computed by moving a window with a squared shape of defined width and height across all of the input feature map's pixels, which connect to each local region or receptive field of the image corresponding to the size of the applied filters. For images, the input feature map has three dimensions which are width, height, and channels for color, commonly representing red, green, and blue. When these feature maps perform 2D convolution, there is a creation of two-dimensional patches with features encompassing. Next, these patches are turned into a one-dimensional vector reflecting the output depth via a convolution kernel. After that, the vectors are spatially reassembled into an output map with two-dimensional being equivalent with all points in the input map. This progress of convolution is depicted in Figure 19 below.



*Figure 19: Feature maps are computed by stride mechanism of convolution process [3].*

In other words, a feature map is a convolutional layer's output, which is typically a three-dimensional tensor with dimensions width, height, and depth. To process inputs or feature maps, the convolutional layer employs a filter, similar to the filtering procedure in picture pre-processing. The primary distinction is that in standard image pre-processing, the filter weights (values) are hand-crafted, whereas in CNNs, during training, these weights are learned. Alternatively, the receptive field in the input feature map corresponds to a convolutional layer's output, and the the receptive field size is governed by expansion and the kernel size. It is not essential to flatten images while using CNNs. However, flattening is required to feed the features maps, being the outputs of convolutional layers, to the fully connected layers. Fully connected layers are the same as the ANNs mentioned above, however,

ANN is a catch-all term for all artificial neural networks, including CNNs. Fully connected layers understand the linear and non-linear correlations between extracting features and classifying each sample into many abstract classes. Figure 20 shows a simple CNN architecture.



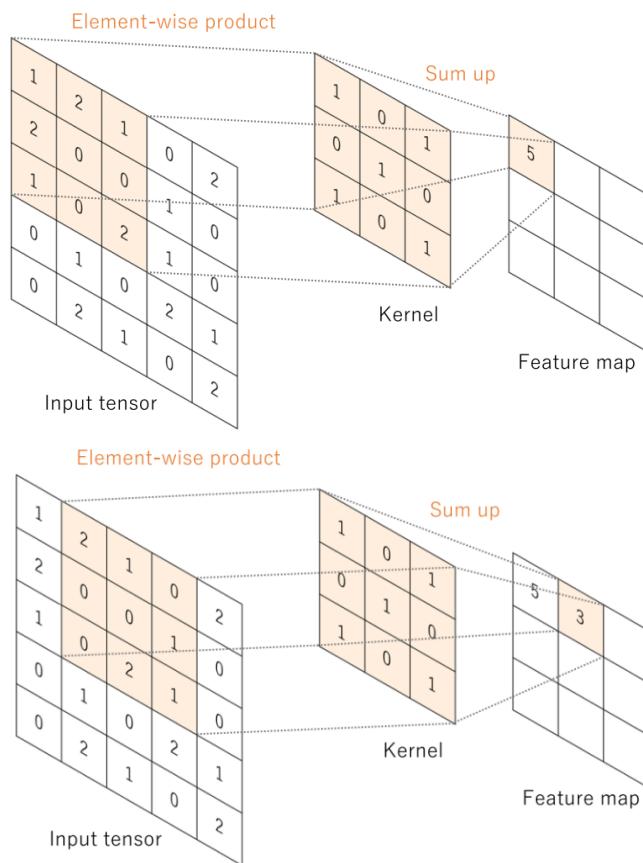
*Figure 20: CNN Architecture.*

There is navigation from hyper-parameters imposing to the output volume's size, they are stride, depth, and zero-padding.

The first component of the output volume decision is depth, which is how many filters can be utilized. Each of them learns to search for some distinctness from the input. If the earliest Convolutional Layer receives the image in raw as an input, distinct neurons along the dimension of depth are able to fire in the attendance of edges with different orientations or color droplets. A column of depth is a group of neurons being all staring at the same region of the input [9].

Stride aids in picture and video data compression tuning. The stride of the convolution is a characteristic that is able to create different sizes of the output in CNNs, which has the ability to make a contribution to a divergent size for the output in CNNs, which is a parameter of convolutional operation defining the distance among the patches

acquired from the feature map's input. With a value is 2 for a stride, the output feature map in width and height is down-sampled via a factor of two with no padding. For example, if the stride is set to 1 in a NN, step by step, the filter will make a reflection onto the matrix of input, which moves one pixel or one unit at a time from left to right in the row till the end of the horizontal point and top to bottom in the column at the end of the matrix. Because the size of the filter influences the encoded output volume, the stride is frequently a positive integer rather than a fraction or decimal.



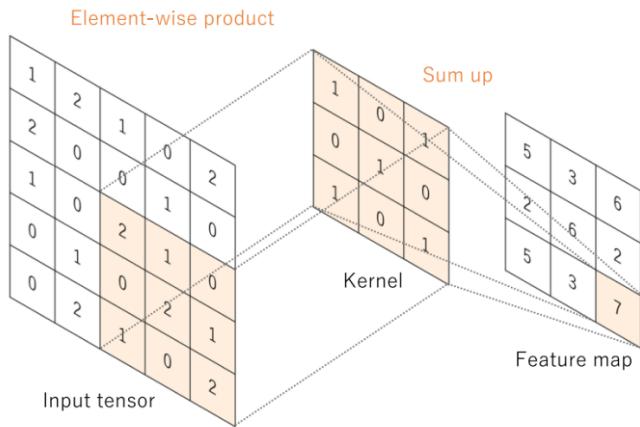


Figure 21: Generating feature map process.

It is sometimes useful for padding the input volume by the around border with zero value. This padding with a size of zero is a hyperparameter. The great thing about zero padding, which allows tweaking the output volumes' size by space, most typically using it for precisely preserving the input volume's size by space so as to be the same as the input and output's width and height [9].

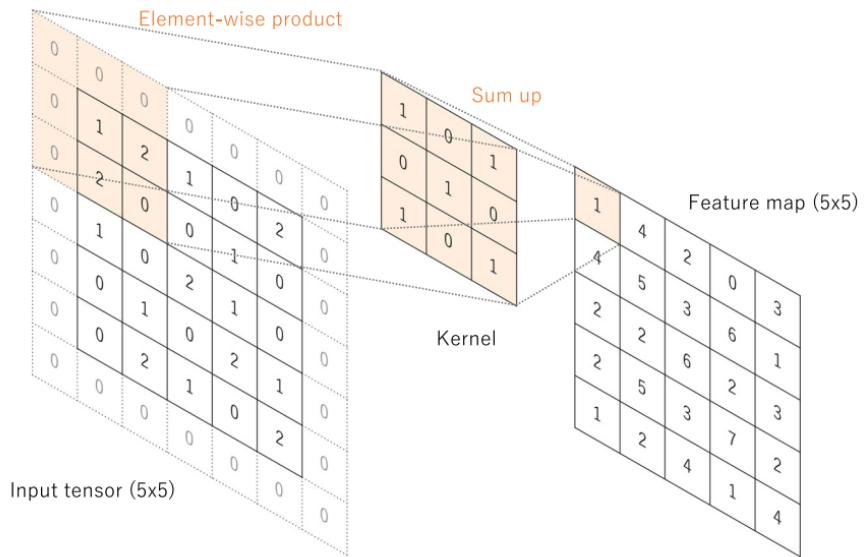


Figure 22: Generating feature map process with zero padding.

A combination of these components will give out the formula for calculating the size of the Convolutional layer:

$$[(N + 2P - F) / S + 1] \times [(N + 2P - F) / S + 1] \times C'$$

*Figure 23: The formula of output size of the Convolutional layer.*

The dimension by space of the output volume may be calculated like an input volume size ( $N$ ) function which represents for the dimension by width and height of input image, the receptive field size of the Convolutional Layer neurons ( $F$ ) as the dimension of the filter by width and height, the zero-padding number placed the border for input image ( $P$ ), the positive integer number of strides with which they are applied ( $S$ ), and the number of filters ( $C'$ ).

The results of an operation in linear such as convolution, which is subsequently processed by a non-linear AF. Despite being preceding utilized smooth non-linear functions like a function of sigmoid or tanh because of being mathematical representations of biological neuron behavior, the ReLU is now the most popular non-linear AF in utilization, simply computing the function in form  $f(x) = \max(0, x)$

In convolutional neural networks, rectified linear units (ReLUs) provide three key advantages:

- The ReLUs efficiently transmit the gradient, reducing the risk of a vanishing gradient problem, which is prevalent in deep neural architectures.
- The ReLUs set negative values to zero, so solving the problem of cancellation and resulting in a much sparser activation volume at its output. Sparsity is helpful for a variety of reasons;

however, it is most commonly used to supply robustness to tiny adjustments in input such as noise.

- The ReLUs include just simple computations (primarily comparisons) and are thus substantially more efficient to implement in convolutional neural networks.

### 3.5.2. Pooling Layer

There are also pooling layers between convolutional layers. They are used to downsample the feature maps and make a decrement of parameter numbers. There is an essential of a huge number of parameters for the following layers after extraction of feature, the reason is having an existence of exponential increment of the determination in depth by the number of the following layers' channels. That makes an increment in parameter number as well as the quantity of the neural network's computation. Hence, to reach the purpose of the computational loading reduction, it is necessary to make a reduction of either the dimensions of the input matrix block or quantities of layer units. Therefore, needing a reduction of the input matrix block's dimensions or a reduction of the layer unit number to pull down the computational load. Since every individual unit will present a result of applying a filter for finding out a specific feature, the cutback for the unit number will be impossible. Another key point, finding a value of representation for every single region by space passing through by the filter to cut down the size of the input matrix block, which only has the ability to reduce the image size, but is not able to change the image's main contours. As a consequence, the process of reducing a matrix is applied.

Average pooling and max pooling are the two most popularly utilized types of pooling layers helping make this reduction purpose. As

the suggestion of their names, average-pooling takes out the mean of the feature map values corresponding to kernel size. Besides, max-pooling calculates the maximum value in the neighborhood corresponding to the kernel size position on the feature map. Both of them travel through the whole feature map by a stride size without overlapping local region positions. In addition, the pooling layers do not have any parameters which need to be learned during training and the stride in the convolutional layer is able to also be used to replace the pooling layers. Figure 24 shows the difference between these two pooling types.

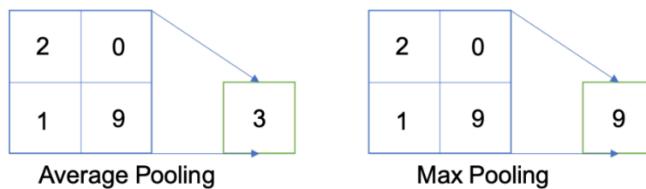


Figure 24: Average (Mean) and Max Pooling Layer.

### 3.5.3. Batch Normalization (BN)

Another critical component of a CNN architecture is batch normalization. Although this is not a required component in CNN, it is a relatively important component with many benefits for training the model to reach the best results. BN allows faster training and stabilization of deep neural networks by stabilizing the distribution of layer inputs during training. This approach mainly involves Internal Covariate Shift (ICS). To improve model training, it is important to reduce ICS by controlling the means and variances of the input data layers. In detail, BN normalizes the feature maps at a convolutional layer's output in the same way which it normalizes the input pictures. BN makes subtracts the batch mean and divides the feature map values by the batch standard deviation for ensuring that the feature maps'

distribution is the same. As a result, BN lowers value shift in hidden layers and BN allows to obtain additional flexibility in the initialization of kernel weights. Furthermore, because of the guarantee of BN which activation does not diverge to very large values, greater learning rates can be employed. In addition, BN is quite similar to a dropout layer which also has some regularization effects, because of adding noise to the feature maps. During training and evaluation, both the BN and dropout layers will behave differently.

When we would like the “function of probability density” (FPD) of the model to be closest to the true real distribution of data as possible in deep learning. Unfortunately, in fact, that we only have a narrow amount of training data at our disposal. As a result, the goal is to get the FPD of the model as near to the FPD of the train as possible by training on a dataset that replicates the domain of application in the real world. A sufficiently big training dataset is required to get improved representation. When the set of training data is insufficiently huge, there will be a significant difference in the error between the FPD of real and the FPD of the train.

In other expressions, even if we produce a model FPD that is very close to the FPD of the train, it will not perform with the FPD of test skillfully, being a sample of the FPD of the real. This is referred to as the overfitting problem. Overfitting occurs when a model produces an impressive performance in the training data set and a poor performance in the data set for testing. However, the model is not overfitted when performing impressive in training and a bit worse in testing. The dropout layer is applied to stay away from overfitting by randomly selecting some neurons to brush aside throughout the training process. A dropped-out node's incoming and outgoing edges are also eliminated. Thus,

dropout not only reduces the problem of overfitting but also simultaneously speeds up training by eliminating training with all nodes.

### 3.5.4. Fully Connected Layer

Typically, there is an applying of AF at the last connected layer that is different from the others. Each job necessitates the selection of an appropriate activation function. An SF is utilized as an AF in the multiclass classification job to work as a normalization for real values of output from the ultimate "fully connected layer" to probabilities of goal class, in which every single value is a real number in the range from 0 to 1 and these values will be made a sum to 1. Typical final layer AF selections for several sorts of jobs.

In addition, the majority of the trainable parameters are located in fully connected layers in CNN. Fully connected layers can give out outputs that are categories for image classification or localization via bounding boxes [11]. Convolutional layer work can also be used as fully connected layers to execute the same duty more efficiently. Instead of a class, the achievement of an image segmentation job is a mask with the same resolution as the input pictures. We may use a 1x1 kernel as the output layer to reduce the depth to the same level as the target classes [10], giving the model pixel-wise classification capability.

Training a CNN consists of two steps: forward and backward. We feed the CNN with input information flow in the forward. To acquire the input for the following layer, feature maps are generated in each layer with predefined weights and biases. A loss value will be computed at the output layer by applying a function of loss for the non-similar between the calculated output and the ground truth. The chain rule is utilized in the backward step to find out the gradients of the loss value for each

trainable parameter. After that, the gradients are utilized to adjust each parameter for the following iteration. This type of update is what allows CNNs to “learn”.

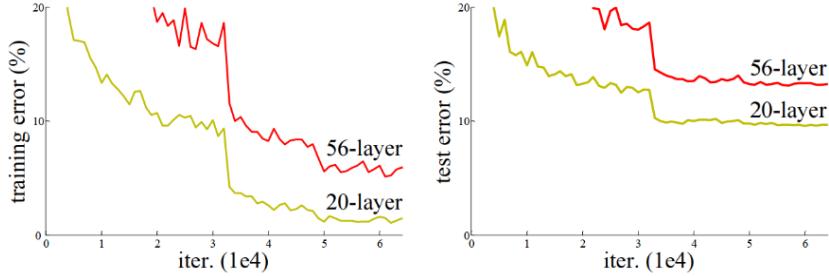
After processing enough iterations, at which the loss value converges to an acceptable small number that means it is approximately close to zero, the training process can be terminated. The loss function computes the difference or error between the outputs and the ground truth. Minimizing loss implies minimizing the difference between the model’s function of probability density and training one when the loss function is applied as a guide. Moreover, the loss function will change depending on the application and objective. Because of its ability to quantify the similarity/difference between two distributions, cross-entropy is a widely used loss function.

Otherwise, there are several optimizer options to help you decide how to update the network parameters. Adam and Stochastic Gradient Descent (SGD) are two options. In general, SGD will produce better results while learning at a slow pace. It will take several iterations to achieve a satisfactory outcome. SGD has a fixed learning rate that does not vary during training. Adam, on the other hand, converges quicker since it employs a vector of learning rates that are adjusted as learning continues during training progresses.

### 3.6. Residual Network (ResNet)

ResNet, an abbreviation for Residual Network, is a form of NN introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in their paper "Deep Residual Learning for Image Recognition". The ResNet models were immensely successful with winning first place in the competition of the ILSVRC 2015 for classification with a top-5

error proportion of 3.57 percentage (An ensemble model), as well as first place in the competitions of ILSVRC and COCO 2015 in ImageNet both detection and localization, Coco detection, and Coco segmentation [20]. Furthermore, ResNet-101 was used to replace VGG-16 layers in Faster R-CNN. They discovered a substantial boost with a proportion of 28 percentage. ResNet was developed because of the vanishing or exploding gradient issue on some state-of-the-art model architectures at that time.



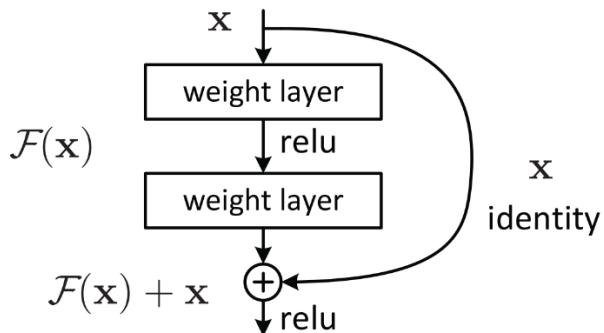
*Figure 25: Performance of 20 layers and 56 layers plain networks on CIFAR-10 [20].*

Before the ResNet, with the advent of neural networks deeper and deeper by a common argument, which is "go deeper", growing up the depth of the network will be an improvement for the quality of the network. Based on that, several new DNN architectures have been born such as AlexNet, GoogleNet, or VGG, which apply the mechanism to increase the number of layers or the number of layer groups. Because these added layers are able to solve complicated issues more efficiently as the non-identical layers could be trained for various duties to get high results of accuracy. Additionally, the number of stacked layers can enrich the features of the model. However, with an increasing number of layers of complexity in deeper CNNs, new trouble has also emerged called vanishing gradients. In specific, as deeper networks may begin to

converge, the accuracy of the network gradually saturates to a certain threshold and then slowly moves down during the backpropagation step, and higher error increases may occur. This issue goes against the purpose of reducing error and increasing accuracy during network model training. In other words, the values go through the gradient step that is chain-rule by the calculation of partial derivatives method, which will quickly return a very small result that approaches 0 after several chain-rule applications [22]. It will lead the learning of the network to be stopped soon, so this backpropagation will not be executed in previous layers, in this case, it may be stopped at an unknown hidden layer. This problem of training very deep networks has been alleviated with these ResNets which are made up of Residual Blocks (RBs).

