

Image Processing and AI Applied to Weldings

Ioannis Papachlimintzos

Diploma Thesis



School of Naval Architecture and Marine Engineering
National Technical University of Athens

Supervisors: Associate Professor G. Papalambrou, Professor M. Samuelides

Committee Members:

Assistant Professor A. Zervaki

November 2022

Acknowledgements

The present thesis constitutes my final effort within the five years of studies at the School of Naval Architecture and Marine Engineering of the National Technical University of Athens (NTUA). At this point, I would like to express my special thanks of gratitude to my supervisor Assistant Professor George Papalambrou who with his essential support and effective guidance helped me to complete my study. His effective teaching and our subsequent collaboration introduced me to the world of computer vision and artificial intelligence and inspired me during the development of my work. I would also like to thank Professor Manolis Samuelides and Assistant Professor Anna Zervaki for their guidance and collaboration for the purposes of this study. I am also grateful to Mr. Panagopoulos, surveyor at Hellenic Coast Guard and the M&C Group for providing images of ship inspections. Last but not the least, I would like to thank my family, for their continued support and encouragement throughout my studies. They stood by my side helping me through my efforts and difficulties in the struggle to achieve my target.

Abstract

Marine automated inspections using Unmanned Aerial Vehicles (UAVs), Remotely Operated Vehicles (ROVs) are emerging technologies which are constantly gaining ground. Intelligent vehicles is essential to have an environmental perception that provides crucial information about each ship's feature so then to classify and inspect it according to the regulations. In this thesis, the main task is research about the application of image segmentation in welding joints. A Fully Convolution Neural Network is proposed based on UNet architecture which was modified so that VGG16 to be implemented as an encoder following a couple of transfer learning strategies. Decoder's convolutional layers were reduced by replacing one layer on each block with Batch Normalization and Dropout operations in order to minimize computational cost and increase model's accuracy. The dataset used for the training and testing of the model consists of images with welding joints which were collected from school's laboratory, ship surveys as well as from the internet (300 images) to achieve greater diversity and increase model's robustness. The experimental results of the testing set show that the mean IoU is 0.46 and mean F1-score is 0.60.

Contents

List of Figures	6
1 Introduction	8
2 Literature Review	10
3 Theoretical Basis	12
3.1 Computer Vision - Image Processing	13
3.1.1 Grayscale and RGB Models	13
3.1.2 Image Transformation	15
3.1.3 Image Filtering	16
3.2 Neural Networks	19
3.2.1 Artificial Neural Networks	19
3.2.2 Network Training	22
3.2.3 Optimizers	24
3.3 Deep learning and Convolutional Neural Networks	27
3.3.1 Convolution Layer	28
3.3.2 Pooling	30
3.3.3 Transposed Convolution and Deconvolution	31
3.3.4 Batch normalization	32
3.3.5 Dropout	32
3.3.6 VGGNet	33
3.3.7 Image segmentation using Deep Neural Networks	34
3.3.8 Transfer Learning	35
4 Welding Datasets	37
4.1 Images	37
4.2 Image Annotation	42
5 Methodology	44
5.1 Hardware and software used	44
5.2 Data Preparation and Augmentation	44
5.3 Model architecture	47
5.4 Loss function & Optimizer	47
5.5 Evaluation of network performance	48
5.6 VGG16 Feature Extraction Testing	50
6 Results	54
6.1 General information about training	54
6.2 Results	55

<i>CONTENTS</i>	4
7 Conclusion	67

List of Figures

3.1	Artificial Intelligence Timeline	12
3.2	Representation of image as a matrix	13
3.3	8-bit grayscale ranging from black to white	14
3.4	RGB Color Model: (a) Primary colors representation; (b) Primary colors cube	14
3.5	Basic 2D planar transformations	16
3.6	Top left: Original Image. Right and bottom: Displacement fields and the resulting image	16
3.7	Example of Neighborhood filtering	17
3.8	Binary image morphology: (a) original image; (b) dilation; (c) erosion; (d) majority; (e) opening; (f) closing.	18
3.9	Non linear model of a neuron	19
3.10	Sigmoid, tanh, and ReLU functions	20
3.11	The sigmoid function and its derivative	21
3.12	Simple Feed-Forward Neural Network architecture	21
3.13	Geometrical view of the error function as a surface sitting over weight space	23
3.14	A small learning rate makes the model converge slowly to the global minimum loss.	25
3.15	AdaGrad vs Gradient Descent: the former can correct its direction earlier to point to the optimum	26
3.16	Performance Comparison on Training cost using different optimizer	27
3.17	CNN layers with rectangular local receptive fields	28
3.18	Connections between layers and zero padding	29
3.19	Dimensionality reduction using a stride of 2	29
3.20	Convolution layers with multiple feature maps	30
3.21	Pooling Layer	31
3.22	Example of 2x2 and stride 2 Max and Average Pooling	31
3.23	Transpose Convolution by 2x2 kernel and stride 2	32
3.24	Visualisation of Dropout	33
3.25	VGG16 Architecture	34
3.26	Semantic Segmentation vs Instance Segmentation	34
3.27	U-net architecture	35
3.28	Three ways in which transfer might improve learning	35
3.29	Transfer learning strategies	36
4.1	Robotic arm for automatic welding, MSST Lab of NTUA	38
4.2	Samples of weld form NTUA LAB	39
4.3	Samples of Images found on internet	40

4.4	Samples of Images from weld on ship	41
4.5	Segments.ai: The software segments the image and the weld is marked(white)	43
4.6	Segments.ai: Sample of quite difficult circumference to be delineated	43
4.7	RGB Image and its Binary Mask	43
5.1	Samples of Augmentation techniques: (a)Initial Image; (b)Initial Mask; (c)Gaussian Blur (d)Color Jitter (e)Elastic Distortion + Flip Y; (f)Mask of (e)	46
5.2	Architecture of UNet-VGG16 with transfer learning	47
5.3	Illustration of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) of the segmentation in a binary case. The blue and red ellipses represent the ground truth component and component recognized by the segmentation method.	49
5.4	A sample image tested	50
5.5	Feature maps exported on Block 1, size 224x224	51
5.6	Part of feature maps exported on Block 2, size 112x112	52
5.7	Part of feature maps exported on Block 3, size 56x56	52
5.8	Part of feature maps exported on Block 4, size 28x28	53
5.9	Part of feature maps exported on Block 5, size 14x14	53
6.1	Model - Case A: BCE loss	55
6.2	Model - Case A: Testing Set - Mean values; IoU, Recall, Precision	55
6.3	Model - Case A results	56
6.4	Model - Case B: BCE loss	58
6.5	Model - Case B: Testing Set - Mean values (a)Summary of Metrics; (b)IoU(th=0.3 and 0.5); (c)Recall(th=0.3 and 0.5), (d)Precision(th=0.3 and 0.5)	58
6.6	Model - Case B: Testing Set - Epoch 150 (a)Precision vs Recall ; (b)Precision vs Recall (th=0.3)	59
6.7	Model - Case B Testing set results 1	60
6.8	Model - Case B Testing set results 2	61
6.9	Model - Case B Testing set results 3	62
6.10	Model - Case B Testing set results 4	63
6.11	Model - Case B Testing set results 5	64
6.12	Model - Case B Testing set results 6	65
6.13	Model - Case B Training set	66

List of Tables

4.1	Images Dataset	38
5.1	Comparison of CPU and GPU	44
6.1	VGG16-UNet parameters	54
6.2	Summary of metrics at Epoch 150	57

Chapter 1

Introduction

Welding is undoubtedly a fundamental process in the contemporary manufacturing industries. The development of modern welding techniques during the Industrial Revolution (19th - 20th centuries) took the place of riveted joints, due to its advantages concerning mainly water-tight and oil-tight properties, better durability and overall a more lightweight structure. The shipbuilding industry has radically changed since then by utilizing arc welding for the majority of steel joints. Arc welding is a group of welding processes that produce coalescence of metals by heating them with an electric arc. [1]

Over the last half century the evolution of welding technology and past experience of welding defects and failures which led to marine accidents have raised SOLAS' (International Convention for the Safety of Life at Sea) minimum safety standards for the safe construction of ships. Ensuring good welding quality is vital for the structure in order to remain intact and functional (durable) even when extreme static and dynamic loads are applied. Therefore, rules and guidelines for shipbuilding and maintenance procedures have been established and incorporated to (in) all ship classification societies' requirements and guidelines, such as American Bureau of Shipping, Lloyd's Register and the International Association of Classification Societies (IACS). Quality control of hull welds shall be performed using non-destructive methods to locate possible defects. As far as the surface imperfection is concerned, visual testing, magnetic particle and penetrant testing can be applied, whereas for sub-surface imperfections ultrasonic and radio-graphic testing shall be conducted. [2]

Nowadays Welding Inspections are mainly carried out by human surveyors under vessel's problematic and dangerous conditions. Prior to each survey, a costly and time-consuming preparation of vessel needs to be done (e.g. such as safety equipment, lighting, ventilation and gas free certificates) in order for the safety precautions to be arranged. The vessel can contain hundreds of kilometers of weld lines while surveys tend to be performed and completed as promptly as possible, thereby are prone to oversights and human errors. [3]

In latest years, poised for explosive growth, Unmanned Aerial Vehicles (UAVs) and Remotely Operated Vehicles (ROVs) are emerging technologies with limitless potential and application in marine surveys. Since 2016 DNV GL has been using camera-equipped drones in surveys [4] and since 2019 DNV has approved service suppliers to provide close-up surveys using remote inspection techniques [5] (RIT) for ships and mobile offshore

units. An automated inspection vehicle equipped with latest breakthrough of cameras and sensors (e.g. LiDAR) is essential to have an environmental perception that provides crucial information about each ship's feature so then to classify and inspect it according to the regulations.