### 3.6.1. Residual Blocks (RBs)

ResNet takes advantage of using RBs to increase model accuracy. The strength of this form of the neural network is the concept of "skip connections" or skip paths, which is at the heart of the RBs. There are two ways for the skip connections to work. At first, they address the issue of vanishing gradients by creating an alternate path as a shortcut for the gradient to go through. Second, they also allow the model to learn a function of identity, which assures that the model's upper levels will have a performance not being worse than the layers laid underneath.



*Figure 26: Residual Block Unit with skip connection [20].*

The construction of an RB is depicted in the Figure 26. The ResNet is developed to create its stack of 2 layers being able to fit  $F(x) = H(x) - x$  rather than training only stacked convolutional kernels have an ability to be suitable to the intended mapping. Supposing that, the original mapping has been recast as  $H(x) = F(x) + x$  [20]. They claim that optimizing the residual mapping is substantially easier, which is  $F(x)$ . In spite of being the optimal option of the identity in an extreme circumstance, the model will give out a result in  $F(x)$  being zero. Using such a framework, networks with more than 100 layers might be trained. Besides, this method appears to have a minor flaw when there is a difference between input dimensions and those outputs, which have the ability to occur with pooling and convolutional layers. When the dimensions of  $F(x)$  differ from those of  $x$ , there are two options:

- To up-sample its dimensions, extra zero entries will be padded to the skip connection.
- To make the dimensions equal, the linear projection approach is utilized, which is accomplished by attaching more  $1 \times 1$  convolutional layers to the input. In this example, the output is as follows:  $H(x) = F(x) + W(x)$ . In this case, adding an extra parameter  $W$ .

In short, the RBs help the layers acquire knowledge of identity functions substantially easier than plain networks. So that, ResNet raises up the performance of DNNs with additional neural layers while reducing the proportion of error. In other words, the skip paths combine

the outputs of prior layers with the outputs of stacked layers, allowing for considerably deeper networks to be trained than previously feasible.

### 3.6.2. ResNet Architecture

#### 3.6.2.1. ResNet – 34 Architecture

The ResNet-34 was the first architecture built, which included inserting shortcut connections into a plain network to transform it into a counterpart of its residual network. The plain network in this case was influenced by VGG neural networks (VGG-16, VGG-19), while the convolutional networks had 3 x 3 filters. On the other hand, ResNets have fewer filters and are less sophisticated than VGGNets. There is a comparison between the 34-layer ResNet attains 3.6 billion FLOPs of performance and larger VGG-19 with 19.6 billion FLOPs [20]. ResNet 34 takes a proportion of 18 percentage of VGG-19, thus, the VGG-19 has more filters and higher complexity than ResNet 34.

The ResNet-34 also adhered to two simple design principles that the layers must have the same filter amounts for the same size of the feature map's output, and the filter amounts are twofold if the size of the feature map is cut off a half to maintain the time complexity for each layer [21]. And one more thing is 34 weighted layers being involved in it. In addition, this plain network plus shortcut connections. Despite being identical in the input and output dimensions, the identity shortcuts were employed by a direct route. There were two choices for considering when the dimensions were raised. The first one was the shortcut would continue to execute identity mapping while padding extra zero entries for an increment of dimensions. The projection shortcut could also be implemented to fit dimensions.

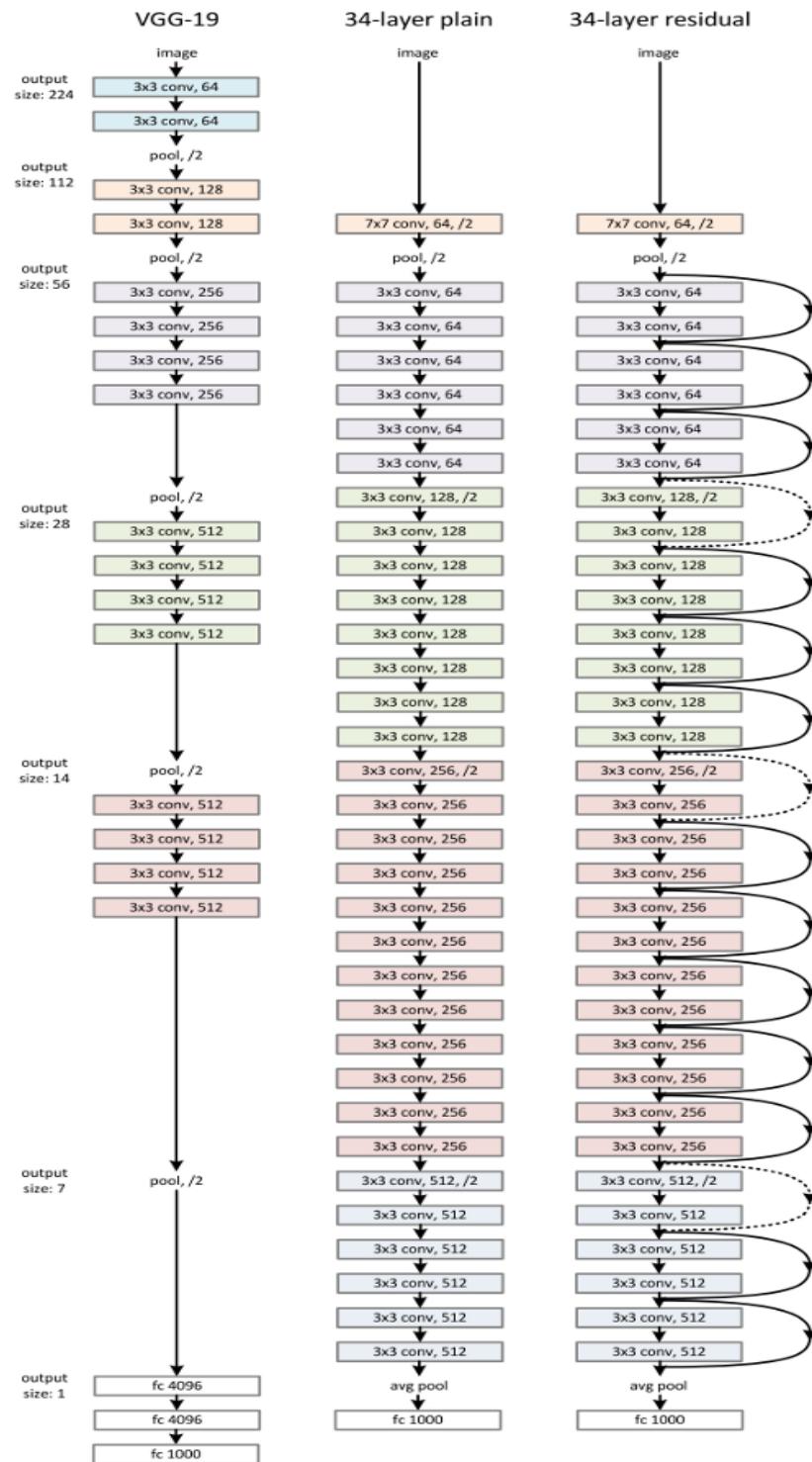


Figure 27: VGG-19 model, 34 Plain Network, ResNet 34 [20].

### 3.6.2.2. ResNet-50, ResNet-101, and ResNet-152 Architecture

Although the ResNet-50 architecture is based on the above model, there is one significant variation. In this situation, the RB was re-created as a design of bottleneck appearance cause of concerns about the requirement of time for training the layers. Instead of using two layers, a three-layer stack was used for building RB. As a result, each of the ResNet-34's 2-layer blocks was taken a place of a 3-layer bottleneck block for the construction of ResNet-50, which is known as “Deeper Bottleneck Architecture”. This model is generally more accurate than the 34-layer ResNet model with a performance of 3.8 billion FLOPS compared with 3.6 billion FLOPS [20].

Furthermore, Larger Residual Networks are the 101-layer ResNet-101 and ResNet-152, which are built with additional 3-layer blocks as well as ResNet-50 [21]. Even with an increment of network depth, the 152-layer ResNet reaches a very high number of FLOPS, it is about 11.3 billion and has specifically lower complexity than VGG-16 or VGG-19 nets with the number of FLOPS about 15.3 and 19.6 billion in respective.

## 3.7. EfficientNet

Following the previous success of AlexNets, GoogleNets, VGGs, ResNets, Gpipe, RCNN, or Mask RCNN with outstanding results in image recognition contests as well as the impressive accuracy in image recognition tasks bringing in reality. EfficientNet was born in 2019 by Quoc V. Le et al from GoogleBrain. Based on inspiration from the process of scaling up ConvNets, these researchers systematically studied it and came up with a completely new solution compared to the state-of-the-art solutions in precedent, which is scaling up the main component of

the network model in 3 dimensions: depth - scaling to add more layers to the model, width - scaling to add more channels for layers, resolution - scaling in resolution is also known as the size of the layers, feature maps, they called it the "compound scaling method"[25].

Before going into more detail about EfficientNet, we will learn about the reason why the Model Scaling definition of the previous network models is the fundamental foundation to construct the EfficientNet.

### 3.7.1. Model Scaling

In fact, the state-of-the-art models are built by Model Scaling method, and "go deeper" is also one of these methods. ResNet is able to be scaled up or down by modifying the layers of the network - depth, whereas WideResNet and MobileNets have the ability to scale based on adjustment of the network's channels - in width. Besides, the network model is also widely acknowledged that larger input image sizes improve accuracy while incurring the burden of additional FLOPS [25]. Although several previous pieces of research have proven that width and depth of network are both crucial for the powerful expression of the ConvNets. It is still not easy to productively scale a ConvNet for a better achievement of accuracy and efficiency.

**Scaling In Depth:** Many ConvNets, including ResNet and Inception Net, scale network depth in this manner. The assumption is that deeper ConvNet is capable to capture more affluent and more complicated characteristics of an image while also generalizing successfully on new tasks. However, because of the vanishing gradient problem, deeper networks are harder to train. In spite of having numerous approaches as ResNet-skip connections and batch

normalization to mitigate the training issue, the accuracy gain of very deep networks will drop, for example: Despite having so many more layers than ResNet-101, ResNet-1000 has equal accuracy [23].

**Scaling In Width:** For small-scale models like MobileNet, MobileNetv2, and MnasNet, scaling network width is widely applied. Wide residual networks of Zagoruyko and Komodakis in 2016 explored how wider networks can take more fine-grained features and are easier for training [25]. On the other hand, outstandingly wide but shallow networks, having a difficulty in catching the features at a higher level. Therefore, when networks expand substantially wider with bigger channels, the accuracy quickly saturates [23].

**Scaling In Resolution:** ConvNets may be able to catch more fine-grained patterns with higher resolution input photos. Early beginning with 224x224 with ConvNets, later ConvNets often employ 299x299 or 331x331. And in 480x480 resolution [23], GPipe just achieved state-of-the-art accuracy of ImageNet. In object detection ConvNets, higher resolutions, such as 600x600, are also commonly employed [23].

So, although the accuracy of the model will be enhanced when making a scale-up at any single network's dimension in width, depth, or resolution, the gain of accuracy reduces in larger models.

### 3.7.2. Compound Scaling Method – EfficientNet

Based on the above 3 options of the Model Scaling method, the EfficientNet's researchers have come up with the compound scaling method which is a uniform combination of 3-dimensional expansion in the width, depth, and resolution of the network, with a set of the ratio, is fixed, this aims to create a balance in the dimensions of all 3 dimensions [25].

$$N = F_k \odot F_{k-1} \odot \dots \odot F_1(X_1) = \bigodot_{j=1 \dots k} F_j(X_1)$$

$$N = \bigodot_{j=1 \dots s} F_i^{L_i} X_{\langle H_i, W_i, C_i \rangle}$$

Figure 28: Common ConvNets equation [25].

The formula in the Figure 28 presents the common structure of the network model based on the ConvNets layer, the ConvNets layer  $i$  was defined as an equation  $Y_i = F_i(X_i)$ , where  $F_i$  is an operator,  $Y_i$  is an output of tensor,  $F_i^{L_i}$  presents  $F_i$  layer iterating  $L_i$  times at  $i$  state,  $X_i$  is tensor input, spatial dimensions are  $H_i$  and  $W_i$  as width and height, and the  $C_i$  represents for the number of channels, all of these things make a combination for a list of layers. In fact, the Convnet layers are divided into plenty of stages and all the layers in each state have the same architecture such as ResNet [25]. EfficientNet expands three dimensions of width ( $C_i$ ), length ( $L_i$ ), and/or resolution ( $W_i, H_i$ ), this action makes no change for  $F_i$ . Based on fixing  $F_i$ , the issue of the design for limitations of new resource will be simplified by model scaling, however, there is an existing of a huge space of design for discovering various of these 3 dimensions in every single layer. So, there is a limitation that all layers have to be scaled consistently with an unchanged ratio to get a further reduction of design space. It serves for purpose of maximizing the model accuracy in any case of having any limitation of provided resource. [25]. Formula below shows the optimal solution of the model scaling, where  $(w, d, r)$  are coefficients for purpose of scaling up the network's dimension in width, depth, and resolution.  $F^{\wedge}_i, L^{\wedge}_i, H^{\wedge}_i, W^{\wedge}_i, C^{\wedge}_i$  are predefined parameters in the baseline network.

The model can achieve better accuracy and efficiency, more specifically this ratio is followed by this principle:

$$\begin{aligned}
 \text{depth: } d &= \alpha^\phi \\
 \text{width: } w &= \beta^\phi \\
 \text{resolution: } r &= \gamma^\phi \\
 \text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 &\approx 2 \\
 \alpha \geq 1, \beta \geq 1, \gamma \geq 1
 \end{aligned}$$

*Figure 29: The setting of coefficient values in the compound scaling method [25].*

Where  $\alpha, \beta, \gamma$  are invariable numbers having the potentiality of being affirmative by a tiny search by the grid. Visually,  $\phi$  (Phi) is a coefficient able to adjust and increase the number of available resources for model scaling by user's designation, while  $\alpha, \beta, \gamma$  are respective to the size of the width, depth, and resolution that are additional to the network [25]. The FLOPS of a normal convolution operation is proportional to  $d, w^2, r^2$ , i.e., increasing network depth twice, will make a double increment of FLOPS too, however, if there is a double gaining of network width or resolution, it will be a cause of a quad-increment for FLOPS [25]. That said, convolution operations often prevail over the cost of reckoning in ConvNets, when making a scale for a ConvNet with the above equation, it will nearly improve overall FLOPS by  $(\alpha \cdot \beta^2 \cdot \gamma^2)^\phi$ . And the default value of the equation was set:  $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$  in order to be with any new  $\phi$ , the total FLOPS will get an approximate increment of  $2\phi$ .

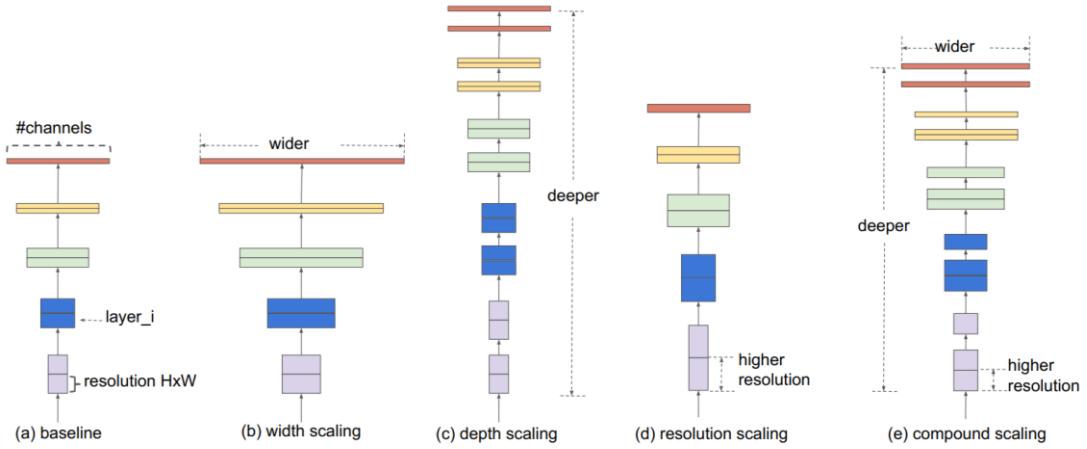


Figure 30: Model Scaling: (a) baseline network; (b) – (d): the networks only scale up a dimension in width, depth, or resolution; (e): method of compound scaling network has a scaling uniform of 3 dimensions with a fixed ratio [25].

### 3.7.3. EfficientNet Architecture

Because model scaling did not modify layer operators  $F^i$  of baseline network and had a valuable baseline network is also essential. The EfficientNet was built based on existing ConvNets with scaling method, in specific, they are MobileNet and ResNet, the mechanism is with a manipulation of a multi-objective neural architecture search, it made an optimization in both accuracy and FLOPS.

Indeed, manipulating  $ACC(m) \times [FLOPS(m)/T]^w$  as a goal of optimization, where  $ACC(m)$  plays the accuracy,  $FLOPS(m)$  plays FLOPS of model m, T presents the target FLOPS and  $w = -0.07$  stands for a hyperparameter for adjusting the exchange between accuracy and FLOPS [25]. This compound scaling method was applied with two steps:

- Firstly, there is a fixed  $\varphi = 1$ , assumption of having an availability of double resources and more, then making a tiny grid searching of  $\alpha, \beta, \gamma$  relied on Equation 2 and Equation 3.

Particularly, finding the best values for EfficientNet-B0 are  $\alpha = 1:2$ ,  $\beta = 1:1$ ,  $\gamma = 1:15$ , under fixed value of  $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ .

- Secondly,  $\alpha$ ,  $\beta$ ,  $\gamma$  will be fixed as constants and expanded baseline network by different  $\phi$  utilizing Equation 3 for achieving EfficientNet-B1 to B7.

$$\begin{aligned} & \max_{d,w,r} \text{Accuracy}(\mathcal{N}(d, w, r)) \\ \text{s.t. } & \mathcal{N}(d, w, r) = \bigodot_{i=1 \dots s} \hat{\mathcal{F}}_i^{d, \hat{L}_i}(X_{\langle r \cdot \hat{H}_i, r \cdot \hat{W}_i, w \cdot \hat{C}_i \rangle}) \\ & \text{Memory}(\mathcal{N}) \leq \text{target\_memory} \\ & \text{FLOPS}(\mathcal{N}) \leq \text{target\_flops} \end{aligned}$$

Figure 31: Optimal formula of scaling up method [25].

Outstandingly, this strategy is feasible to attain even higher performance by directly looking for  $\alpha$ ,  $\beta$ ,  $\gamma$  from a huge model, however, the cost for searching becomes extremely expensive for more huge models. This method overcomes this problem by conducting a single search on the tiny baseline network at the first step and then utilizing the identical coefficients of expanding for all other models at the second step [25].

Stage $i$	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	$224 \times 224$	32	1
2	MBCConv1, k3x3	$112 \times 112$	16	1
3	MBCConv6, k3x3	$112 \times 112$	24	2
4	MBCConv6, k5x5	$56 \times 56$	40	2
5	MBCConv6, k3x3	$28 \times 28$	80	3
6	MBCConv6, k5x5	$14 \times 14$	112	3
7	MBCConv6, k5x5	$14 \times 14$	192	4
8	MBCConv6, k3x3	$7 \times 7$	320	1
9	Conv1x1 & Pooling & FC	$7 \times 7$	1280	1

Figure 32: Optimal formula of scaling up method [25].

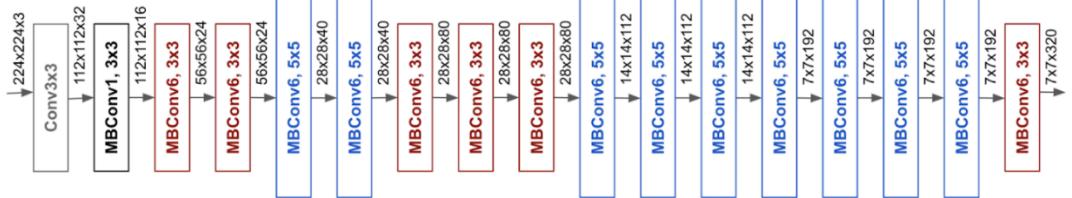


Figure 33: EfficientNet-B0 [25].

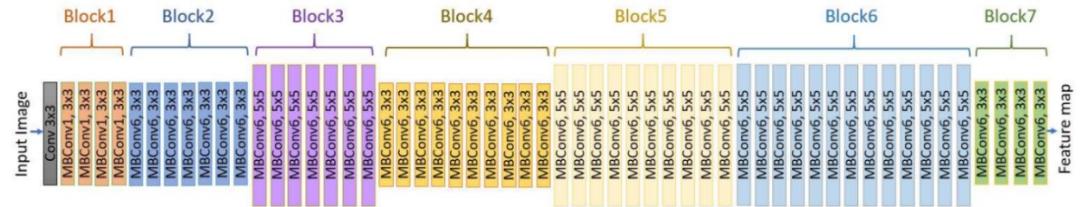


Figure 34: EfficientNet-B7 [31].

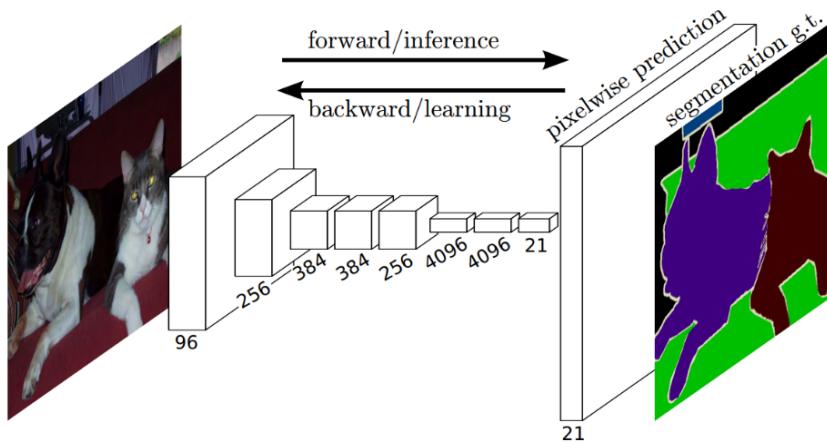
As a result, EfficientNets give out an outstanding performance to other ConvNets. Specifically, EfficientNet-B7 got over the best-existing accuracy of GPipe by utilizing 8.4x lesser parameters than and executing 6.1x quicker on inference [24], it was about 66 million parameters and 37 billion FLOPS that reach a proportion of 84.3 percent of the accuracy. Adding, there was a comparison of ResNet-50 and EfficientNet-B4 that made an improvement of the peak of accuracy from 76.3 percent to 83.0 percent, increasing about 6.7 percent with similar FLOPS.

Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPs	Ratio-to-EfficientNet
<b>EfficientNet-B0</b>	<b>77.1%</b>	<b>93.3%</b>	<b>5.3M</b>	<b>1x</b>	<b>0.39B</b>	<b>1x</b>
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
<b>EfficientNet-B1</b>	<b>79.1%</b>	<b>94.4%</b>	<b>7.8M</b>	<b>1x</b>	<b>0.70B</b>	<b>1x</b>
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
<b>EfficientNet-B2</b>	<b>80.1%</b>	<b>94.9%</b>	<b>9.2M</b>	<b>1x</b>	<b>1.0B</b>	<b>1x</b>
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
<b>EfficientNet-B3</b>	<b>81.6%</b>	<b>95.7%</b>	<b>12M</b>	<b>1x</b>	<b>1.8B</b>	<b>1x</b>
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
<b>EfficientNet-B4</b>	<b>82.9%</b>	<b>96.4%</b>	<b>19M</b>	<b>1x</b>	<b>4.2B</b>	<b>1x</b>
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
<b>EfficientNet-B5</b>	<b>83.6%</b>	<b>96.7%</b>	<b>30M</b>	<b>1x</b>	<b>9.9B</b>	<b>1x</b>
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
<b>EfficientNet-B6</b>	<b>84.0%</b>	<b>96.8%</b>	<b>43M</b>	<b>1x</b>	<b>19B</b>	<b>1x</b>
<b>EfficientNet-B7</b>	<b>84.3%</b>	<b>97.0%</b>	<b>66M</b>	<b>1x</b>	<b>37B</b>	<b>1x</b>
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

Figure 35: EfficientNets Performance Result on ImageNet [25]

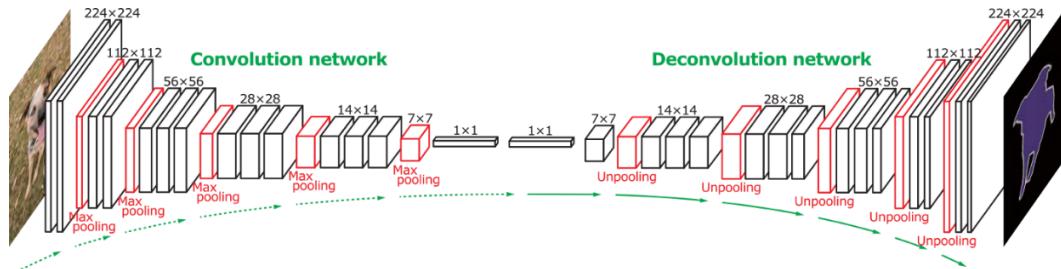
### 3.8. Fully Convolutional Networks (FCNs)

One of the turning points in the field of image detection was the introduction of the FCN, which is known as a variant of CNN for image segmentation work [10]. The FCN has a difference from ordinary CNNs in which the fully connected layers are taken over by convolution layers at the end of the network. So, there is a generation of a network making computation of a nonlinear filter for each layer's output vectors. As a result, not only can the completed network function on an input of any size and make a production of output with respective dimensions by space, but this allows the network for classification to generate a desiring class's heatmap. In addition, adding layers and a spatial loss to the network results in an efficient machine for end-to-end dense learning [10]. Figure 36 shows the transformation of FCN.



*Figure 36: Fully Convolutional Network serves for semantic segmentation tasks [10].*

The use of FCNs allows to take multiple input image sizes and to have better efficiency for dense predictions of learning via in-network up-sampling [10]. Furthermore, the FCN may preserve spatial information from the input that is very important for semantic segmentation tasks because of involving both localization and classification for the task. Although an FCN has the potentiality for accepting any size input image, there is a decrement of the output resolution by being applied convolutions without any padding. These were added to make filters minimal and requirements of calculation manageable. As a result, the coarse output is reduced in size by a factor equal to the pixel stride of the output unit's receptive field [10].



*Figure 37: Fully Convolutional Network Architecture [32]*

In other words, the FCN is divided into 2 parts that are convolution network (down-sample) as an encoder step and the following is deconvolution network (up-sample) known as decoder step, the convolution network part is very similar to casual CNN architecture so we will go deep down to the up-sample part.

In Semantic Segmentation, after using the method of convoluted features extraction to decrease the input's resolution, the following step is the job of upscaling the poor resolution turning back to the primitive resolution of the input image. Pooling is the process of converting a bunch of values to a solitary value, whereas up-sample is the process of converting a solitary value to a bunch of values and it can be named Un-pooling, like Pooling, the un-pooling can be carried out in a variety of ways [26].

- **Nearest Neighbour:** Choosing a value and populating it into the surrounding cells (number of cells depending on the increment in resolution), Figure 38 shows the copy process of all values in matrix  $2 \times 2$  to every cell with a size  $2 \times 2$  square in new matrix  $4 \times 4$  [26].

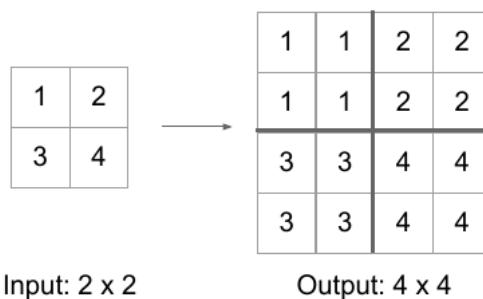


Figure 38: Nearest Neighbour Unpooling [26].

- **Bed of Nails:** Inserting the value in turn at a predefined cell and adding the rest with 0. Figure 39 is an example of filling

the value from the top-left cell of each  $2 \times 2$  square in a new matrix with a size  $4 \times 4$  [26].

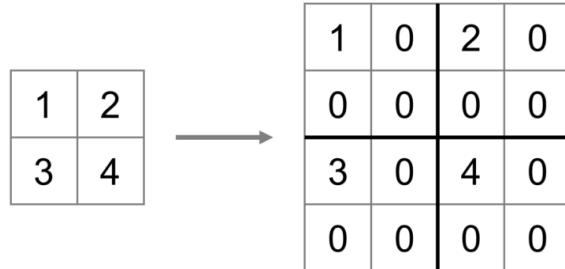


Figure 39: Bed of Nails Unpooling [26].

- **Max Unpooling:** In CNN, the max-pooling layer pulls the maximum values from the picture regions covered by the filter in order to down-sample the data, and the un-pooling layer provides the value to the location from which the values were picked up in order to up-sample the data [26]. However, these approaches are not data-dependent, which means they are able to learn nothing from data; they are just programmed computations making them become a specific task and costly in terms of calculation timing. The generic answer for these situations is transposed convolutions.

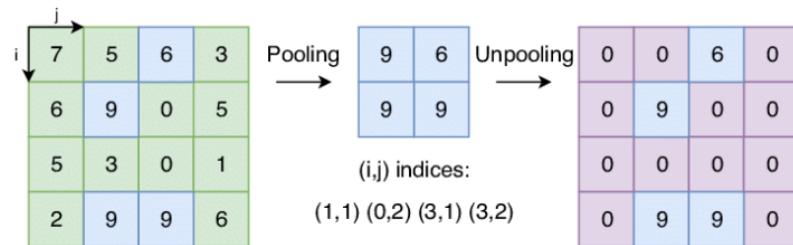
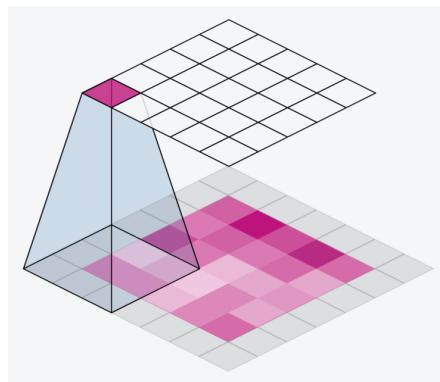


Figure 40: Max Unpooling [26].

Transposed Convolution, also known wrongly as Deconvolution, is the inverse action of Convolution. A transposed convolution layer, as opposed to convolution, is utilized to up-sample the decreased feature of

resolution giving back to its primitive resolution. For achieving the ultimate output from the features with lower resolution, a set of strides and padding values is learned. Transposed convolution is superior to other Up-sampling techniques because of being unlike earlier techniques (Nearest Neighbor, Bed of Nails, Max Un-pooling), the transposed convolution is a learnable up-sampling technique being highly efficient. The image below describes the method in a very simple way [26].



*Figure 41: Transposed Convolution [26].*

Next, because of an existing main problem with in-network down-sampling in an FCN, which is the reduction of the resolution of the input by a huge proportion, making it impossible to recreate finer features during up-sampling even with complicated techniques like Transpose Convolution. As a result of getting a coarse output. One solution is to include "skip connections" in the up-sampling from the previous layers stage and then make a sum of the total of the two feature maps. These skip connections supply enough information to the following layers to produce correct borders of segmentation. This combination of fine and coarse layers yields local forecasts with virtually accurate global (spatial) structure. The skip connection learns how to integrate coarse and fine layer information. So, there is a performance of grids with varying

degrees of spatial coarseness for layers, with the intermediate convolution layers of FCN deleted for clarity [26].

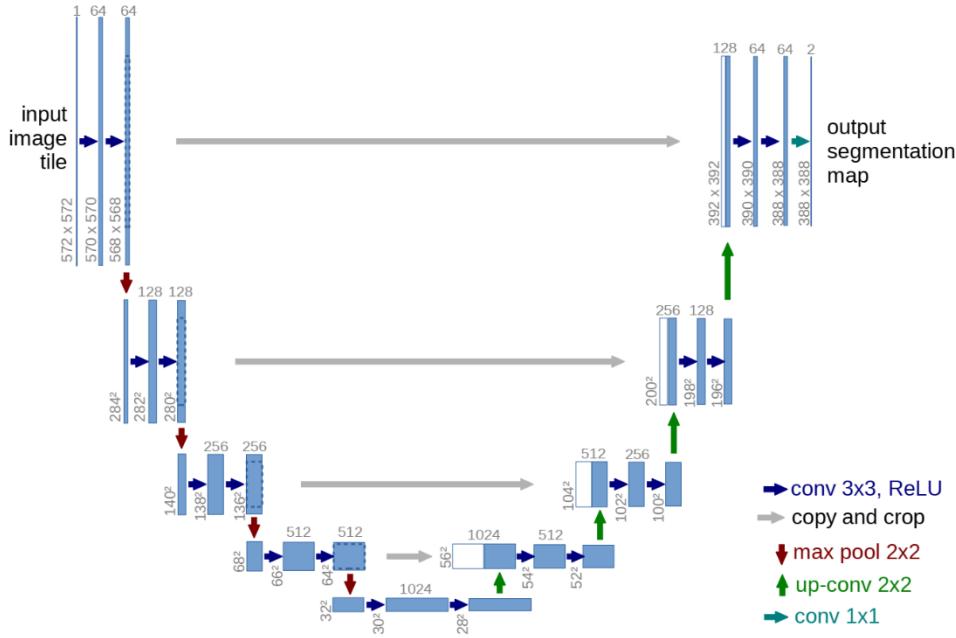
### 3.9. U-Net

The U-Net architecture is a variant of the FCN that is commonly used for semantic segmentation duties. Ronneberger et al. created the first U-Net architecture in 2015 to conduct picture segmentation and localization for biological applications [27]. U-Net is superior to traditional CNNs since they can give localization as well as classification in their output. In this scenario, localization implies labeling every single pixel from an image with a certain class. It is also more favorable than FCN because U-Net can function with fewer training photos while producing more exact segmentations. This is accomplished by constructing up-sampling layers with a huge amount of feature channels allowing context information to be propagated to layers having better resolution. Additionally, the segmentation networks of U-Net are the most appropriate for this application because of being the most easily scalable and sizeable FCN. Especially, they can keep away from a compromise between contextual information and localization, furthermore, they are frequently employed in cutting-edge semantic segmentation approaches [27].