The goal of this study is the weld detection on an image by creating a binary segmentation mask. This is implemented using Fully Convolutional Deep Neural Networks and potentially can be integrated in such devices as [©]Robohop [6]. In Chapter 2 an overview of the literature is presented. Chapter 3 elaborates the fundamental theoretical concepts regarding image processing and deep neural network algorithms. In Chapter 4 is presented the dataset used for the training and testing phase. In Chapter 5 is presented and explained the methodology followed to train the neural network. In Chapter 6 the results are presented and analyzed through diagrams and visual examples. Finally, in Chapter 7 are presented the conclusions of this work.

Chapter 2

Literature Review

There is a large volume of published studies investigating classification and segmentation tasks in welding technology. Traditional image processing and machine learning algorithms created for welding defects or weld segmentation need human interference to feed algorithms with proper data, usually taken from a stable environment, and sometimes to set by hand parameters such as thresholds [7] [8]. The emergence of deep learning technology has dominated over the old machine learning techniques and so nowadays the scientific community mainly uses deep Convolutional Neural Networks (CNN) for computer vision tasks

Zhifen Zhanga, Guangrui Wena, Shanben Chen [9] proposed a CNN classification model with 11 layers to identify weld penetration defects. In order to improve the generalization ability of the CNN model, weld images from different welding current and feeding speed were captured for the CNN model. Then they applied augmentation techniques to the images to increase model's robustness.

A more complex architecture, the Faster R-CNN, used by Chenhua Liu , Shen Chen and Jiqiang Huang [10] for the detection of weld area using a bounding box. Transfer learning technique was also employed for the convolutional core of their model. They compared 6 different state-of-the-art pre-trained models used as backbone in order to define the effectiveness of each model. Their study shows that the VGG16 model is the best in weld seam area recognition with accuracy 91.68%.

Additional studies, where CNN has been applied to weld defects detection are [11][12].

Studies regarding image segmentation in welds using fully convolutional neural networks have also been successfully conducted. Yang Lei, Huaixin Wang, Benyan Huo, Fangyuan Li, and Yanhong Liu from Zhengzhou University [13] evaluated an automatic welding defect location method based on an improved U-net architecture. Digital X-ray images were used for welding defects location and the dataset was augmented to increase robustness. Their experiments showed that the method could acquire the detection precision up to 88.4%. Weld seams semantic segmentation method has been proposed by a group of academics in Leibniz University Hannover [14] who use deep learning and Xception model architecture to perform semantic segmentation of micrographs of complex weld areas to achieve the automatic detection and quantization of weld seam properties achieving a value of around 95% for weld contour detection and

76.88% of mean intersection over union.

Given weld seam forming results from the nonlinear formation of multiple welding parameters, studies have shown that convolutional neural networks (CNN) with nonlinear properties can successfully cope with tasks concerning weld seam detection or the detection of defects on it.

Chapter 3

Theoretical Basis

Artificial Intelligence (AI) as a term was first coined in 1956 by John McCarthy, also known as the father of AI, at the Dartmouth Summer Research Project. During that meeting John McCarthy and his team tried to clarify and develop their views about thinking machines [15]. A few years before, a milestone in AI history was set in 1950, when Alan Turing published the paper "COMPUTING MACHINERY AND INTELLIGENCE" and formed the basis for artificial intelligence [16]. He also raised the crucial question "Can a machine think ?" and introduced the *Imitation Game*, a deceptively simple method determining whether a machine can think intelligently like humans.

Machine learning (ML) is a subset of AI that enables a computer system to make predictions or take decisions using historical data without being explicitly programmed. Machine learning uses a massive amount of structured and semi-structured data so that a model can generate accurate results or give predictions based on that data.

Deep learning (DL) is an outgrowth of Machine learning and differs from machine learning techniques in that it can automatically learn representations from training data (e.g. images, video, text) by creating non linear and complex correlations. That can be achieved by artificial deep neural networks which try to mimic the human brain neurons' functionality like interconnections and their learning process.

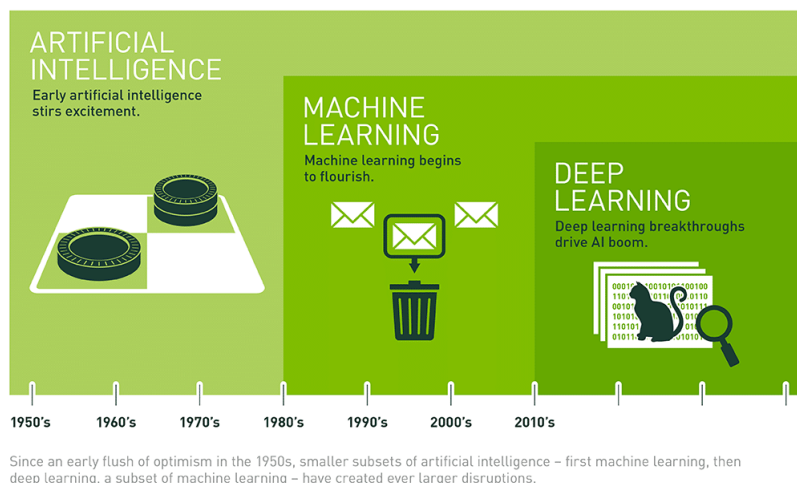


Figure 3.1: Artificial Intelligence Timeline

The implementation of Graphics Processing Units (GPUs) in the Convolutional Neural Network (CNN) training process [17] led to breakthroughs. Nowadays, the exponentially increasing development of GPUs provides a solution in large-scale computer vision and speech-based applications which demand high computational power.

3.1 Computer Vision - Image Processing

Computer vision is a field of artificial intelligence (AI) that trains computers and systems to interpret and understand the visual world by analyzing digital images. Richard Szeliski mentions in his book [18] the difficulties and challenges to cope with Computer Vision as he calls it *"(it is) an inverse problem, in which we seek to recover some unknowns given insufficient information to fully specify the solution and we must therefore resort to physics-based and probabilistic models, or machine learning from large sets of examples, to disambiguate between potential solutions."*

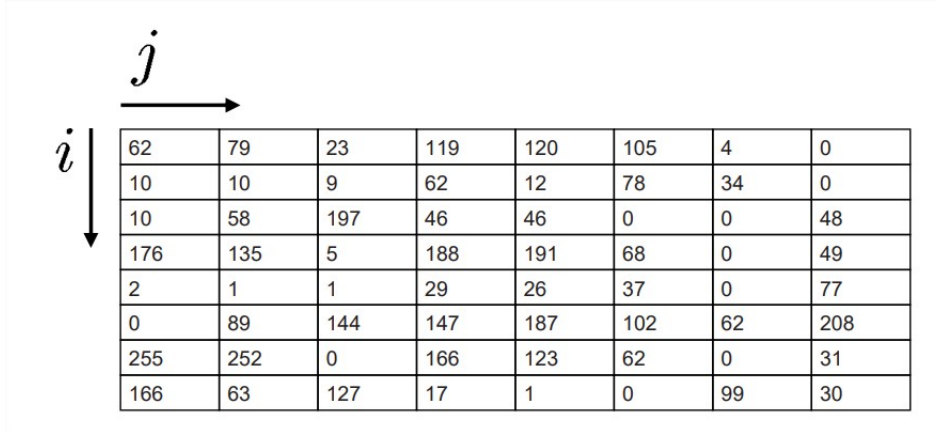
An image can be defined as continuous-tone and spatially continuous brightness distribution:

$$f : R^2 \rightarrow R \quad (3.1)$$

where $f(x,y)$ is the intensity of spatial (plate) coordinates (x,y) . For computer processing an image $f(x, y)$ is sampled so that the resulting digital image has i rows and j columns. The values of the coordinates (x,y) now become discrete quantities and the continuously varying brightness f at each sample is quantized (3.2) (converted to a one of set of integers).

$$f[i, j] = \text{Quantize}f(i\Delta, j\Delta) \quad (3.2)$$

where Delta is the distance of samples.



62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

Figure 3.2: Representation of image as a matrix

3.1.1 Grayscale and RGB Models

Grayscale

The Figure 3.2 above could be the representation of a grayscale image. Each of these (i, j) cells correspond to a pixel which is denoted as the numerical value and these

numbers are called Pixel Values. These pixel values denote the intensity of the pixels. For a grayscale or black and white image, we have pixel values ranging from 0 to 255. The smaller numbers closer to zero represent the darker shade while the larger numbers closer to 255 represent the lighter or the white shade.