#### 3.9.1. U-Net Architecture

The U-Net architecture is made up of a route of contraction at the input and a route of expansion at the output. Ronneburger's paper introduces the basic architecture of this network, the first route is the route of contraction, which is also known as the encoder path, being essentially a stack of CNN components such as convolution, activation,

and pooling layers, that are designed to take the context from the input image. The encoder output is produced tinier than the input, which is gradually expanded in the route of expansion. With transposed convolutions, the route of expansion has another name is decoder path which allows for exact localization. Through a series of up-convolutions and concatenation with appropriate feature maps belonging to the contracting path, the expansion pathway integrates high-level features with spatial information. The architecture is depicted in Figure 42 [27].



*Figure 42: U-Net Architecture (The blue boxes present for feature maps in multi-channel. The quantities of channels are indicated on the box's top. The size of x-y is specified on the box's lower left side. The white boxes represent feature maps that have been replicated. The arrows represent the various operations) [27].*

The encoder route of U-Net is a typical CNN design, including recurring 3x3 convolutions and operations of max pooling with stride 2 for down-sampling which double quantity of feature channels before

appearing of up-sampling the feature map and performing a 2x2 convolution on the expanding route. In detail, these actions remove the quantities of feature channels in a half before concatenating them with the feature map being cropped from the route of contraction, this concatenation of the U-Net is the different point with the skip connection of FCN. After that, an extra convolution layer is added to map every single feature vector to the expected number of classes [27]. The U-Net can collect image context by employing the contracting route, on the other side, the symmetric expanding route allows for exact localization. By integrating the characteristics with high resolution which come from the contracting path being up-sampled output, localization is activated. Added, the propagation of context information to the layers with higher resolution is allowed by the model's up-sampling section with a huge quantity of feature channels [27].

Based on the feature maps with low level from the encoder contain more spatial information which is helpful to the analysis of complicated scenes containing multiple objects and their relative configurations, there is a combination of the feature maps at intermediate low level from EfficientNet or ResNet and the feature maps at intermediate high level from U-Net decoder. The network is able to transfer information of context to layers supplying higher resolution due to the huge quantity of feature channels in the up-sampling section. Because of the symmetric characteristics in conventional U-Net. In this work, we have tried to use EfficientNets and ResNets as the encoders in the contraction route instead of a regular collection of convolution layers. The module of the decoder is like the primordial U-Net. The detailed implementation for the combination of these architectures will be mentioned in the following chapter named “implementation”.

## **Chapter 4: Methodology Approaches**

Given the stated requirement of defects detection in steel through images accompanied by information about defects based on EncodedPixels, these defects are represented as masks on each image with contour lines that are without any fixed shape (this will be discussed in detail in chapter 5), so this is an image segmentation problem. With the utilization of the basic CNN model for identification, and some state-of-the-art models like VGG and ResNet that can use for classification and localization of the object in the image. They may be not the main models that are suitable for this issue. There was a thought about using the U-Net model as the first approach to finding a solution. Because U-Net is one of the state-of-the-art models being developed based on CNN for the field of image recognition, specifically image segmentation.

### **4.1. The U-Net Model Approach**

In the first approach, there was an implementation for a solution relying on the original U-Net architecture that the author has developed. With the training data provided, there was a step of image processing that converts this dataset to tensor format to fit the model and divides this training data set into 80% for the training and 20% of the remaining for the validation, and the input image is normalized. The training process was utilized Adam as an optimizer, went through 200 epochs, and set EarlyStop callback function with the condition of monitoring the "val\_loss" and patience equal 15. However, with a total number of 600,756 model parameters can "learn" from this approach, the training model stopped after going through about 45 epochs and achieved the result of training dice coefficient value (training accuracy of the model)

of 0.77 and validation dice coefficient value (validation accuracy of the model) of 0.59. With the result of the binary cross-entropy loss function at 0.01 for training and 0.03 for validation. This is a relatively positive result for the initial steps to find a solution, but it is overfitting and not the most optimal result.



Figure 43: Dice coefficient on data of training and validation in U-Net model approach.

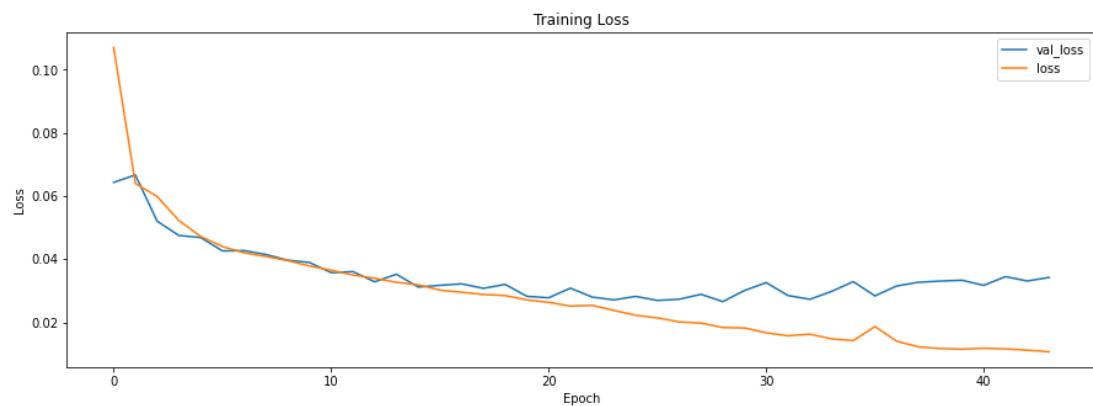


Figure 44: Loss on data of training and validation in U-Net model approach.

## 4.2. Combination of the ResNet and U-Net Model Approach

With the first approach of the U-Net model with only 600,756 parameters. There was a thought that the number of parameters was not enough and this model was not deep enough to learn, which led to an idea of a significant increment of the quantities of parameters the model could learn based on the methodology that the deeper can go, the better the model can learn. And ResNet is one of the state-of-the-art networks that prove it with residual blocks helping models avoid premature convergence and premature stop of learning. There has been an approach by combining Resnet and U-Net, specifically, this second approach is different from the first in using U-Net as the main architecture and different versions of Resnet as the backbone instead of CNNs. initial layers of the U-Net architecture, along with pre-trained weights of "imagenet" and freezing encoder part of the model during training. The number of trainable parameters of the model increased significantly, which were the backbone of ResNet-34 with 3,167,495, ResNet-50 with 9,059,079, and ResNet-101 with 9,111,303. After about 35 - 40 epochs that model was stopped by EarlyStop callback, the results were the training dice coefficient values reached about 0.80 – 0.83, and the validation dice coefficient values reached about 0.65 – 0.68, and the result binary cross-entropy loss function around 0.01 – 0.0079 for training and around 0.02 – 0.03 for validation. With this approach, the results have improved slightly, however, the results presented that the models were overfitting.

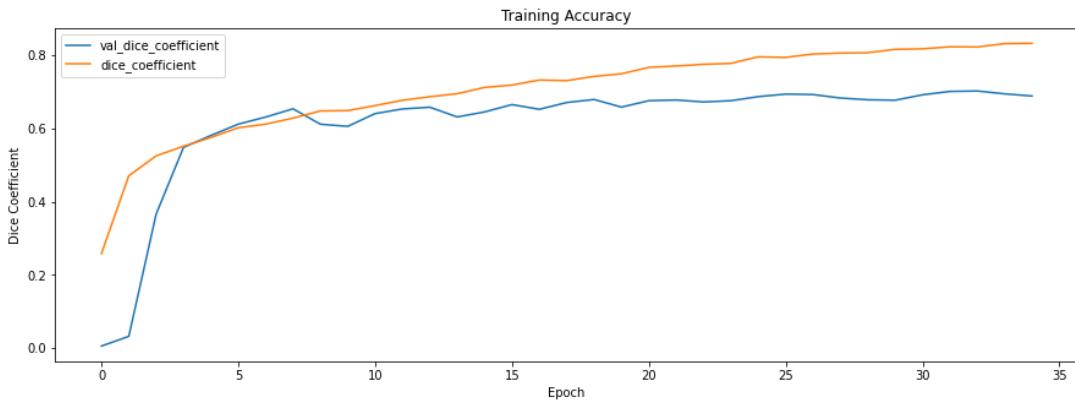


Figure 45: Dice coefficient on data of training and validation in ResNet34 - U-Net model approach.

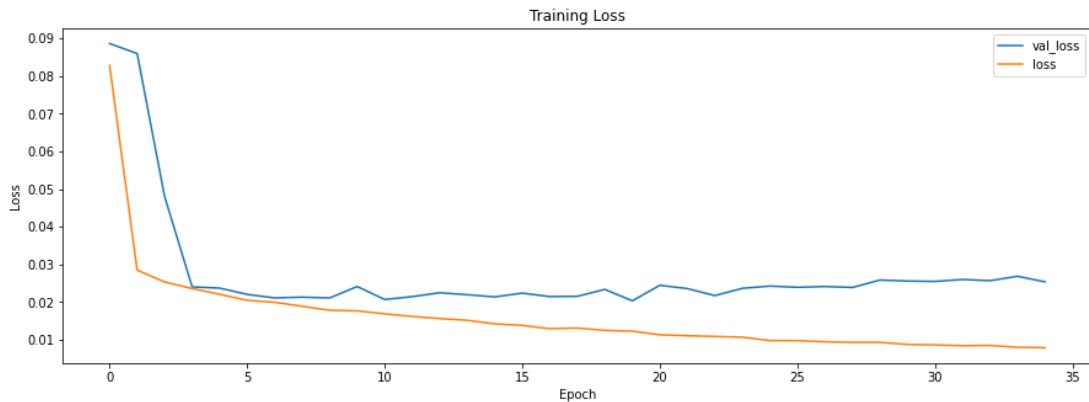


Figure 46: Loss on data of training and validation in ResNet34 - U-Net model approach.

### 4.3. Combination of the EfficientNet and U-Net Model Approach

Realized that the experiment with the combination of ResNet models with U-Net was not objective enough and the results were not satisfactory. There was a study and application of EfficientNet architecture as the backbone of the U-Net Model as an alternative. However, the achieved results only reached a slight improvement. Specifically, the implementation of the EfficientNet-B5 with the U-Net model reached 0.88 for the value of the dice-coefficient and 0.70 for validation one, the values of loss and validation loss got 0.0052 and 0.034 respectively.

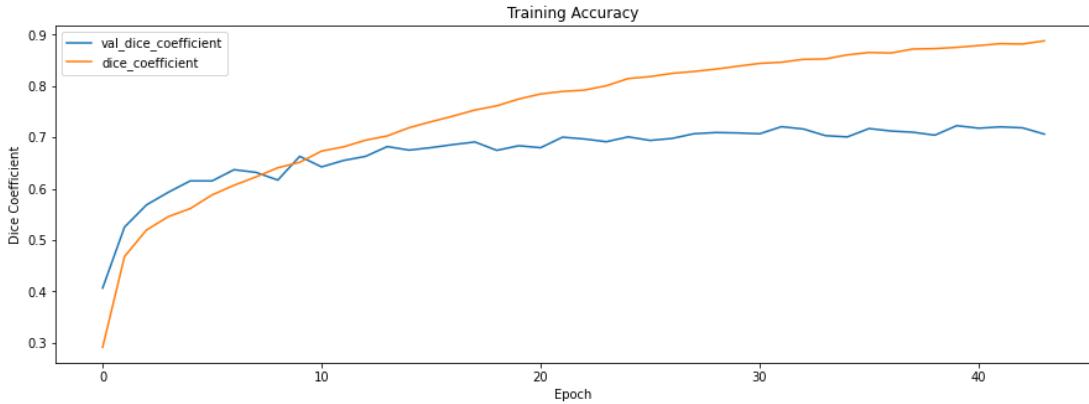


Figure 47: Dice coefficient on data of training and validation in EfficientNet-B5 - U-Net model approach.

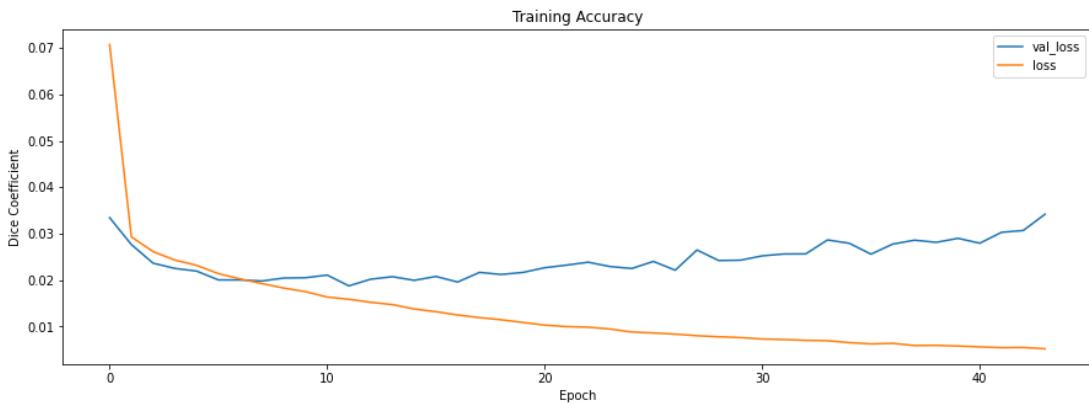


Figure 48: Loss on data of training and validation in EfficientNet-B5 - U-Net model approach.

#### 4.4. Data augmentation and K-Fold Approach

After the above approaches and their results, it became clear that a change in model usage was not the only way to improve the results. There has been a step using data augmentation to increase the number of defects with few records to balance the amount of data in the defect type. Moreover, using the K-fold method during training to avoid using the same data set in the whole process as well as evenly dividing the data of defect types in the set of both training and validation. Then conducted model training with model combinations between ResNets and U-Net,

and EfficientNets and U-Net with settings of 150 epochs for each Fold, respectively. This process stops after 5 folds with about 100 epochs and gives very good results, specifically, the values of dice-coefficient in both training and validation reached about 0.90 to 0.94, and their values of loss function accounted for the range of 0.0017 - 0.0031. This is the best-achieved result in all mentioned approaches. This approach will be presented in more detail in chapter 5 as the main implementation for the thesis and the achievements in chapter 6 as the results.

#### 4.5. Comparison among the approaches

After all mentioned approaches, using the U-Net model as the main model is a fundamental step in finding a solution of image segmentation, however, given the number of parameters provided by CNNs. layers in the U-Net model are not enough to achieve an impressive result. In addition, the ResNet and the EfficientNet are architectures that make the model capable of performing deeper learning. With the difference that ResNet architecture is built by many layers of residual blocks, while EfficientNet is built by model scaling method which not only adds more layers but also expands the number of channels in layers and layer's resolutions. Therefore, using separately each architecture of these as the backbone of the U-Net model with their pre-trained weights, helped improve slightly the approximately same results for the problem of this thesis. Moreover, due to the imbalance of defect types in the data, it has a great influence in achieving results. Therefore, balancing the data and using the K-fold technique is also a very important step for helping the model achieve the most optimal results.

## Chapter 5: Implementation

### 5.1. Dataset – Image Processing

As mentioned in chapter 1 in this thesis, the data set provided by Severstal was used, which is a dataset containing images of steel defects, specifically, there are 4 types of defects, defect type 1 is shaped like marks of bumps with screw or angled hard object, defect type 2 is shaped like surface stains, defect type 3 is shaped like scratches on the surface, and finally defect type 4 is shaped like scabrous cracks.