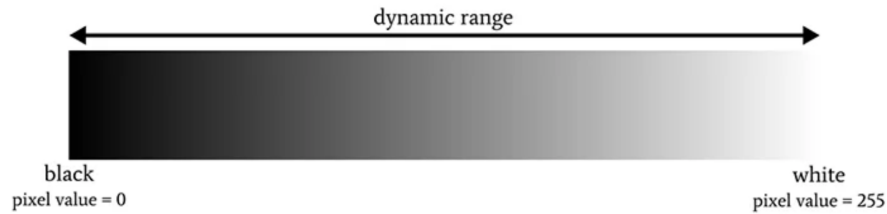


Figure 3.3: 8-bit grayscale ranging from black to white

RGB

The RGB color model, one of the most well-known multi-channel color models, specifies colors in three primary colors (three channels or components), i.e., red (R), green (G), and blue (B), and is an additive color model in which red, green, and blue light are combined in various ways to reproduce a broad array of colors. Each of these metrics would again have values ranging from 0 to 255 where each of these numbers represents the intensity of the pixels. Finally, all of these channels or all of these matrices are superimposed so the shape of the image, when loaded in a computer, will be $N \times M \times 3$ where N is the number of pixels across the height, M would be the number of pixels across the width, and 3 is representing the number of channels, in this case, we have 3 channels R, G, and B. If all components are of the highest intensity, then the resulting color is white (255,255,255). In the RGB color model, one original color image can be constructed from three gray-scale value images (channels or components), as shown in [Figure 3.4](#). As the most widely used color space for sensing, representation, and display of images in electronic systems, the RGB color model plays a very important role in the image processing field. [19]

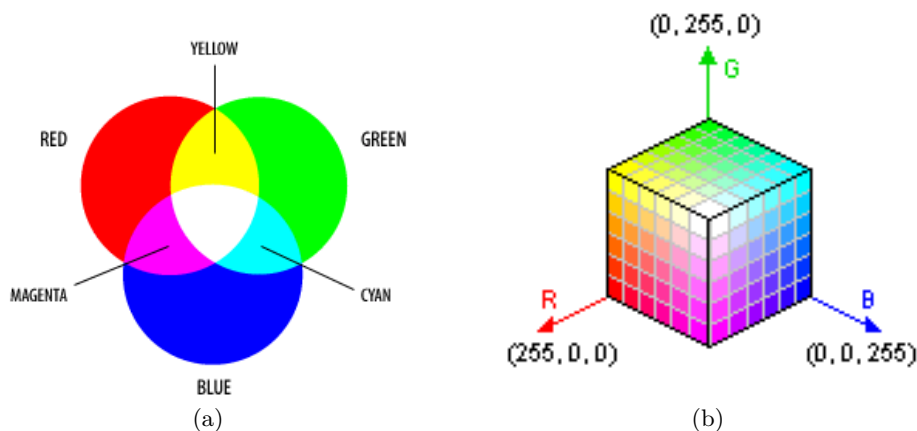


Figure 3.4: RGB Color Model: (a) Primary colors representation; (b) Primary colors cube

3.1.2 Image Transformation

Given a classification task, when the dataset is limited, one may apply transformations to generate additional data and let the learning algorithm infer the transformation invariance. This invariance is embedded in the parameters, so it is in some sense free, since the computation at recognition time is unchanged. If the data is scarce and if the distribution to be learned has transformation-invariance properties, generating additional data using transformations may even improve performance. The pixel coordinates in an image (2D points) can be denoted using a pair of values $\mathbf{x} = (x, y) \in R^2$ or:

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} \quad (3.3)$$

The new target location, at position (x,y) is given with respect to the previous position. There are two types of transformation that can be applied on a 2d matrix.

Rigid

Rigid called a transformation of a geometric object when the distance between each pair of points of the object is preserved. In other words, rigid transformations preserve the shape and size of an object, such as:

- Reflection - e.g. Flip across y

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (3.4)$$

- Translation

$$\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \end{aligned} \quad (3.5)$$

- Rotation - e.g. About the origin by an angle θ

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (3.6)$$

or a combination of these.

Non-Rigid

Non - rigid transformations can change both the size and the shape of the image using simple variations of the above transformations. In addition, more complex algorithms can be applied to create a displacement field with random deformation of each pixel such as the elastic distortion technique ([Figure 3.6](#)), proposed by Simard, P.Y. and Steinkraus, D. and Platt, J.C. in 2003 [\[20\]](#)

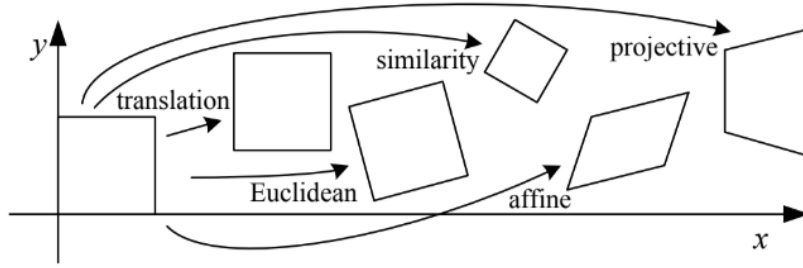


Figure 3.5: Basic 2D planar transformations

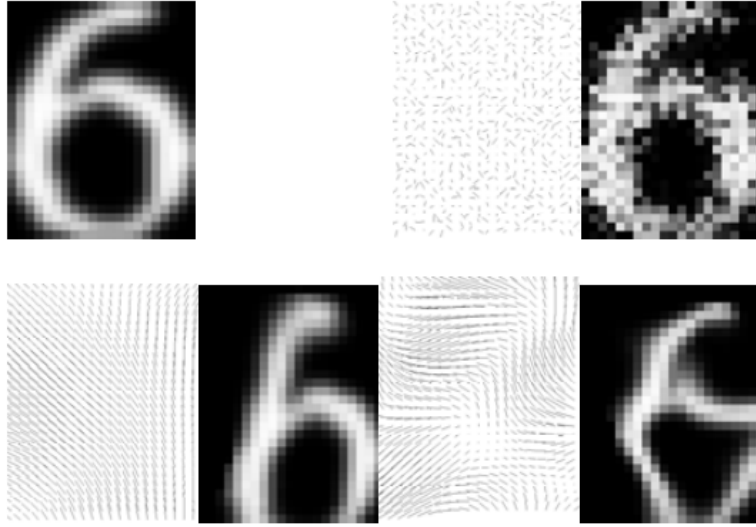


Figure 3.6: Top left: Original Image. Right and bottom: Displacement fields and the resulting image

The image transformation types may be applied on a case-by-case basis. It is essential for users to visualize each transformed image and check if it could be a realistic scenario. An image that does not correspond to reality may confuse the model and have opposite effects to that intended.

For example in a number classification task, an image with the letter "six" can not be rotated 180 deg (upside down) because it will resemble the "nine".

3.1.3 Image Filtering

Linear Filters

Neighborhood operator or local operator [18] is a filtering process which uses a collection of pixel values in the vicinity of a given pixel to determine its final output. The most widely used type of Neighborhood operator is the linear filter, where an output pixel's value is a weighted sum of pixel values in the input pixel's neighborhood N (see [Figure 3.7](#)) and it is expressed by the *correlation* operation:

$$g = f \otimes h \quad (3.7)$$

$$g(i, j) = \sum f(i + k, j + l)h(k, l) \quad (3.8)$$

where the k, l are called filter coefficients and h is called kernel or mask. A common variant on this formula by reversing the sign of the offsets in f is:

$$g(i, j) = \sum_{k, l} f(i - k, j - l)h(k, l) = \sum_{k, l} f(k, l)h(i - k, j - l) \quad (3.9)$$

The convolution operator is given by the equation:

$$g = f * h \quad (3.10)$$

where h is the impulse response function. Equation (3.10) can be interpreted as the superposition (addition) of shifted impulse response functions $h(i - k, j - l)$ multiplied by the input pixel values $f(k, l)$.

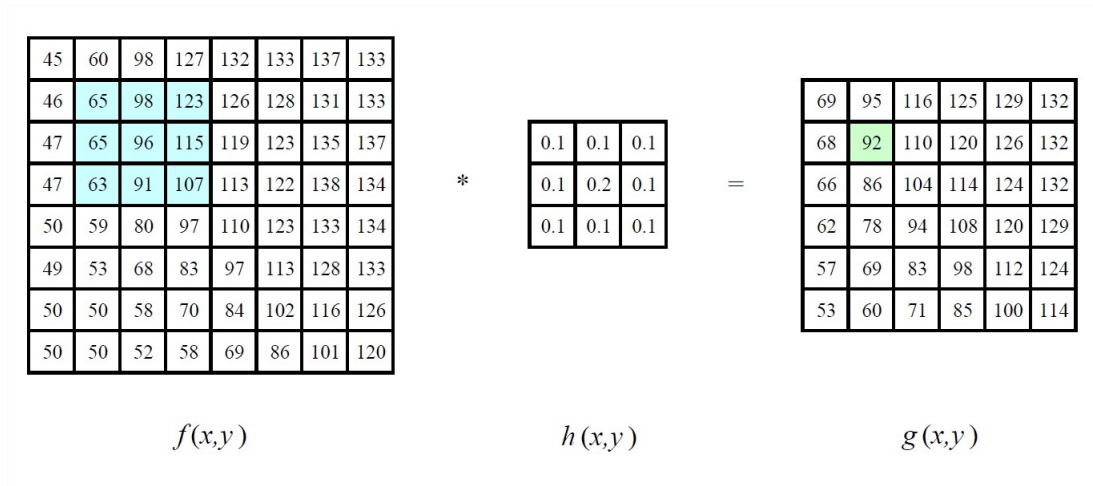


Figure 3.7: Example of Neighborhood filtering

Types of widely used linear filters are listed below:

- Moving average or box filter, which simply averages the pixel values in a $K \times K$ window.
- Gaussian filters, are good for reducing small standard deviations at the cost of some blurring.
- Bartlett filter, which is used to smoothen an image by convolving the image with a piece-wise linear “tent” function.

Non-Linear Filters

Non-linear filters indicate better performance than linear ones, for example in case where the noise has very large values and the linear Gaussian filter cannot suppress it completely. An important class of nonlinear filters is the statistical filters, such as the median filter which operates over a window by selecting the median value from each pixel’s neighborhood and the bilateral filter, a variation of the median filter with an additional weighting term.

In addition, non-linear filters are frequently used to process binary images (consist only of black and white pixels). Such images can be created by applying a thresholding operation to a grayscale image:

$$\theta(f, t) = \begin{cases} 1 & \text{if } f \geq t, \\ 0 & \text{else} \end{cases} \quad (3.11)$$

The most common binary image operations are called morphological operations because they change the shape of the underlying binary objects. The simplest operation can be applied is the convolution of a binary image f with a 3×3 structuring element s :

$$c = f \otimes s \quad (3.12)$$

The standard operations used in binary morphology include dilation, erosion, majority, opening and closing, the result of each is shown in the image below.

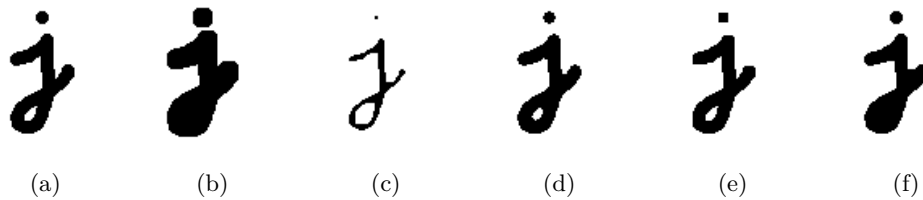


Figure 3.8: Binary image morphology: (a) original image; (b) dilation; (c) erosion; (d) majority; (e) opening; (f) closing.

3.2 Neural Networks

3.2.1 Artificial Neural Networks

Model of Artificial Neuron

A neuron is an information - processing unit which is the fundamental element of the neural network. The architecture of the neuronal model is shown in [Figures 3.9](#) where can be identified three basic elements:

- The set of synapses or connecting links that receive the input signal x_j and multiply it by the corresponding weight w_{kj} .
- The adder that sums the bias and the products of input signal and weight.
- The activation function (also referred as squashing function) that limits the amplitude of neuron's output v_j . Typically, the normalizes amplitude range of the output of a neuron is written as the close unit interval $[0,1]$ or $[-1, 1]$.

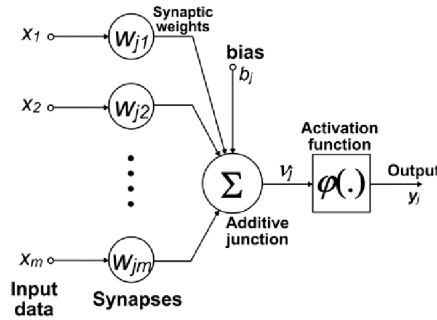


Figure 3.9: Non linear model of a neuron

In mathematical terms, the above statements can be written as:

$$u_k = \sum_{j=1}^m w_{kj} x_j \quad (3.13)$$

and

$$y_k = \phi(u_k + b_k) \quad (3.14)$$

where x_j the input signals, w_{kj} the synaptic weight, u_k is the linear combiner output, b_k the bias, $\phi(\cdot)$ the activation function and y_k the output of neuron k.

The activation functions are a key step in the formation of value of neuron's output. As mentioned above, they limit and normalize the output value as well as provide non-linearity, which aids the learning of high order polynomials for deeper networks. Even in linear correlation between input and output, activation functions are required to convert it to non-linear. The most widely used activation functions are listed below and plotted in [Figure 3.10](#).

Activation functions

Neural Networks - A Comprehensive Foundation - Simon Haykin.pdf

- **Sigmoid**

Sigmoid is a preferred activation function as it maps any real value to the range (0,1) and is defined as:

$$f(x) = \frac{1}{1 + e^{-ax}} \quad \forall x \in R \rightarrow f \in (0,1) \quad (3.15)$$

Provided that it is useful for AI applications where a real number needs to be converted to a probability and therefore it usually placed as the last layer of a AI model. A drawback of using sigmoid function is the called "Vanish gradient", which occurs near the limits of (0,1). When the inputs of the sigmoid function becomes larger or smaller, the derivative becomes close to zero, as shown in [Figure 3.11](#).

- **Tanh**

The tanh activation function is like the sigmoid but the output is scaled in the range of (-1,1), shifted and stretched:

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \quad \forall x \in R \rightarrow f \in (-1,1) \quad (3.16)$$

The tanh, in contrast with sigmoid, suppress the input value in larger scale and offers stronger gradients and bigger learning steps

- **ReLU**

ReLU derives form Rectified Linear Unit and nowadays is the most used activation function in the world as it mainly operates at the output of convolutional layers and is defined by the formula:

$$f(x) = \max(0, x) \quad \forall x \in R \rightarrow f \in [0, \infty) \quad (3.17)$$

It is computationally efficient since there is a lack of expensive exponential operations and just need to pick $\max(0, x)$. The main drawback of ReLU, commonly called "Dying ReLU", is that during training, some neurons effectively die, meaning they stop outputting anything other than 0. For that reason, ReLU activation function variants have been developed such as LeakyReLU ELU and SELU to alleviate that issue.

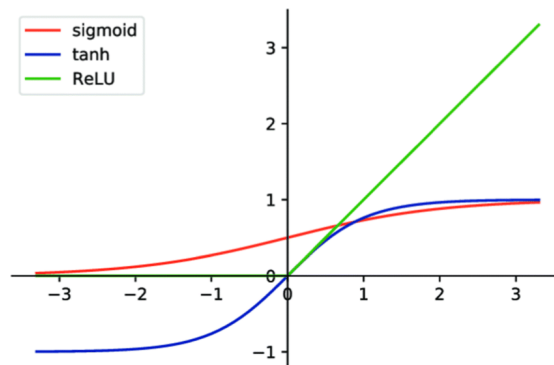


Figure 3.10: Sigmoid, tanh, and ReLU functions

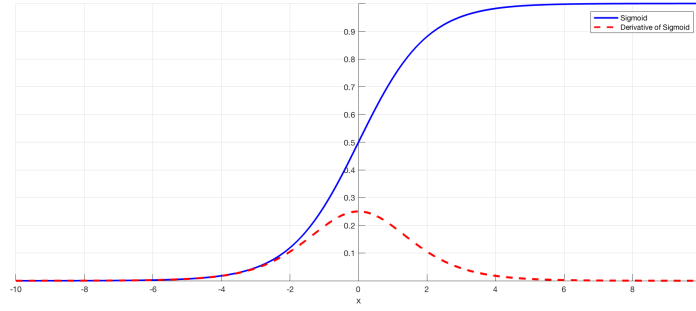


Figure 3.11: The sigmoid function and its derivative

The choice of activation function is determined by the nature of the data and the assumed distribution of target variables

Neural Network Architecture

One well-ordered cluster of neurons forms a neural network whose simplest form is depicted in [Figure 3.12](#). The information travels from Input towards the output layer through the intermediate layers, which are called hidden layers and consist the main core of the model.

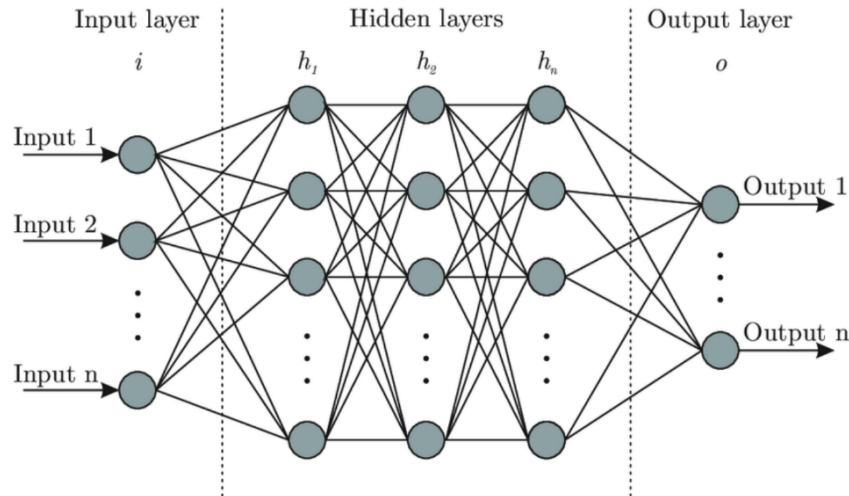


Figure 3.12: Simple Feed-Forward Neural Network architecture

The neural network can be visualized as a non-linear function $y(x)$ with millions of unknown parameters that are determined during the training phase. There is no exact recipe for adopting a type of model and its specifications (Number of hidden layers, activation function etc) which can present the highest accuracy to a given task. Nevertheless, the extended literature can lead the way in the right direction. Different types of neural combinations have been developed and tested such as Convolution Neural Networks, LSTM, RNN etc each one of them is used according to the data and the task to be addressed.