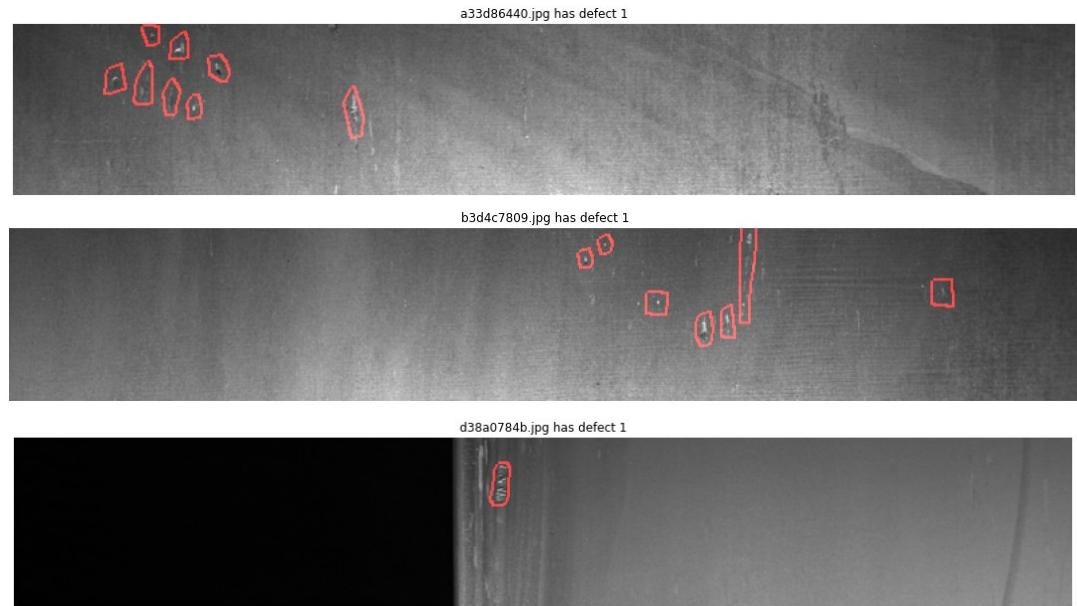
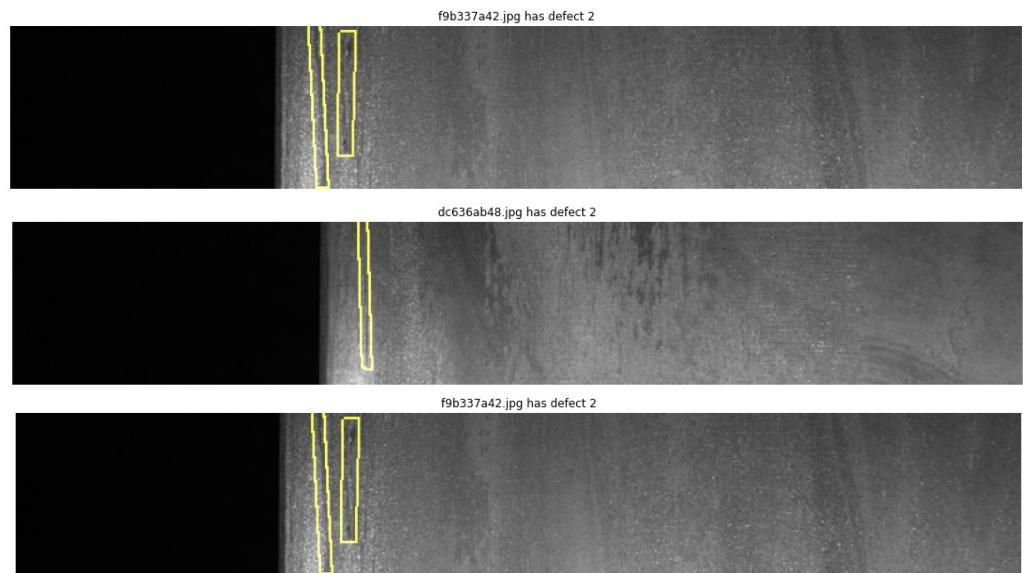
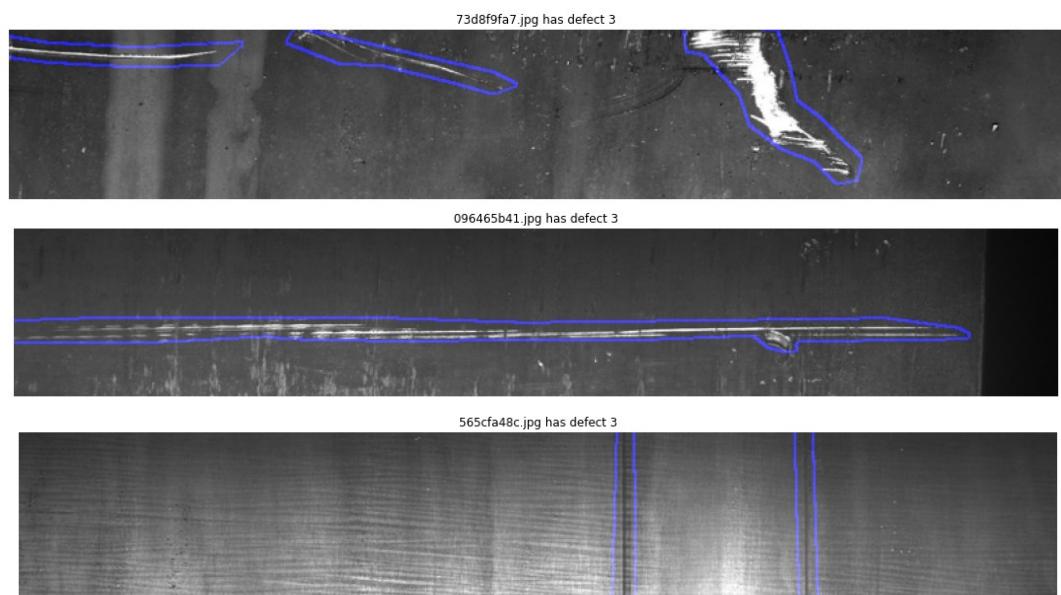


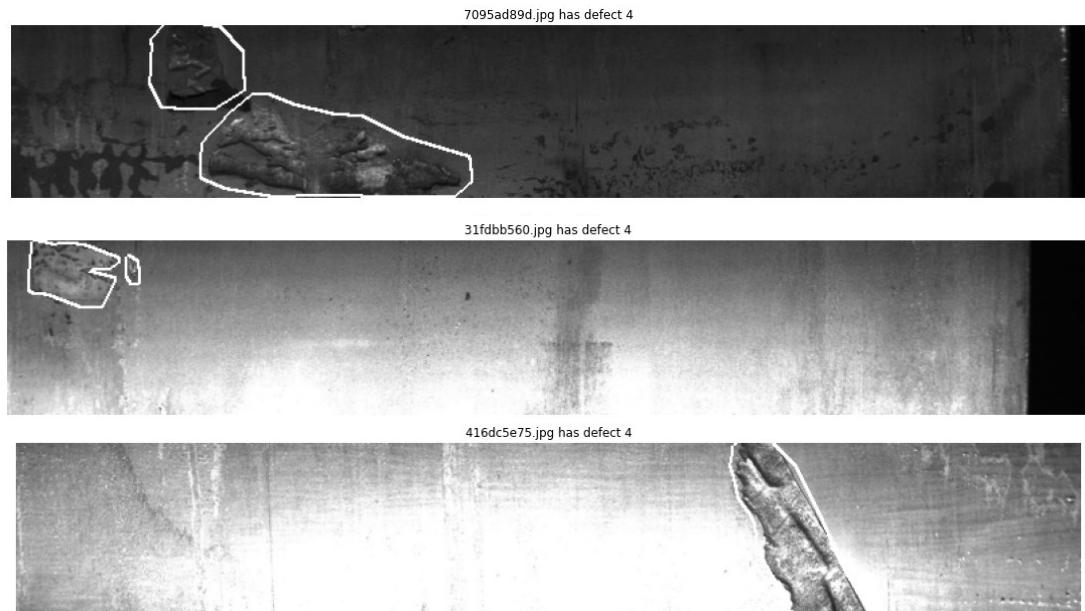
Figure 49: Defect type 1 with red border



*Figure 50: Defect type 2 with yellow border*



*Figure 51: Defect type 3 with blue border*

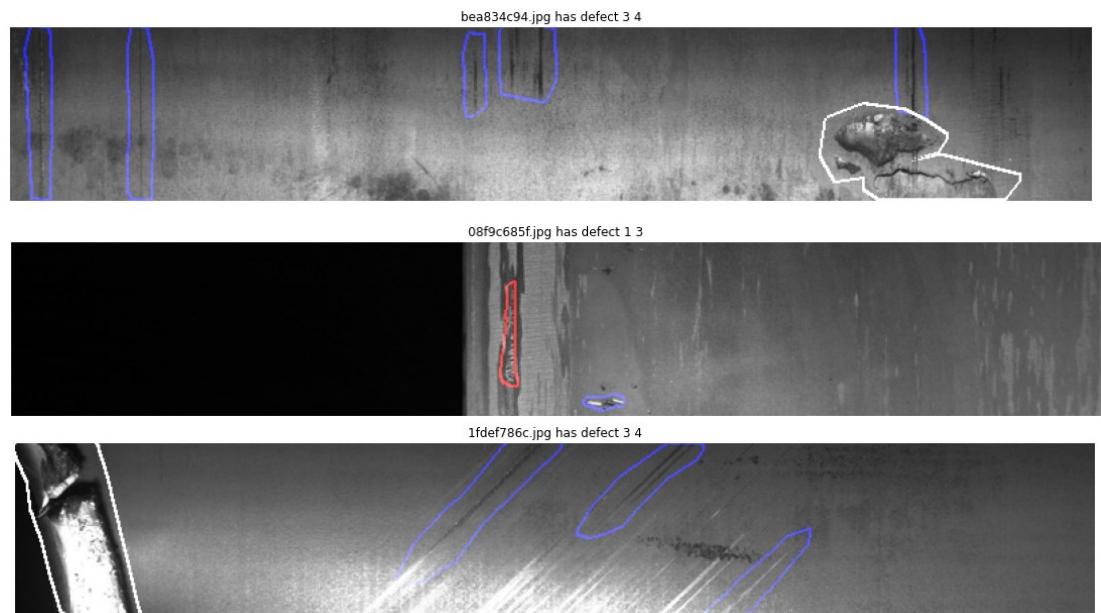


*Figure 52: Defect type 4 with white border*

The set of the data is divided into 2 main parts including images for the train set and test set, and csv files containing content about pre-classification of defects on each image. Specifically, the two image folders of the train set and test set are RGB images with 1600 x 256 pixels in size, they are both photos of manufactured steel and the content of the defects on them is not the same, most of them contain 1 type of defect, and a small part contains 2 types of defects. The total image quantities in the data set are more than 12.5k records, while the images in the test set are more than 5.5k records. Besides, the csv file is a file that summarizes the information of the image data file, more specifically, in the data set provided by the company, there are 2 csv files, the train.csv file carries the information of the train set including columns of information about file name (ImageId), type of defect (ClassId), pixels on the image are defects (EncodedPixels), while the remaining csv file has the same structure but no information about EncodedPixels and the ClassId in the test set from the provider. Therefore, the requirement here is to create a model that relied on the train set for the purpose of

detecting defects on the test set without given ground-truth masks to verify.

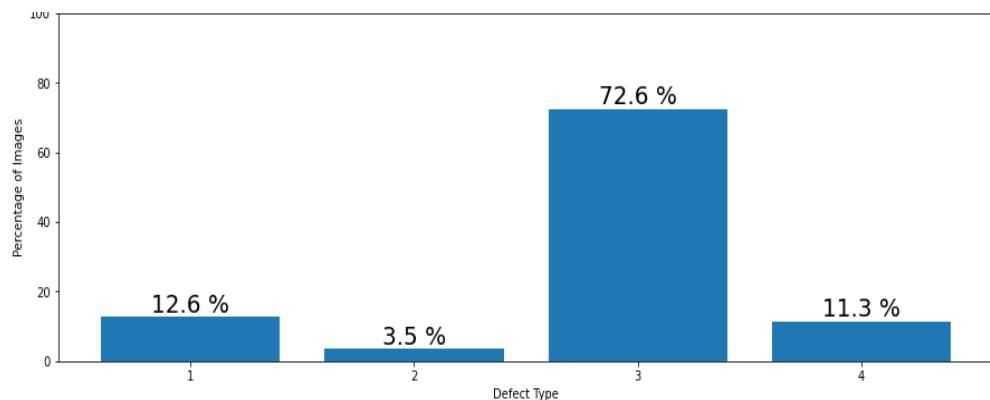
The RGB image is essentially formed by 3 matrices stacked on top of each other in layers or channels, and the values in each matrix represent each pixel on the image, and each value has a specific index certainly. In this dataset, each defect is represented by each pixel on each image. It makes a huge size of data when storing all of the information of defect pixels on each image by matrix or image format. To resolve this issue, the provider utilized EncodedPixels as an encoder method that minimizes the amount of information and memory size of pixel information containing defects in an image. The encoded defect information based on starting position of pixels and run-length based on these matrices, which has the format [pixel index] [run length] e.g., 10 25 100 200, it means there are 2 defects on this image, which runs from 10th pixel to 35th pixel (10 + 25) and the second one is from 100th to 300th (100 + 200).



*Figure 53: Image with multiple defect types*

### 5.1.1. Data Augmentation

Train set with more than 12.5k records, but in reality, the number of defects is not balanced and the number of records with defects in the csv file is just over 6.6k. In this data set, the number of type 3 defects accounts for the majority compared to other types of defects, and type 2 defects account for the smallest number in the data set, specifically, in total counting of both single and multiple defects on the image, type 1 defects account for 12.6% with 897 records, type 2 accounted for 3.5% with 247 records, type 3 accounted for 72.6% of 5150 records, and type 4 accounted for 11.3% of 801 records. That is reflected in Figure 54.



*Figure 54: Percentage of defect types include in train data*

For that reason, it has been utilized data augmentation method to equilibrate the classes. When labels in the data for the training model are not balanced, they will have a great influence on the accuracy of the model, in this case, the model will detect and predict defect type 3 very well, and poorly the rest of the defects (especially the type 2 since it has the lowest number of samples). Therefore, balancing the data is a very important procedure to improve the quality of the model.

In detail, with a smaller number in categories 1, 2, 4, there is a utilization of the method of rotating 180 degrees, 10 degrees and flipping the image to increase the number of records of these types, ending up

getting a result number of defects appearing in the train set at equilibrium with 1291 records in both single and multiple defects appearing in the image.

Before implementing the augmentation method, 25 records for each single defect type and 25 image records containing multiple defect types have been retrieved for the test set, for a total of 250 image records. The reason is that the test set provided by the supplier does not have ground-truth masks data, resulting in no data source to re-evaluate the model with the test set after conducting the training. Therefore, with these 250 records with ground-truth masks, it is easy to re-evaluate the model after training.

### 5.1.2. K-fold cross-validation

With the train set balanced after the augmentation method, the K-fold strategy is applied to them during the training model to increase the efficiency of the model. This strategy divides the supplied set of training data into  $k$  disjointed subsets with a roughly similar size. The resulting subsets number is denoted by the term "fold", which this partition is accomplished by picking randomly examples from the training set without replacing them. Training of the model utilizes  $k - 1$  subset that plays a role in the training set as a whole. The model will be then applied to the other subset remaining as the set for validation, and its performance is evaluated [30].

This approach is iterated until all  $k$  subsets have been executed as validating sets. The cross-validated performance is the mean value of the  $k$  measurements of performance across the  $k$  validating sets. This process is depicted in Figure 49 with  $k = 10$ , i.e., 10 - fold cross-validation. Firstly, the earliest subset in the first fold acts as the validation set  $D_{val,1}$ ,

while the following 9 subsets execute as the training set  $D_{train,1}$ . Secondly, the validation set is the second subset in the second fold, while the other subsets will be the set of training, and so on [30].

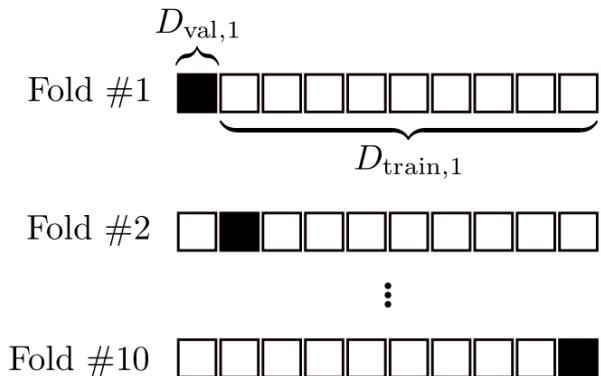


Figure 55:  $K$  – Fold Cross-Validation [30].

### 5.1.3. Data Generation

With a certain amount of data for training and validation work, there is a creation of a data generator class to match and optimize the hardware in the training model, which has function generating an output with processed image and ground-truth masks. Specifically, the processed image, which is a reduction of the input image size by half from 1600 x 256 pixel to 800 x 128 pixel, dividing the number of the input images into 16 batch sizes for each training epoch, moreover, the image has been normalized from RGB format with range 0 - 255 to grayscale format with the range of 0 – 1 that helps model easy to decide to be a pixel of defect or not. Then the masks are 4 matrices with size 800 x 128 equivalent to the image, each matrix represents a defect type contained in the image, in which the pixel is defined as defect having the value 1 and other ones without defect having a value of 0. These matrices are generated by decoding information from the EncodedPixels provided by the *Severstal Corporation*.

## 5.2. Model

Based on the achievements of state-of-the-art networks such as ResNet, EfficientNet, and U-Net. There have been two separate combinations: the combination of ResNet and U-Net, and the combination of EfficientNet and U-Net, where U-Net was the architecture acting as the main framework of the model, and Resnet and EfficientNet played a role as the backbone of the model, they replaced the original encoder part belonging to the U-Net architecture, with this combination there were models with a huge quantity of parameters created, and with skip connections in the ResNet Blocks and compound scaling blocks in EfficientNet made it possible to train the model more deeply than the original one, which gave the model a very significant improvement.

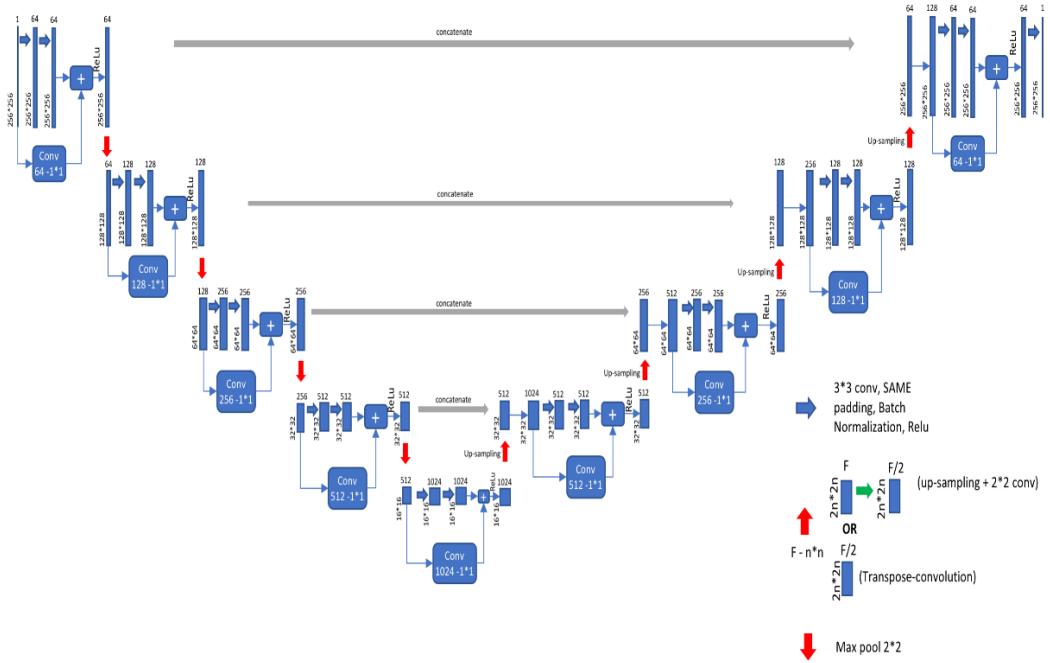


Figure 56: U-Net with ResNet as the backbone [34].