3.2.2 Network Training

So far, we have defined neural networks as a general class of parametric nonlinear functions from a vector \mathbf{x} of input variable to a vector \mathbf{y} of output variables. A simple approach to the problem of determining the network parameters is to visualize it as a polynomial curve fitting where the precise form of the function is determined during the training phase, also known as the learning phase, by dynamically adjusting the parameters to minimize sum-of-squares error functions [21]. Given a training set comprising a set of input vectors x_n , where $n = 1, 2, \dots, N$, together with a corresponding set of target vectors t_n , the following error function is to be minimized:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \|y(x_n, w) - t_n\|^2. \quad (3.18)$$

However, by taking the advantage of probabilistic predictions and giving this kind of interpretation to the network outputs, a much more general view of network training can be provided.

$$p(t|x, \mathbf{w}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}) \quad (3.19)$$

where β is the precision (inverse variance) of the Gaussian noise. For the conditional distribution given by 3.19, it is sufficient to take the output unit activation function to be the identity, because such a network can approximate any continuous function from x to y . Given a data set of N independent, identically distributed observations $X = \{x_1, \dots, x_N\}$, along with corresponding target values $t = \{t_1, \dots, t_N\}$, we can construct the corresponding likelihood function:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^N p(t_n|x_n, w, \beta). \quad (3.20)$$

By taking the negative logarithm, we obtain the error function:

$$\frac{\beta}{2} \sum_{n=1}^N \{y(x_n, w) - t_n\}^2 - \frac{N}{2} \ln \beta + \frac{N}{2} \ln(2\pi) \quad (3.21)$$

which can be used to learn the parameters w and β . It should be mentioned that in the neural networks literature, the primary goal is the minimization of an error function rather than the maximization of the (log) likelihood, and so here we shall follow this convention. Consider first the determination of \mathbf{w} . Maximizing the likelihood function ($p=1$) is equivalent to minimizing the sum-of-squares error function given by:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, w) - t_n\}^2 \quad (3.22)$$

where we have discarded additive and multiplicative constants. The value of \mathbf{w} found by minimizing $E(\mathbf{w})$ will be denoted w_{ML} because it corresponds to the maximum likelihood solution. In practice, the nonlinearity of the network function $y(x_n, \mathbf{w})$ causes the error $E(\mathbf{w})$ to be nonconvex, and so in practice local maxima of the likelihood may be found, corresponding to local minima of the error function.

In the case of a binary classification in which we have a single target variable t such that $t = 1$ denotes class C_1 and $t = 0$ denotes class C_2 . We consider a training set of

independent observations. In the same way, the error function which is given by the negative log likelihood, is then a cross-entropy error function of the form:

$$E(\mathbf{w}) = - \sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln 1 - y_n\} \quad (3.23)$$

where $y(\mathbf{x}, \mathbf{w})$ the output of a network whose activation function is sigmoid and the conditional distribution of targets Bernoulli distribution

Parameter Optimization

As mentioned above, the principal task when training a neural network is to find a weight vector \mathbf{w} which minimizes the chosen function $E(w)$. At this stage, it is useful to visualize the error function as a surface sitting over weight space as shown in [Figure 3.13](#). We can make a small step in weight space from \mathbf{w} to $\mathbf{w} + \delta\mathbf{w}$ to change the error function by $\delta E \approx \delta\mathbf{w}^T \nabla E(\mathbf{w})$. Because the error $E(\mathbf{w})$ is a smooth continuous function of \mathbf{w} , its smallest value will occur at a point in weight space such that the gradient of the error function vanishes so that:

$$\nabla E(\mathbf{w}) = 0 \quad (3.24)$$

as otherwise, we could make a small step in the direction of $-\nabla E(\mathbf{w})$ and thereby further reduce the error.

The calculation of a vector \mathbf{w} such that $E(\mathbf{w})$ takes its smallest value is the ultimate goal. However it is not always possible because the error function has a highly nonlinear dependence on the weights and bias parameters, and so there will be many points in weight space at which the gradient vanishes.

Considering that there is clearly no hope of finding an analytic solution to the equation $\nabla E(\mathbf{w}) = 0$ we resort to iterative numerical procedures. Optimization of continuous non-linear functions is a common issue, widely studied and many algorithms and techniques have been proposed to solve it efficiently. Most techniques involve choosing some initial value $\mathbf{w}^{(0)}$ for the weight vector and then using the gradient information to redefine the new vector $\mathbf{w}^{(\tau+1)}$, where τ labels the iteration step.

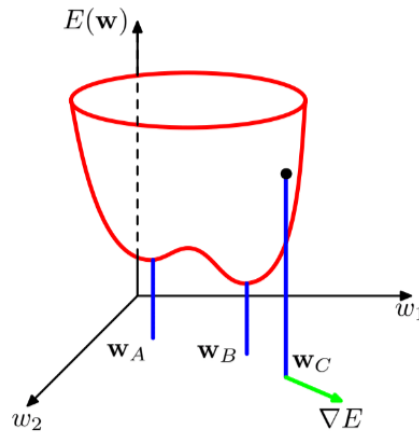


Figure 3.13: Geometrical view of the error function as a surface sitting over weight space

Error Backpropagation

Error backpropagation is an efficient technique for evaluating the gradient of and error function $E(\mathbf{w})$ for a feed-forward neural network and therefore transfer the information forwards and backwards through the network. Most training algorithms involve an iterative procedure for minimization of an error function, with adjustments to the weights being made in a sequence of steps. At each such step, we can distinguish between two distinct stages. In the first stage, the derivatives of the error function with respect to the weights must be evaluated. In the second stage, the derivatives are then used to compute the adjustments to be made to the weights. The backpropagation procedure can therefore be summarized as follows:

- Apply an input vector \mathbf{x}_n to the network and forward propagate through the network and calculate the activations of all the hidden and output units
- Evaluate the δ_k for all the output units:

$$\delta_k = y_k - t_k \quad (3.25)$$

- Backpropagate the δ 's to obtain δ_j for each hidden unit of the network using the formula:

$$\delta_j = h'(\alpha_j) \sum_k w_{kj} \delta_k \quad (3.26)$$

- Evaluate the required derivatives using:

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i \quad (3.27)$$

where $\delta_j \equiv \frac{\partial E_n}{\partial a_j}$ and z_i the activation of a unit

3.2.3 Optimizers

Gradient Decent

Gradient Descent Algorithm iteratively calculates the next point using gradient at the current position, scales it by a learning rate η and subtracts obtained value from the current position. It subtracts the value because we want to minimize the function $f(x)$. This process can be written as:

$$w^{(k+1)} = w^{(k)} - \eta \nabla_w f(w^{(k)}) \quad (3.28)$$

where w the model's parameters at step k .

The learning rate η is the most crucial parameter because it determines how quickly the model is adapted to the problem. Choosing the right learning rate is not an easy task as it affects the training process in many ways. Smaller learning rates require more training epochs given the smaller changes made to the weights on each update, whereas larger learning rates result in rapid changes and require fewer training epochs. In addition, a too large learning rate can cause the model to converge too quickly to a suboptimal solution, whereas a learning rate that is too small can cause the process to get stuck as shown in [Figure 3.14](#).

<https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21>

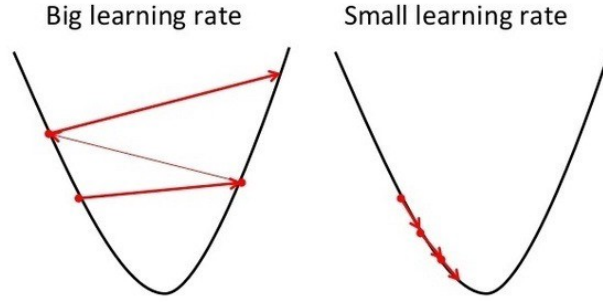


Figure 3.14: A small learning rate makes the model converge slowly to the global minimum loss.

AdaGrad

The AdaGrad algorithm derives from Adaptive Gradient algorithm and was developed by J.Duchi, D.Hazan and Y.Singer [22]. It is an improved algorithm based on gradient descent which individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all of their historical squared values. This method can be described by two basic steps:

- Calculation of the sum of the squares of the past gradients.

$$g^{(k+1)} = g^{(k)} + \nabla_w f(w^{(k)})$$

- Calculation of the new model's parameters.

$$w^{(k+1)} = w^{(k)} - \frac{\eta \nabla_w f(w^{(k)})}{\sqrt{g} + \varepsilon}$$

where ε is the vector of small numbers to avoid dividing by zero.

The parameters with the largest partial derivative of the loss have a correspondingly rapid decrease in their learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate. The net effect is greater progress in the more gently sloped directions of parameter space. In the context of convex optimization, the AdaGrad algorithm enjoys some desirable theoretical properties. However, empirically it has been found that-for training deep neural network models-the accumulation of squared gradients from the beginning of training can result in a premature and excessive decrease in the effective learning rate.

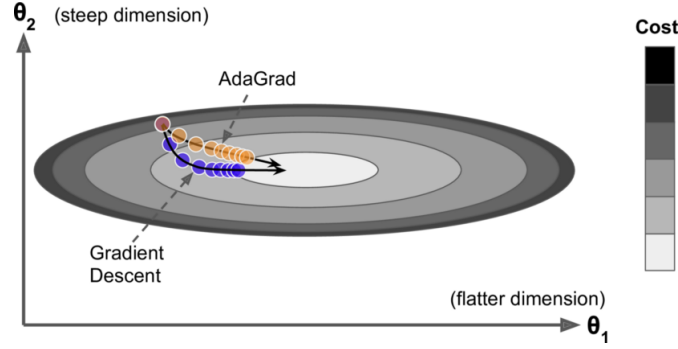


Figure 3.15: AdaGrad vs Gradient Descent: the former can correct its direction earlier to point to the optimum

RMSProp

The RMSProp algorithm [23] modifies AdaGrad to perform better in the non-convex setting by changing the gradient accumulation into an exponentially weighted moving average.

AdaGrad operates by shrinking the learning rate according to the entire history of the squared gradient and may have made the learning rate too small before arriving at such a convex structure. AdaGrad is designed to converge rapidly when applied to a convex function. When applied to a non-convex function to train a neural network, the learning trajectory may pass through many different structures and eventually arrive at a region that is a locally convex bowl.

On the other hand, RMSProp uses an exponentially decay average v_t^w to discard history from the extreme past so that it can converge rapidly after finding a convex bowl. Moreover, it takes away the need to adjust the learning rate, and does it automatically and a different learning rate for each parameter w_{t+1} .

In RMS prop, each update is done according to the equations described below. This update is done separately for each parameter:

$$v_t^w = \beta * v_{t-1}^w + (1 - \beta)(\nabla w_t)^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t^w + \epsilon}} * \nabla w_t$$

where β is the momentum.

Adam

Adaptive Moment Estimation (Adam) introduced by Diederik P. Kingma, Jimmy Ba in 2014 [24] and is another optimization algorithm that provides more efficient neural network weights.

In addition to storing an exponentially decaying average of past squared gradients v_t like RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t , similar to momentum. Whereas momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction, which thus prefers flat minima

in the error surface [25]. We compute the decaying averages of past and past squared gradients and respectively as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

where m_t and v_t are the estimated values of the first and second moment of gradients respectively. As m_t and v_t are initialized as are initialized with zero vectors at the first iteration, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small. They counteract these biases by computing bias-corrected first and second moment estimates:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Building upon the strengths of previous models, Adam optimizer gives much higher performance than the previously used and outperforms them by a big margin into giving an optimized gradient descent. The plot is shown below clearly depicts how Adam Optimizer outperforms the rest of the optimizer by a considerable margin in terms of training cost (low) and performance (high) **Figure 3.16**

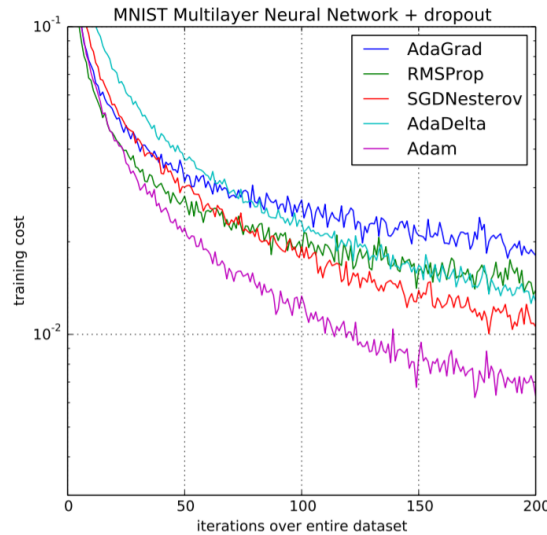


Figure 3.16: Performance Comparison on Training cost using different optimizer

3.3 Deep learning and Convolutional Neural Networks

In the previous section we discussed the basic features of a simple neural network model. Deep Neural Networks are complex structures based on neurons and their architecture differs according to the type of the task. In addition the existent dataset has to be clarified as it will determine which of the following learning approaches to be applied.

- Supervised learning is a machine learning approach that's defined by its use of labeled datasets. These datasets are designed to train or "supervise" algorithms into classifying data or predicting outcomes accurately. Using labeled inputs and outputs, the model can measure its accuracy and learn over time.
- Unsupervised learning in general uses machine learning algorithms to analyze and cluster unlabeled data sets. These algorithms discover hidden patterns in data

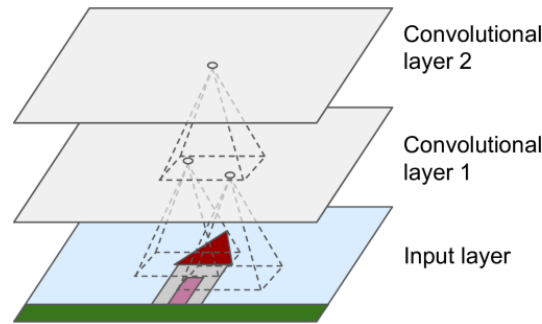


Figure 3.17: CNN layers with rectangular local receptive fields

without the need for human intervention and they can be used in tasks such as clustering, denoising and dimensionality reduction.

The main distinction between the two approaches is the use of labeled datasets. To put it simply, supervised learning uses labeled input and output data, while an unsupervised learning algorithm does not. Unsupervised learning models are used for three main tasks: clustering, association and dimensionality reduction.

3.3.1 Convolution Layer

Convolution layers are the fundamental building blocks of the Convolutional Neural Network [26]. Their interconnection differs significantly in contrast to a dense layer.

The [Figure 3.17](#) illustrates the basic sequence of convolutional layers. Neurons in the first convolutional layer are not connected to every single pixel in the input image but only to pixels in their receptive fields. In addition, each neuron in the second convolutional layer is connected only to neurons located within a small rectangle in the first layer. This architecture allows the network to concentrate on small low-level features in the first hidden layer, then assemble them into larger higher-level features in the next hidden layer, and so on.

A neuron located in row i , column j of a given layer is connected to the outputs of the neurons in the previous layer located in rows i to $i + f_h - 1$, columns j to $j + f_w - 1$, where f_h and f_w are the height and the width of the receptive field. In order for a layer to have the same height and width as the previous layer, a boarder of zeros can be added as shown in [3.18](#)

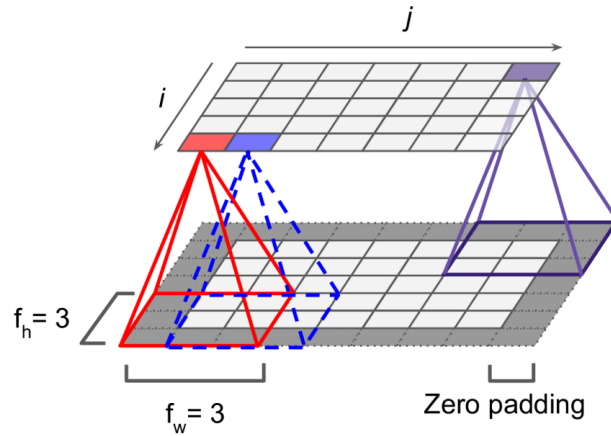


Figure 3.18: Connections between layers and zero padding

In some cases, it is preferable to reduce the size of the layer so that the computational power needed to be reduced accordingly. This can be achieved by spacing out the receptive fields, as shown in 3.19. The shift from one receptive field to the next is called the stride.

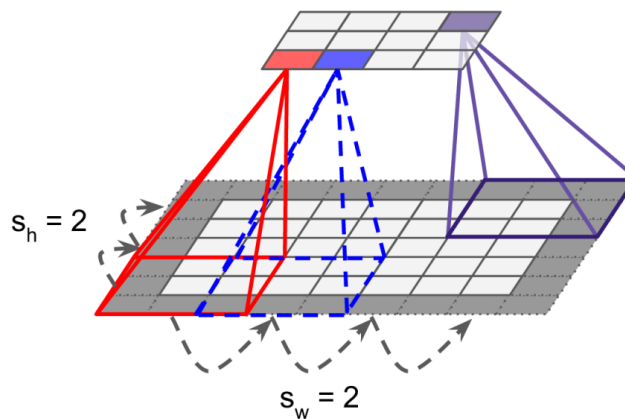


Figure 3.19: Dimensionality reduction using a stride of 2

A commonly used convolutional layer has multiple filters, and it outputs one feature map per filter, so it is more accurately represented in 3D 3.20. To do so, it has one neuron per pixel in each feature map, and all neurons within a given feature map share the same parameters (same weights and biases). However, neurons in different feature maps use different parameters. A neuron's receptive field is the same as described earlier, but it extends across all the previous layers' feature maps. In other words, a convolutional layer simultaneously applies multiple trainable filters to its inputs, making it capable of detecting multiple features anywhere in its inputs.