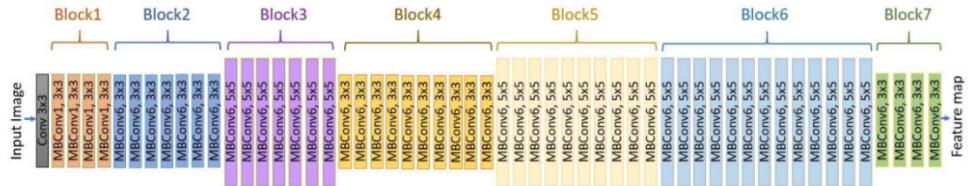


Figure 57: EfficientNet-B7 architecture [31].

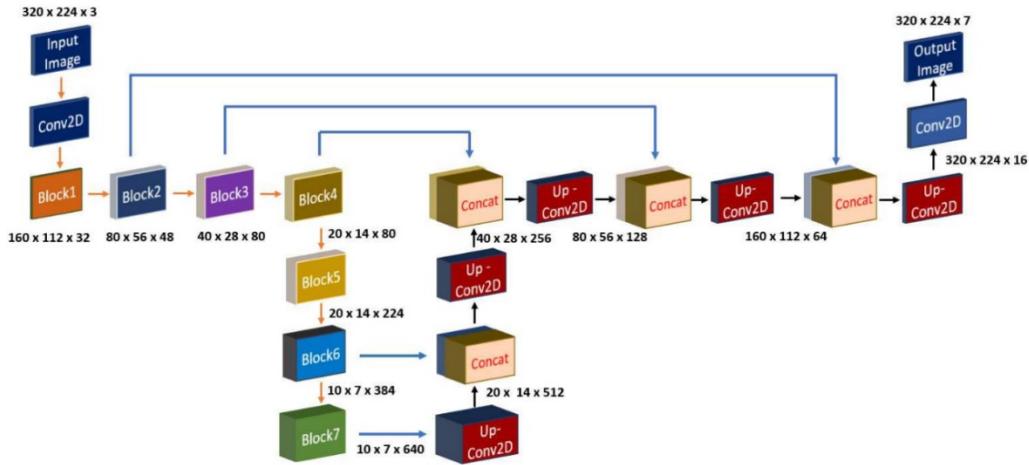


Figure 58: U-Net with EfficientNet-B7 as backbone [31].

### 5.2.1. Evaluation Metrics

As the evaluation matrix for this thesis, there is a utilization of the mean Dice coefficient to make a comparison about the pixel-wise agreement between a prediction of segmentation and its matching ground-truth mask. The dice coefficient is calculated by counting the similar pixels (taking intersections that appear in both images) in the two images being compared, multiplying it by 2, and dividing it by the total pixels of both images. The formula is shown in Figure 59:

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

Figure 59: Dice – Coefficient [36].

where X presents for the set of predicted pixels and Y stands for the ground-truth mask.

### 5.2.2. Loss function

To detect which pixels are defective or not on the input image, there is a utilization of the loss function of binary cross-entropy for this thesis, with this loss function operating on the returned real numeric values in the range 0 - 1, with values  $\geq 0.5$  will be considered as defects, and vice versa values  $< 0.5$  will be considered as background.

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

*Figure 60: Binary cross entropy [35].*

where  $y$  presents for the label (1 stands for pixels of the defect and 0 stands for pixels of background) and  $p(y)$  stands for the probability of the predicted pixel that is a defect for all  $N$  pixels [35]. The formula expresses that for each pixel of the defect ( $y = 1$ ), it is multiplied to  $\log(p(y))$  to the loss, which is, the log probability of its defect. Nevertheless, it will be multiplied to  $\log(1 - p(y))$ , which is the log probability of it being background, for every pixel of background ( $y = 0$ ) [35].

### 5.2.3. Execution Environment

The thesis is used the Colab Pro platform of Google, it is an application of cloud computing running the Linux operating system, which provides a very useful tool with code cells and the run-time of python with version 3.7, quite similar to Anaconda notebook running on a local machine. The platform offers quite a powerful hardware for ML as well as DL research, with GPU of NVIDIA TESLA P100 16GB, 27.3 GB for memory, and 168 GB for the local storage. This specification of hardware helps save training time and supply enough space for storing the data in research.

#### 5.2.4. Model Training Process

Implementing this idea, there have been some experiments that have been done on several versions of the combination of ResNet and U-Net, and some versions of the combination of EfficientNet and U-Net. And there has been a utilization of transfer learning for all of the above architectures based on "Segmentation Models" library version 1.0.1, specifically with ResNet, having usages of ResNet-34, ResNet-50, and ResNet-101 as the backbone. On the opposite side, with EfficientNet, having usages of instances B0, B1, B3, B5, and B7 as the backbone of the model.

All versions of the model use the same loss function of binary cross-entropy and dice-coefficient as an evaluation matrix. Since the requirement is to identify 4 different types of defects, the number of classes needs to be segmented accordingly with the total number of defect types. The output layer of the model has a shape of 128 x 800 x 4 corresponding to the height, width, and number of defects, which is applied to the sigmoid function as the AF. Furthermore, to help this model quickly converge, having usage of pre-trained weights of "imagenet" corresponding to each type of backbone used, and the weights of this model encoder are frozen during training. Therefore, a huge number of parameters in the encoder part cannot be trained and only the remaining parameters in the decoder part are trained. Adding, the optimization method utilized for the model is Adam.

In the training process, there was a usage of the K - fold method, with K = 5 to create a training and validation set with a ratio of 80 - 20 respectively in the total training data. The models were trained this bunch of data in 5 folds, each fold made duty on running through 150

epochs, however, there was an implementation of EarlyStopping callback with monitoring the change of "val\_loss" value in the range of 15 for patience value with the mode is "min" aims to stop the training work when the model has reached the convergence point and there is no possibility of further improvement. With this selection of implementation, regular models are trained through a total of 90 - 120 epochs to reach convergence and the EarlyStopping callback is executed.

And these implementations all yield positive and slightly different results.

## Chapter 6: Results

After several experiments training the models, there were relatively positive results being obtained. Specifically, after training for about 90 - 120 epochs out of a total of 5 folds, there was an evaluation for the model with validation and test sets being executed. In general, with the implementations of Resnets and EfficientNets as the backbone for the model along with the k-fold cross-validation strategy during training, the average train loss and validation loss values of binary cross-entropy accounted for the range of 0.0017 - 0.0031, and the average values of the evaluation matrix of dice coefficient in training and validation accounted from 0.904 to 0.941. It is shown in Figure 61.

U-Net Model with backbone	Time Execution (minutes)	Average Train Loss in 5 - folds	Average Val Loss in 5 - folds	Average Train Dice Coefficient in 5 - folds	Average Val Dice Coefficient in 5 - folds	Test loss	Test Dice Coefficient
ResNet-34	<b>104.75</b>	<b>0.00311</b>	<b>0.00311</b>	<b>0.904</b>	<b>0.904</b>	<b>0.033</b>	0.692
ResNet-50	147.91	0.00194	0.00194	0.938	0.938	0.038	0.701
ResNet-101	182.82	0.00252	0.0025	0.926	0.928	0.036	0.701
EfficientNet-B0	117.58	0.00254	0.00254	0.927	0.926	<b>0.047</b>	<b>0.668</b>
EfficientNet-B1	126.39	0.0023	0.0023	0.927	0.927	0.039	0.696
EfficientNet-B3	162.6	0.0022	0.0023	0.924	0.927	0.034	0.714
EfficientNet-B5	255.26	<b>0.0017</b>	<b>0.0018</b>	<b>0.941</b>	<b>0.94</b>	0.045	0.711
EfficientNet-B7	<b>389.69</b>	0.0023	0.0024	0.926	0.925	0.034	<b>0.714</b>

Figure 61: Result of training models with evaluation values.

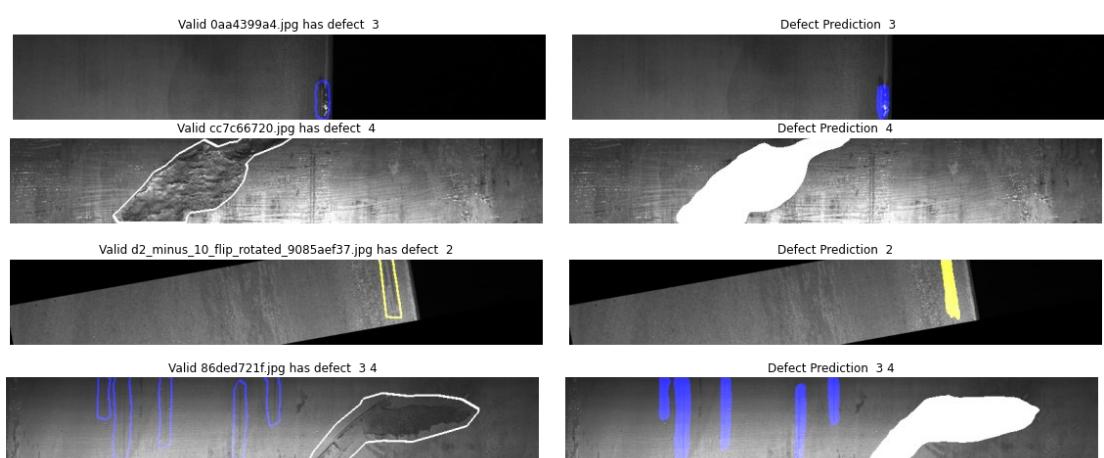
U-Net Model with backbone	Total Parameters	Trainable Parameters
ResNet-34	24,456,589	<b>3,167,495</b>
ResNet-50	32,561,549	9,059,079
ResNet-101	51,605,901	9,111,303
EfficientNet-B0	<b>10,115,936</b>	6,106,404
EfficientNet-B1	12,641,604	6,126,436
EfficientNet-B3	17,868,268	7,170,052
EfficientNet-B5	37,469,108	9,126,340
EfficientNet-B7	<b>75,048,532</b>	<b>11,259,588</b>

Figure 62: Model Parameters.

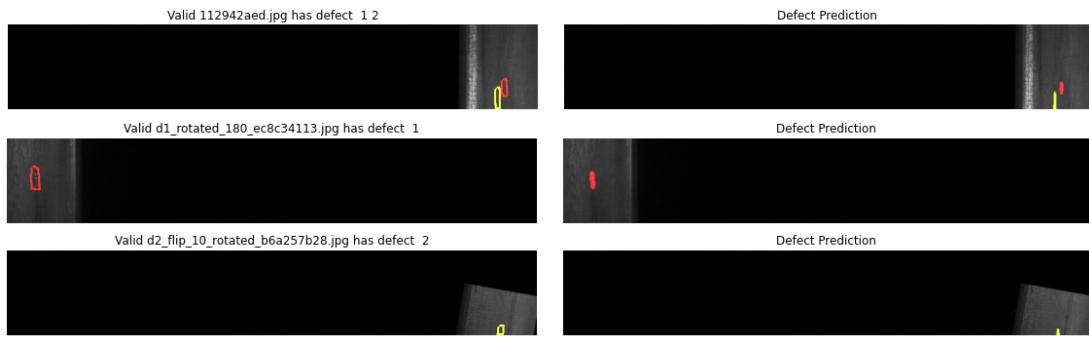
Based on Figure 61 we can see the combination of ResNet-34 and U-Net gives the least efficient results, with results of the average loss of data set of training, validating, and testing were 0.00311, 0.00311, and 0.033 respectively. Moreover, the average training and validating of dice coefficient had a value of 0.904, while the test dice coefficient came in at 0.692, which reached almost the bottom of the results. In contrast, the model with the EfficientNet-B5 backbone had the best achievements, with the average loss of the set of training data and the set of validating data, and loss of the set of testing data values of 0.0017, 0.0018, and 0.045, respectively. Then, the accuracy of the dice coefficient in both training and validation accounted for 0.941 and 0.94, the test dice coefficient value reached 0.711 being almost the highest result value. Looking at the results of the test dataset with 250 records, the model using EfficientNet-B0 gave the worst result with 0.047 for the test loss and 0.668 for the test dice coefficient. Oppositely, the model with the EfficientNet-B7 backbone gave the best and most optimal results for the loss test and dice coefficient test, which accounted for values of 0.034 and 0.714 respectively.

Furthermore, when looking at both Figure 61 and Figure 62, we see that the number of trainable parameters between the models was not the same, with the models with the fewest number of trainable parameters giving the lowest results and the least time cost for training, specifically, ResNet-34 backbone gave us more than 3M trainable parameters and only took more than 104 minutes to perform the training task. Whereas the EfficientNet-B7 backbone was the choice that consumes the most parameters and execution time and also brought the best results for the test set, with more than 11M trainable parameters it took more than 389 minutes to complete training the model.

Evaluation of the performance of the classification model is the next step, firstly, the model prediction step resulted in prediction masks, specifically, the output of prediction contains 4 matrices with the size 128 x 800 representing 4 types of defects that could be shown relied on its values, because of applying sigmoid function for output layer of the model, the values inside each matrix will be in the range from 0 to 1, the value is equal or bigger than 0.5 presenting an image pixel of the defined defect type corresponding the order of matrix and the value is less than 0.5 presenting for an image pixel of background that is not a defect. Secondly, based on that result, there was a step to calculate the total number of defect pixels on each matrix for defect classification on the image, specifically, the matrix is respective to a specific defect type, which has a total number of pixel defects equal or greater than 200 in the total number of ( $128 \times 800 = 102,400$ ) pixels on an image, being determined as the respective defect type existing on the predicted input image, besides, the matrix contained a total number of defect pixels less than 200 being known as the respective type of defect not existing on the predicted image.



*Figure 63: Predictions with masks that contain equal to or more than 200 defect pixels on each defect type with the respective label.*



*Figure 64: Predictions with masks that contain less than 200 defect pixels on each defect type without being labeled.*

Thirdly, processed the step of label encoding by creating a list that relied on previous results for the calculation of the accuracy of classification work. The matrices that were known as the defect types existing on the image would be set as the value of 1 and other background matrices would be the value of 0, these values would be collected as an item. Then we aggregated these items in a list of label encoding. Finally, it was to utilize both "accuracy\_score" function and "multilabel\_confusion\_matrix" function which are provided by the "Sklearn" library. With the "accuracy\_score" function, we performed the label encoding for the image's ground-truth masks from the test set as a value of the "y\_true" parameter and utilized the list of label encoding from predicted results in previous work as a value of the "y\_pred" parameter, then we performed a general evaluation of accuracy on the test set with non-single defect types, which is supposed to be how many defect types appear on an image, used its results to calculate an average result of accuracy score. And then it was an evaluation step for each defect type separately. After that, there was the use of the "multilabel\_confusion\_matrix" function to give the number of matrices corresponding to the number of defect types to be predicted, here with 2

parameters "y\_pred" and "y\_true" used in this function returned the result as 4 matrices. Each matrix encloses the number of records of false negative (FN), true negative (TN), false positive (FP), and true positive (TP) results on the total quantity of records of the set of test data.

U-Net Model with backbone	Accuracy classification score for total defects (250 records)	Accuracy classification score in each defect (250 * 4 = 1000 records)	Confusion Matrix in defect type 1 [TN FP FN TP] [Accuracy]	Confusion Matrix in defect type 2 [TN FP FN TP] [Accuracy]	Confusion Matrix in defect type 3 [TN FP FN TP] [Accuracy]	Confusion Matrix in defect type 4 [TN FP FN TP] [Accuracy]
ResNet-34	<b>0.7</b>	<b>0.915</b>	167 15 12 56 (0.892)	189 1 13 47 (0.944)	129 28 13 80 (0.836)	168 2 1 79 (0.988)
ResNet-50	0.764	0.933	172 10 12 56 (0.912)	190 0 14 46 (0.944)	139 18 11 82 (0.884)	169 1 12 79 (0.992)
ResNet-101	0.756	0.927	173 9 11 57 (0.92)	187 3 13 47 (0.936)	138 19 15 78 (0.864)	168 2 1 79 (0.988)
EfficientNet-B0	0.748	0.928	173 9 17 51 (0.896)	189 1 9 51 (0.96)	139 18 15 78 (0.868)	168 2 1 79 (0.988)
EfficientNet-B1	0.772	0.933	177 5 15 53 (0.92)	188 2 8 52 (0.96)	141 16 17 76 (0.832)	167 3 1 79 (0.984)
EfficientNet-B3	0.804	0.943	178 4 15 53 (0.924)	188 2 8 52 (0.96)	140 17 10 83 (0.892)	169 1 0 80 (0.996)
EfficientNet-B5	<b>0.82</b>	<b>0.949</b>	177 5 13 55 (0.928)	189 1 7 53 (0.968)	144 13 10 83 (0.908)	169 1 1 79 (0.992)
EfficientNet-B7	0.804	0.943	178 4 15 53 (0.924)	188 2 8 52 (0.96)	140 17 10 83 (0.892)	169 1 0 80 (0.996)

*Figure 65: Accuracy Scores and Confusion Matrix Result of Training Models.*

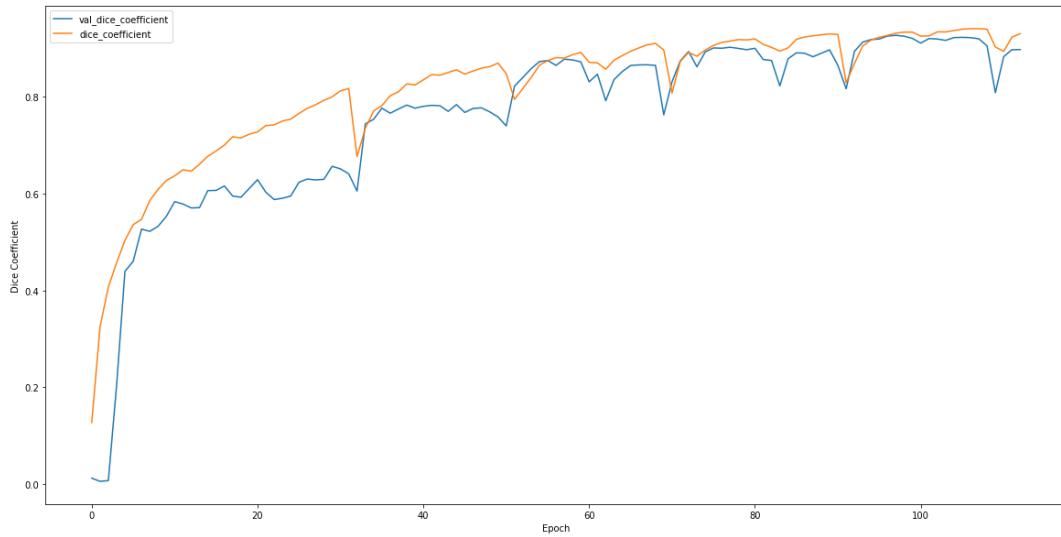
Looking at the Figure 65, the model using ResNet-34 as the backbone has the lowest accuracy score for the total number of defect types as well as for each defect type, with results of 0.7 and 0.915 respectively. Meanwhile, the model using EfficientNet-B5 as backbone

has the best accuracy score with 0.82 for total defect types, and 0.949 for each defect type. In addition to these 2 options, we have 2 other options with the same very good results, these are EfficientNet-B3 and EfficientNet-B7 as the backbone with 2 pairs of results being 0.804 - 0.943 for accuracy score of total defect types and each defect type.