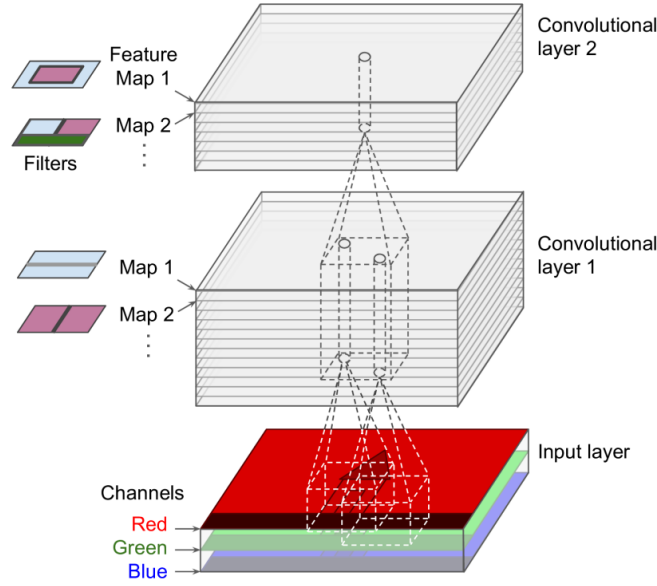


Figure 3.20: Convolution layers with multiple feature maps

The equation computing the output of a neuron in a feature map k of a convolutional layer is:

$$z_{i,j,k} = b_k \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f'_n-1} x_{i',j',k'} w_{u,v,k',k} \quad (3.29)$$

where:

- $z_{i,j,k}$ is the output of the neuron located in row i , column j in feature map k of the convolutional layer l
- s_h and s_w are the vertical and horizontal strides, f_h and f_w are the height and width of the receptive field and f'_n is the number of feature maps in the previous layer
- $x_{i',j',k'}$ is the output neuron located in previous layer
- $w_{u,v,k',k}$ is the connection weight between any neuron in feature map k of the layer and its input located at row u , column v (relative to the neuron's receptive field) and feature map k' .

3.3.2 Pooling

Pooling is one of the Convolution Networks distinctive concepts as it reduces the dimensionality of feature maps and eliminates noisy and unnecessary convolutions [27]. Using pooling layers the parameters of the consecutive model are radically reduced (Figure 3.21) as well as the computational power needed and the Neural network becomes less prone to overfitting. There are multiple types of max pooling like Max, Sum and Average which can be implemented using kernels whose size and stride can vary.

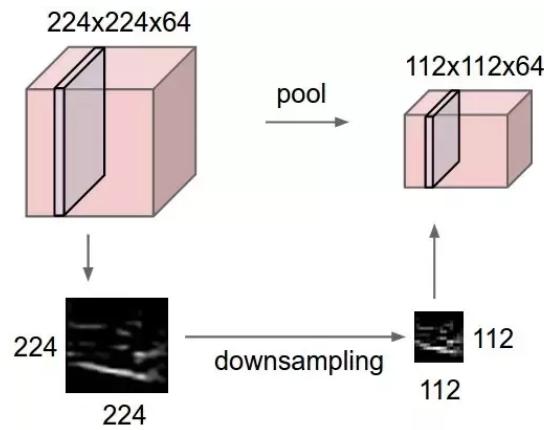


Figure 3.21: Pooling Layer

Max pooling layer is the most widely used filter as it provides better performance with sparse coding and simple linear classifiers. It operates by defining a spatial neighborhood and getting the maximum unit of the feature map. An advantage of max pooling is its noise-suppression properties, as it works on discarding those activations which contain noisy activation.

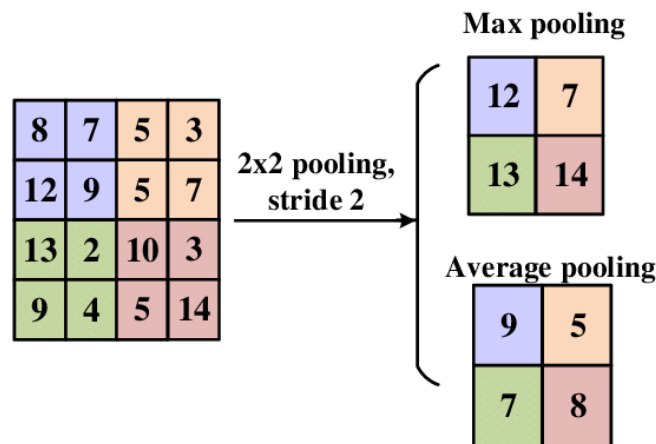


Figure 3.22: Example of 2x2 and stride 2 Max and Average Pooling

On the other hand, a major downside of max pooling is that considers only the maximum element from the pooling area and ignores other elements. For example in [Figure 3.22](#) when a 2x2 and stride 2 max pooling layer is applied to a feature map passes forward only the 25% of the existing information. If the majority of the elements in the pooling area would be of high magnitudes, the discerning features get disappeared after performing max pooling operation.

3.3.3 Transposed Convolution and Deconvolution

In image segmentation tasks transposed convolutional layers are normally used for up-sampling in the decoding phase (Section 3.3.7) and generate an output feature map that has a spatial dimension greater than that of the input feature map [28]. The parameters that characterize a transposed convolutional operation are kernel size, padding and

stride like the standard convolution. These values of padding and stride are the ones that hypothetically were carried out on the output to generate the input dimensions. For instance, if the output is passed through standard convolution with stride and padding defined, it will generate the spatial dimension the same as that of the input.

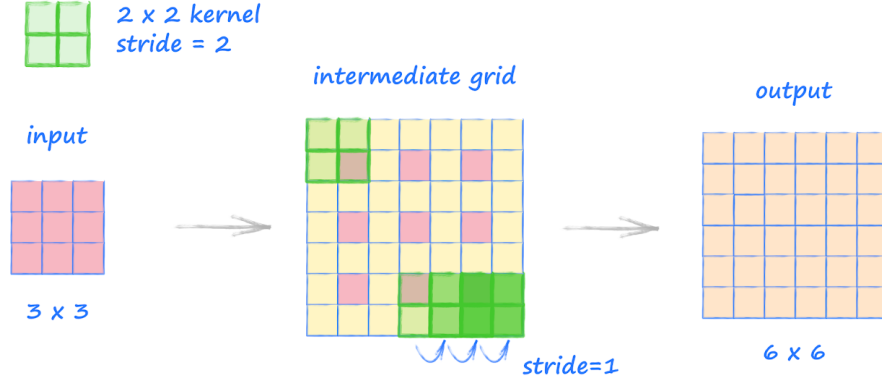


Figure 3.23: Transpose Convolution by 2x2 kernel and stride 2

Deconvolution is a mathematical operation that reverses the effect of convolution and should not be confused with transpose convolution [29]. Let us assume that we pass an image through a convolutional layer and collect the output. If we pass the output through a deconvolutional layer, we get back the exact same input.

3.3.4 Batch normalization

Batch Normalization (BN) [30] is an algorithmic method which makes the training of Deep Neural Networks faster and more stable. During each iteration, the network computes the mean μ_B and the variance σ_B corresponding to each value x of mini-batch $B = \{x_1 \dots x_m\}$ (Eq. 3.30). Then normalization is applied to x (Eq. 3.31) and the final output y_i is calculated by applying a linear transformation with the two learnable parameters γ, β (Eq. 3.32).

$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i \quad \sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2 \quad (3.30)$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (3.31)$$

$$y_i = \gamma \hat{x}_i + \beta \quad (3.32)$$

where ϵ is a constant added for numerical stability.

The batch normalization according to S.Ioffe and C.Szegedy [30] should be added after the convolution layer and before the activation function.

3.3.5 Dropout

Dropout is a regularization technique and was first introduced by Nitish Srivastava, Geoffrey Hinton et al. in 2014 [31]. It is developed to overcome the serious problem of overfitting on the training set in deep neural networks by randomly dropping out a percentage p of units from hidden and visible layers, as shown in Figures 3.24. In other

words, it adds noise to the network layers in order to prevent the model from learning the complex relationship between input and output values of the training set, which would lead to poor performance on new data (testing set).

A common value of p is 0.5 which seems to produce the best results in most applications, but it could be chosen and tested in range of $[0.5, 0.8]$.

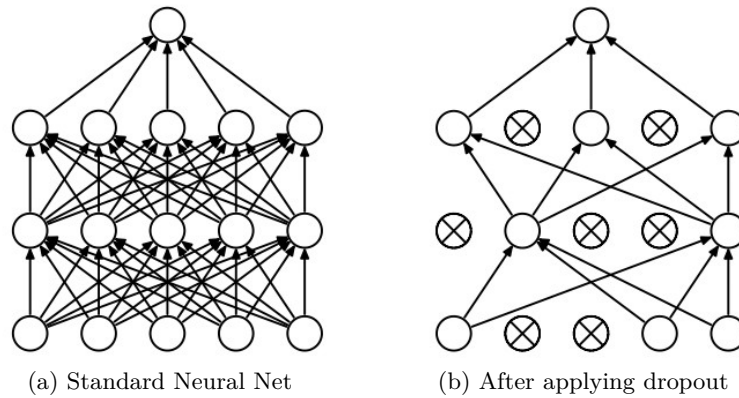


Figure 3.24: Visualisation of Dropout

3.3.6 VGGNet

VGG comes from Visual Geometry Group from Oxford. The architecture, shown in [Figure 3.25](#), was developed by Karen Simonyan and Andrew Zisserman of the University of Oxford at the ILSVRC competition in 2014, where it took the 2nd place [\[32\]](#). The model consists of blocks, where each block is composed of a few 2D Convolution layers followed by a Max Pooling layer, where its innovation was the use of small filters 3×3 instead of large-size filters (such as 11×11 , 7×7). The model supports 16 or 19 convolution layers (VGG-16 / VGG-19) all equipped with the rectification non-linearity (ReLU), which contains more than 138 million parameters and needs approx 550 MB disk space. The original VVGNet was trained for 2-3 weeks on a computer system equipped with four NVIDIA Titan Black GPUs. Consequently, it is not feasible for an individual to build and train it from scratch. However, it is widely used by the community as an image feature extractor through transfer learning techniques which will be discussed later.