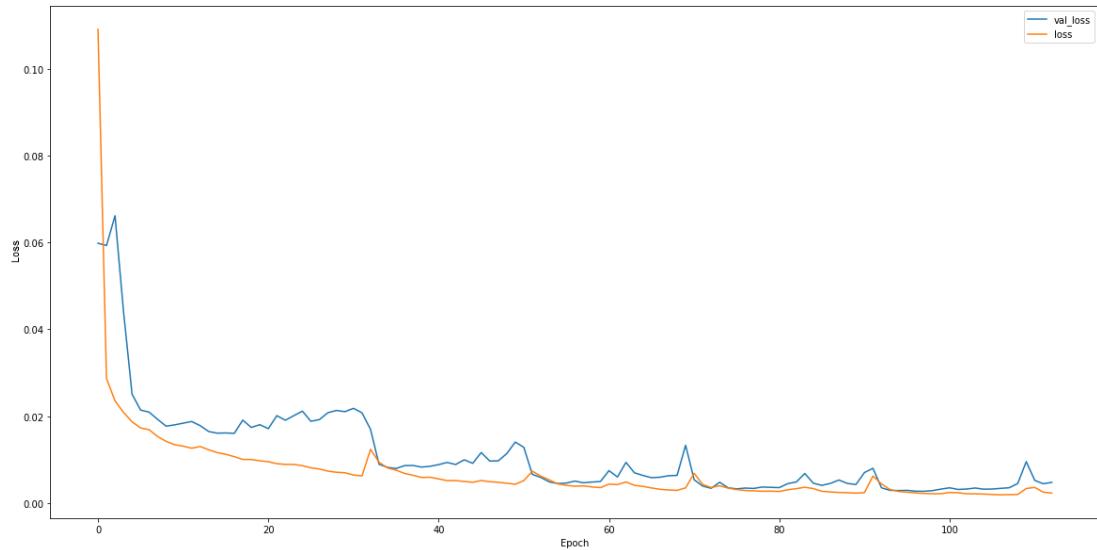
Subsequently, the results of using confusion matrix for model evaluation, overall, all the models have a pretty good result above 0.83. Specifically, there was the lowest result in defect type 3 with the values ranging between 0.832 - 0.908. However, the prediction of defect types 1, 2, and 4 yielded more impressive results, with types 1 and type 2 giving fluctuating results in the range of 0.89 - 0.96, in particular, type 4 gave excellent results at approximately above 0.98.

Based on the above results, we can conclude that the combination of ResNet-34 and U-Net gave the lowest optimal result, on the contrary, using EfficientNet-B5 and U-Net gave the most optimal results with the highest accuracy values in these versions and a reasonable execution time of 255.26 minutes for training the model. Although the use of EfficientNet-B7 got a slightly better result, we had to trade off the execution time of up to 389.69 minutes for the training model.

Figures 66, 67 show the results of dice coefficient and loss in both training and validation data set in the combination of ResNet-101 and U-Net model.

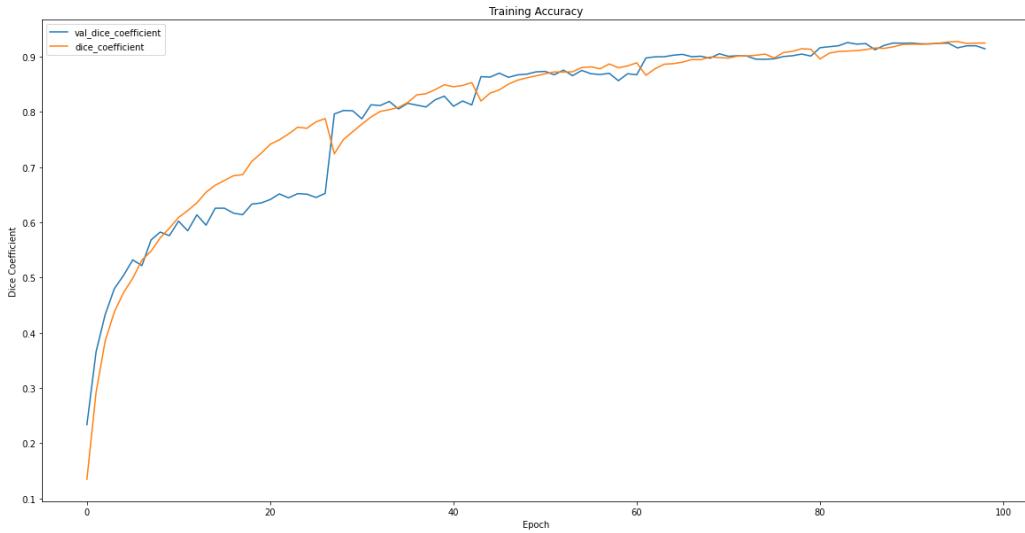


*Figure 66: Dice coefficient on data of training and validating in ResNet101 - U-Net model with 5 – Folds.*

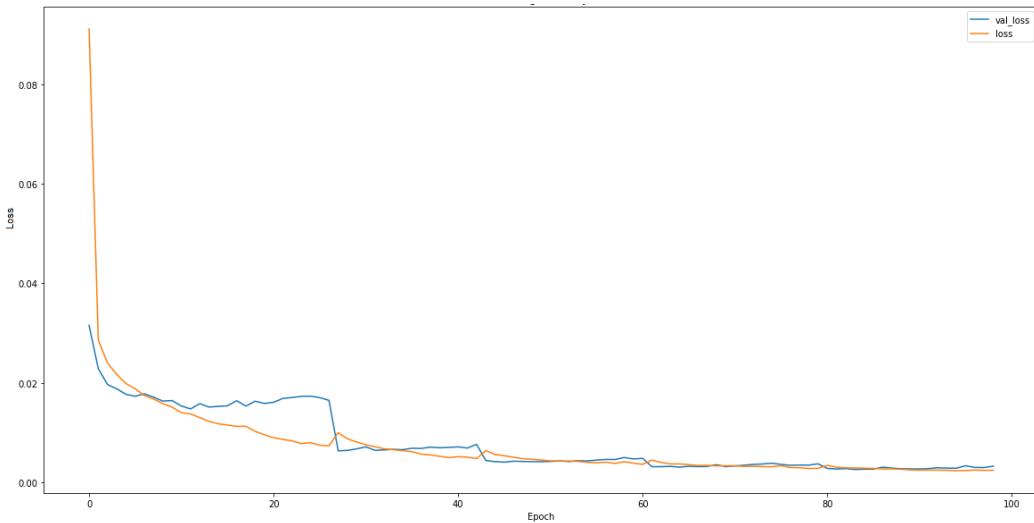


*Figure 67: Loss on data of training and validating in ResNet101 - U-Net model with 5 – Folds.*

Figures 68, 69 show the results of dice coefficient and loss in both training and validation data set in the combination of EfficientNet-B5 and U-Net model.



*Figure 68: Dice coefficient on data of training and validation in EfficientNet-B5 - U-Net model with 5 – Folds.*



*Figure 69: Loss on data of training and validation in EfficientNet-B5 - U-Net model with 5 – Folds.*

Figures 70 and 71 show the results of image prediction in validation and test data set in the combination of ResNet-101 and U-Net model and Figures 72 and 73 show the results of image prediction in validation and test data set in the combination of EfficientNet-B5 and U-Net model. The predicted label relied on the quantities of pixels that the model marked as the defect, the defect type would be supposed a defect if there are more than 200 pixels of being predicted defect.

## RESULTS

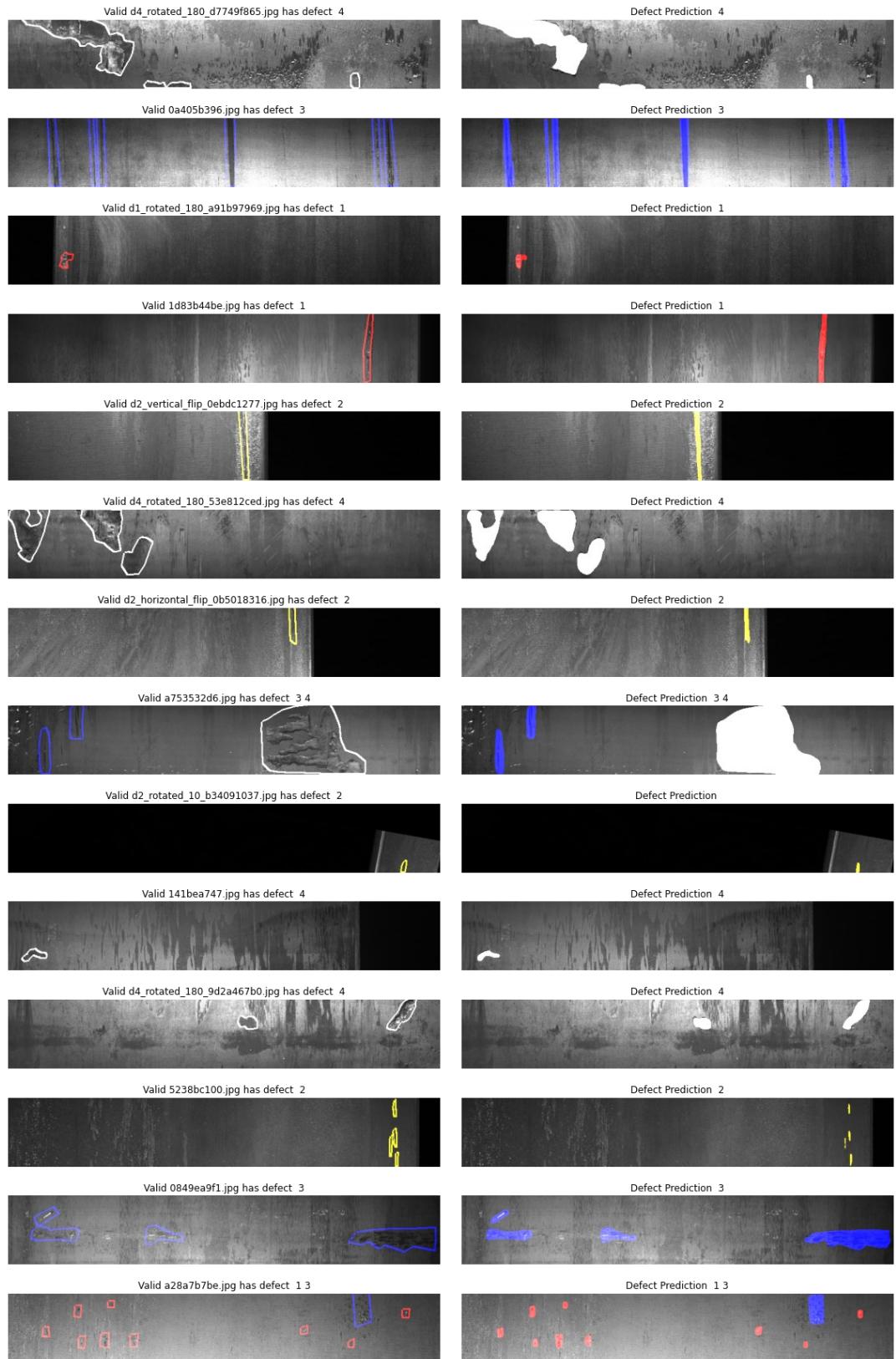


Figure 70: Prediction result on validation data in ResNet-101 – U-Net model with ground-truth compared to predicted masks.

## RESULTS

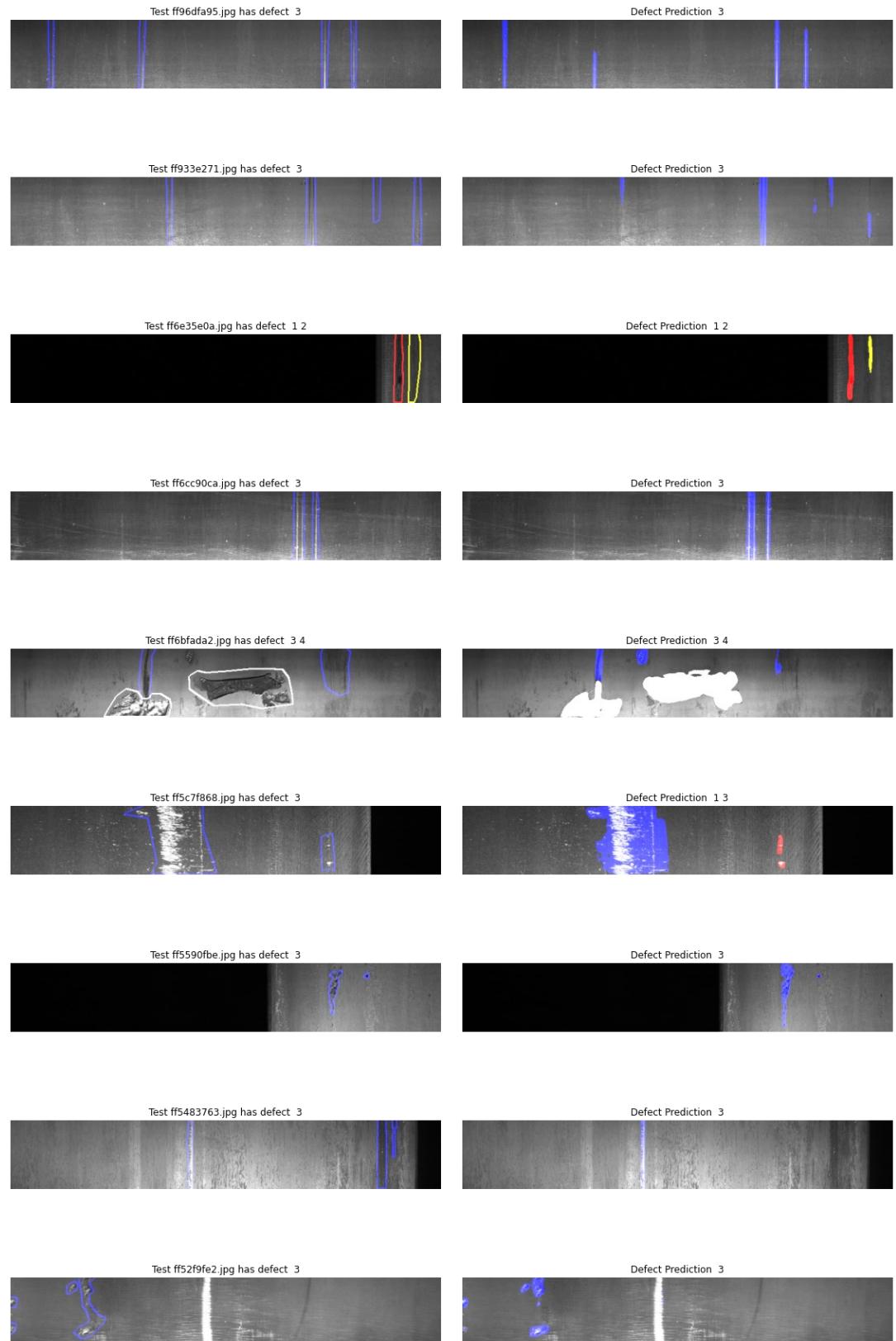


Figure 71: Prediction result on test set in ResNet-101 – U-Net model with ground-truth compared to predicted masks.

## RESULTS

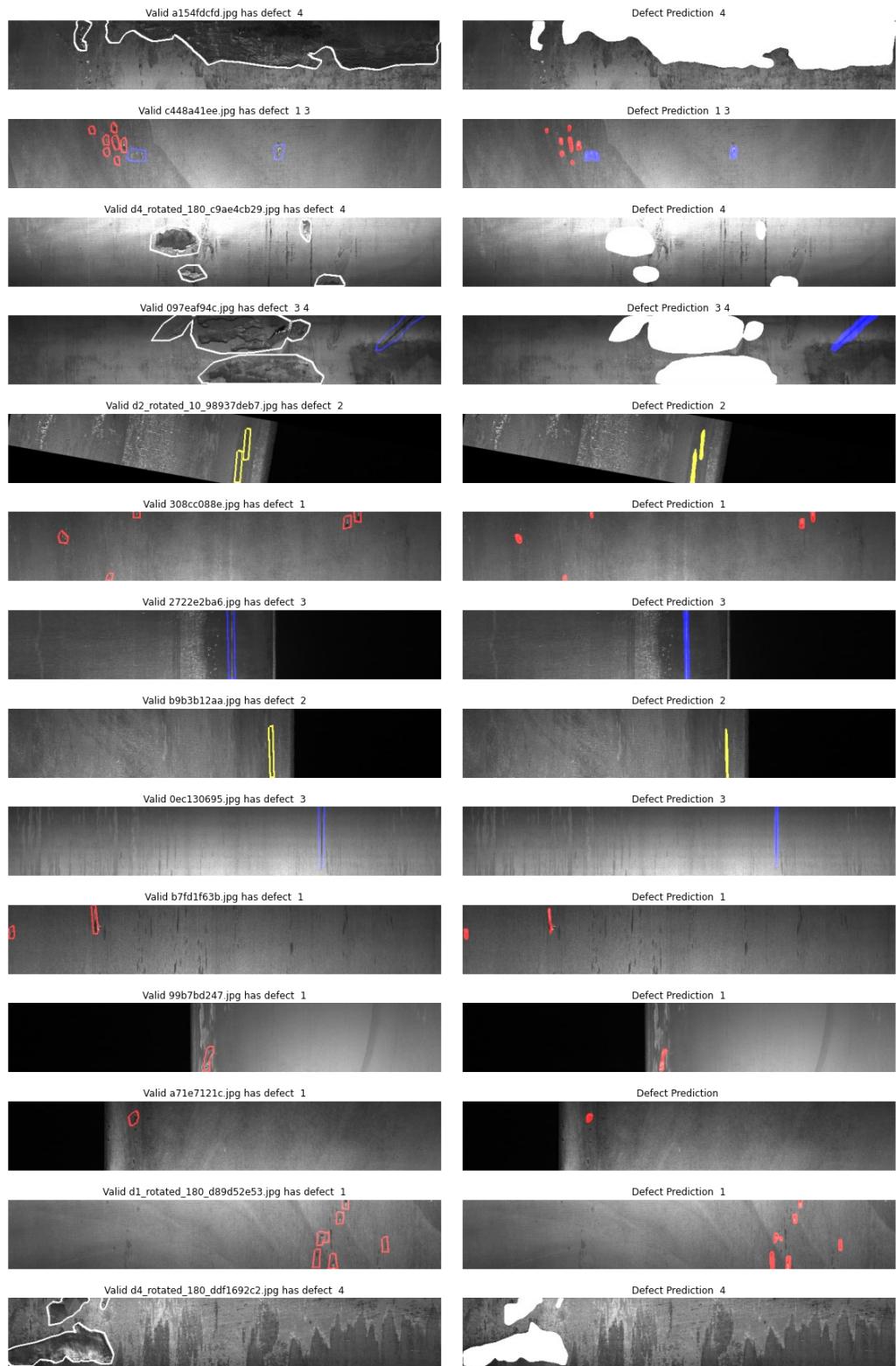


Figure 72: Prediction result on validation data in EfficientNet-B5 – U-Net model with ground-truth compared to predicted masks.

## RESULTS

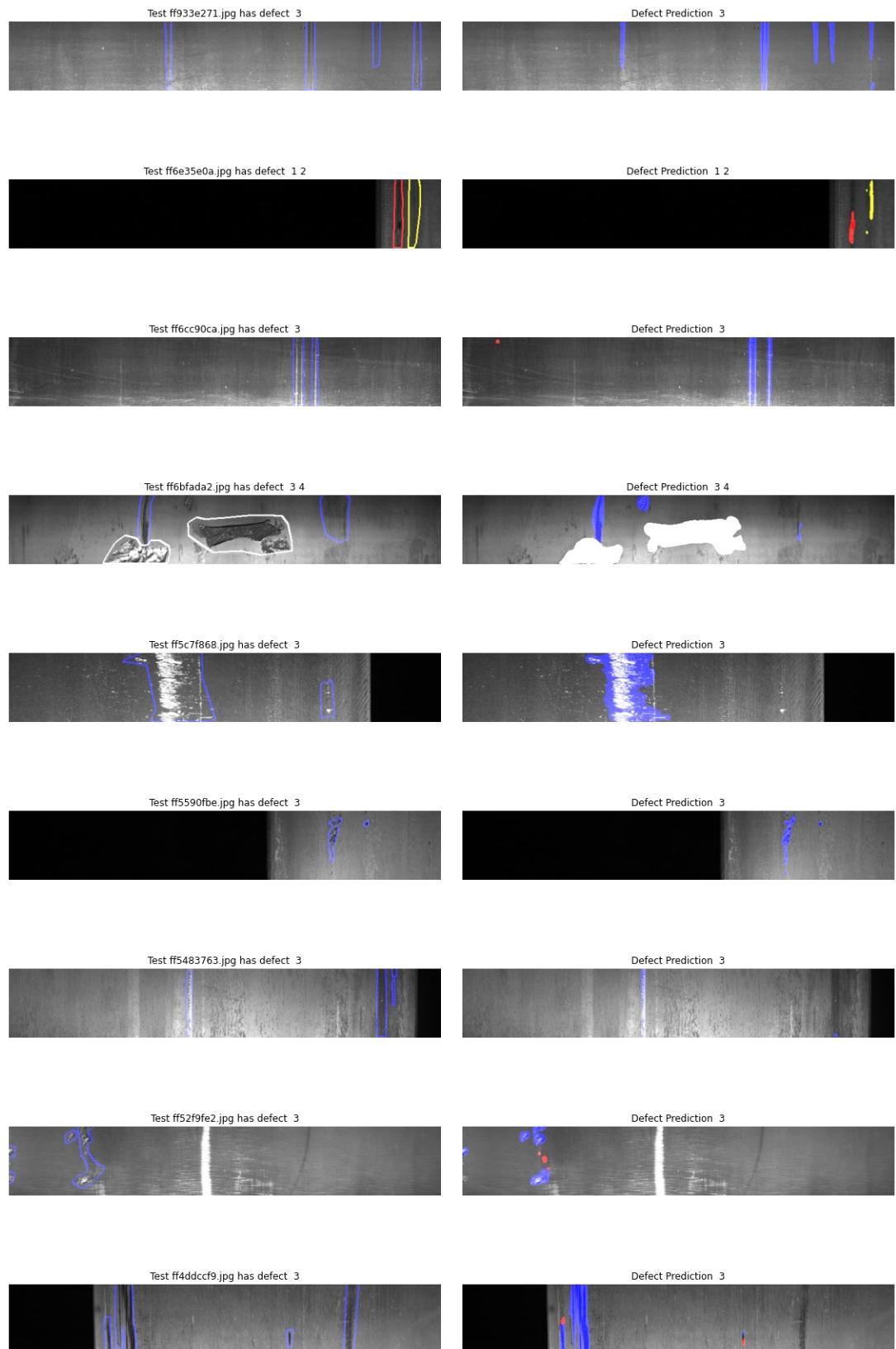
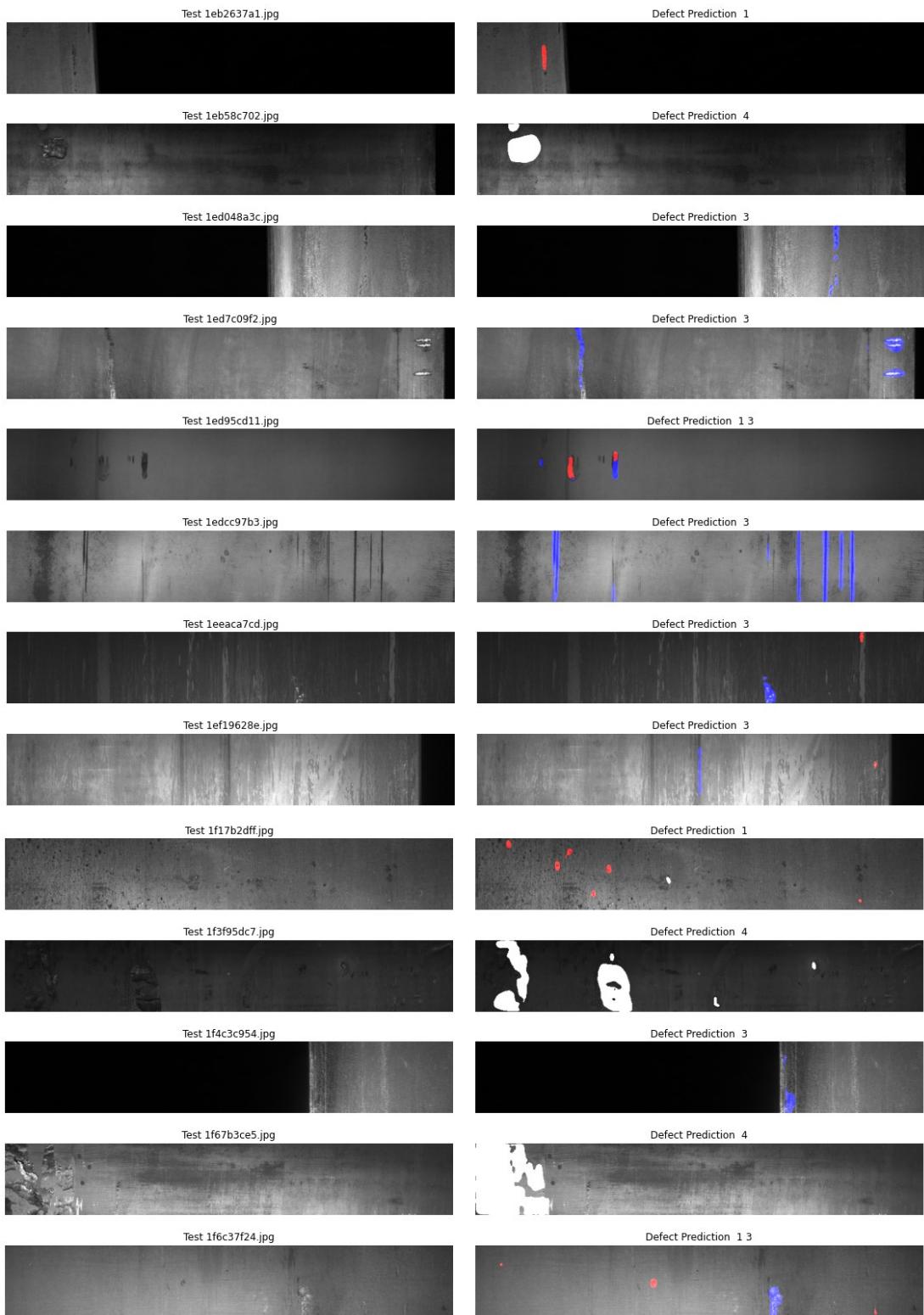


Figure 73: Prediction result on test data in EfficientNet-B5 – U-Net model with ground-truth compared to predicted masks.



*Figure 74: Prediction result on some original test data without ground-truth masks in EfficientNet-B5 – U-Net mode*

## **Chapter 7: Conclusion and Future Works**

In conclusion, with the problem posed to detect defects on images of steel surfaces, along with based on the background knowledge of ML as well as DL, and applying state-of-the-art works in DL, specifically, the combination of ResNets and U-Net architectures, and the combination of EfficientNets and U-Net architectures, the thesis gives a relatively satisfactory result for this problem, which is using semantic segmentation based on these complex architectures with binary cross-entropy of loss and dice coefficient for evaluation metrics, fine-tuning with pre-trained weights of "imagenet" corresponding to backbone type of model and Adam as an optimizer. And cannot lack making training data balanced. This strategy of implementation gave out the best validation of loss about 0.0018. Hopefully, this thesis can contribute something in the practical application of the problem of identifying defects on the surface of steel and materials in general, and a better solution will be found to reach a higher improvement in the future.

Future works will be devoted to finding a way to fine-tune the model for a better result, or another approach based on other state-of-the-art architectures to the development of a backend server to provide a direct API for mobile devices or IoT devices, specifically here devices including camera, which aims to predict steel images containing defects through camera scanning. Another scope is learning and researching how to integrate inference model function on Android platform based on TensorFlow Lite that google provides, it helps to integrate model offline on android devices which is also an interesting idea.

## Bibliography

- [1] G. Cybenko, “*Approximation by superposition of a sigmoidal function*”, Math. Control Signals Systems, 1989.
- [2] F. Chollet. “*Deep Learning with Python*”, Manning, Shelter Island, NY, USA, 1<sup>st</sup> edition, 2017
- [3] I. Goodfellow, Y. Bengio, and A. Courville. “*Deep Learning (Adaptive Computation and Machine Learning Series)*”, MIT Press, Cambridge, MA, USA, 2016.
- [4] <https://www.ibm.com/cloud/learn/gradient-descent>
- [5] Sebastian Ruder. “*An overview of gradient descent optimization algorithms*”
- [6] <https://builtin.com/data-science/gradient-descent>
- [7] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell. “*Understanding convolution for semantic segmentation. In IEEE Winter Conference on Applications of Computer Vision (WACV)*”, pages 1451 - 1460, March 2018.
- [8] Kunihiko Fukushima, “*Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*”, Biological cybernetics, vol. 36, no. 4, pp. 193–202, 1980.
- [9] <https://cs231n.github.io/convolutional-networks/>
- [10] Jonathan Long, Evan Shelhamer, and Trevor Darrell, “*Fully convolutional networks for semantic segmentation*”, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3431–3440.
- [11] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, “*You only look once: Unified, real-time object detection*”, in Proceedings

of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788.

- [12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “*Imagenet classification with deep convolutional neural networks*”, 2012.
- [13] J. Han and C. Moraga, “*The influence of the sigmoid function parameters on the speed of backpropagation learning,*” in Natural to Artificial Neural Computation. IWANN 1995. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1995, pp. 195–201.
- [14] Y. LeCun, Y. Bengio, and G. Hinton, “*Deep learning*,” Nature, vol. 521, no. 7553, pp. 436–444, 2015.
- [15] B. Karlik and A. Vehbi, “*Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks*,” International Journal of Artificial Intelligence and Expert Systems (IJAE), vol. 1, no. 4, pp. 111–122, 2011.
- [16] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, “*Language Modeling with Gated Convolutional Networks*,” arXiv, 2017.
- [17] V. Nair and G. E. Hinton, “*Rectified linear units improve restricted Boltzmann machines*,” Haifa, 2010, pp. 807–814.
- [18] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, and G. E. Hinton, “*On rectified linear units for speech processing*,” in International Conference on Acoustics, Speech and Signal Processing. IEEE, 2013, pp. 3517–3521
- [19] X. Glorot, A. Bordes, and Y. Bengio, “*Deep Sparse Rectifier Neural Networks*,” in International Conference on Machine Learning, 2011
- [20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “*Deep residual learning for image recognition*”
- [21] <https://viso.ai/deep-learning/resnet-residual-neural-network/>

- [22] <https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8>
- [23] <https://medium.com/@nainaakash012/efficientnet-rethinking-model-scaling-for-convolutional-neural-networks-92941c5bfb95>
- [24] <https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html>
- [25] Mingxing Tan, Quoc V. Le, “*EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*”
- [26] <https://www.mygreatlearning.com/blog/fcn-fully-convolutional-network-semantic-segmentation/>
- [27] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “*U-net: Convolutional networks for biomedical image segmentation*”.
- [28] <https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>
- [29] <https://ruder.io/optimizing-gradient-descent/index.html#momentum>
- [30] [https://www.researchgate.net/publication/324701535\\_Cross-Validation](https://www.researchgate.net/publication/324701535_Cross-Validation)
- [31] Bhakti Baheti, Shubham Innani, Suhas Gajre, Sanjay Talbar, “*EffUNet: A Novel Architecture for Semantic Segmentation in Unstructured Environment*”
- [32] <https://medium.com/@wilburdes/semantic-segmentation-using-fully-convolutional-neural-networks-86e45336f99b>
- [33] <https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7>
- [34] <https://medium.com/@nishanksingla/unet-with-resblock-for-semantic-segmentation-dd1766b4ff66>
- [35] <https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a>

- [36] <https://www.kaggle.com/c/severstal-steel-defect-detection/>
- [37] <https://www.expert.ai/blog/machine-learning-definition/>
- [38] <https://www.ibm.com/topics/computer-vision>
- [39] <https://xd.adobe.com/ideas/principles/emerging-technology/what-is-computer-vision-how-does-it-work/>
- [40] <https://medium.com/analytics-vidhya/image-classification-vs-object-detection-vs-image-segmentation-f36db85fe81>
- [41] <https://viso.ai/deep-learning/image-segmentation-using-deep-learning/>
- [42] <https://developer.nvidia.com/deep-learning>
- [43] <https://www.geeksforgeeks.org/object-detection-vs-object-recognition-vs-image-segmentation/>
- [44] <https://towardsdatascience.com/hyper-parameter-tuning-techniques-in-deep-learning-4dad592c63c8>
- [45] <https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5>
- [46] <https://towardsdatascience.com/statistics-is-freaking-hard-wtf-is-activation-function-df8342cdf292>
- [47] <https://study.com/academy/lesson/how-to-recognize-linear-functions-vs-non-linear-functions.html>