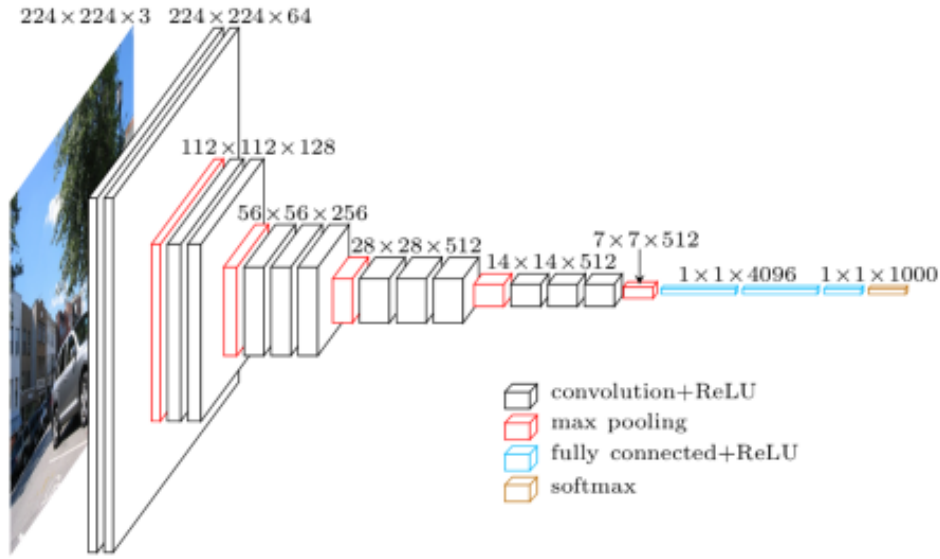


Figure 3.25: VGG16 Architecture

3.3.7 Image segmentation using Deep Neural Networks

Image segmentation can be formulated as a classification problem of pixels with semantic labels (semantic segmentation) or partitioning of individual objects (instance segmentation)[33]. Semantic segmentation performs pixel-level labeling with a set of object categories (e.g., human, car, tree, sky) for all image pixels, thus it is generally a harder undertaking than image classification, which predicts a single label for the entire image. Instance segmentation extends semantic segmentation scope further by detecting and delineating each object of interest in the image as shown in Figure 3.26. In this thesis, the Semantic Segmentation technique is applied for weld detection from an image.

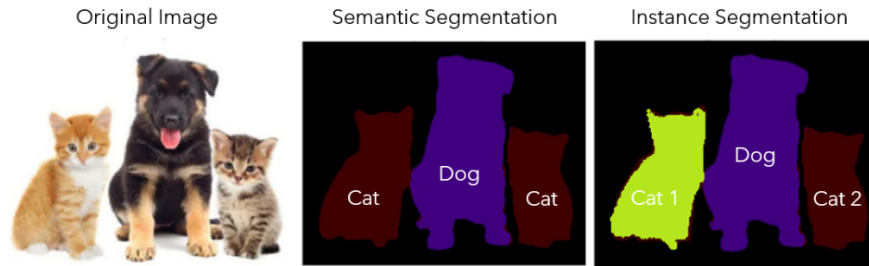


Figure 3.26: Semantic Segmentation vs Instance Segmentation

A Popular family of deep models for image segmentation is based on the convolutional encoder-decoder architecture. It was initially developed for medical image segmentation, such as UNet [34] but now also is being used for other applications. The UNet is a Fully Convolutional Network (Figure 3.27) and was developed in 2015 by O.Ronneberger, P.Fischer, and T.Brox and consists of convolutional, max pooling and transposed convolutional layers, whose function discussed in previous subsection.

A key-feature of UNet's architecture is the skip-connection operation, depicted with gray arrow and mentioned as "copy and crop", in Figure 3.27. During encoding phase the image is downsampled and lot of information is lost which would be useful for a

detailing upsampling. Thus, the developers connected and concatenated the fine-grained details learned in the encoder with layers in the decoder part.

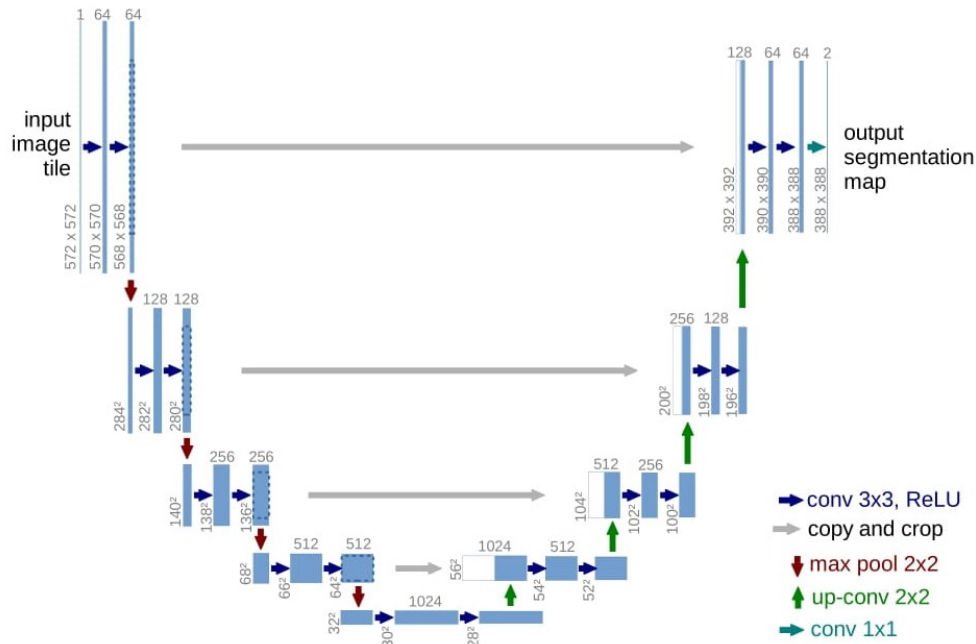


Figure 3.27: U-net architecture

3.3.8 Transfer Learning

The goal of transfer learning is to improve learning in the target task by leveraging knowledge from previous tasks [35]. There are three common measures by which transfer might improve learning, as shown in Figure 3.28. First is the initial performance achievable in the target task using only the transferred knowledge, before any further learning is done, compared to the initial performance of an ignorant agent. Second is the amount of time it takes to fully learn the target task given the transferred knowledge compared to the amount of time to learn it from scratch. Third is the final performance level achievable in the target task compared to the final level without transfer.

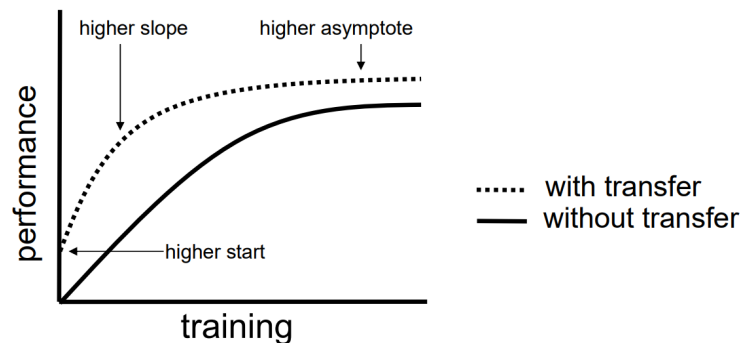


Figure 3.28: Three ways in which transfer might improve learning

In computer vision, where deep neural networks consist of millions of parameters, is not feasible for an individual to train a model from scratch. In classification, object detection and segmentation tasks the idea of transfer learning is to utilize one or more layers of an existing deep neural model without its final layer(classifier). These models have been trained in large datasets achieving high accuracy. Consequently, their convolutional layers parameters have been adjusted to extract the important characteristics of each image. The pre-trained model's convolutional weighted layers function as universal feature extractor and in transfer learning usage won't be updated with the unseen data, during back propagation.

There are three strategies that can be followed for the transfer learning implementation:

- Retrain only the last layers or the hole classifier of the model.
- Freeze part of the convolutional base and train the rest model. Following this strategy, we take advantage of the ability of lower layers to detect general features and higher layers more specific.
- Train all model's parameters. In that way, the old information is discarded and the model needs a large dataset to be retrained.

In image segmentation tasks only the last two strategies are applicable and due to the lack of dense-layer classifier.

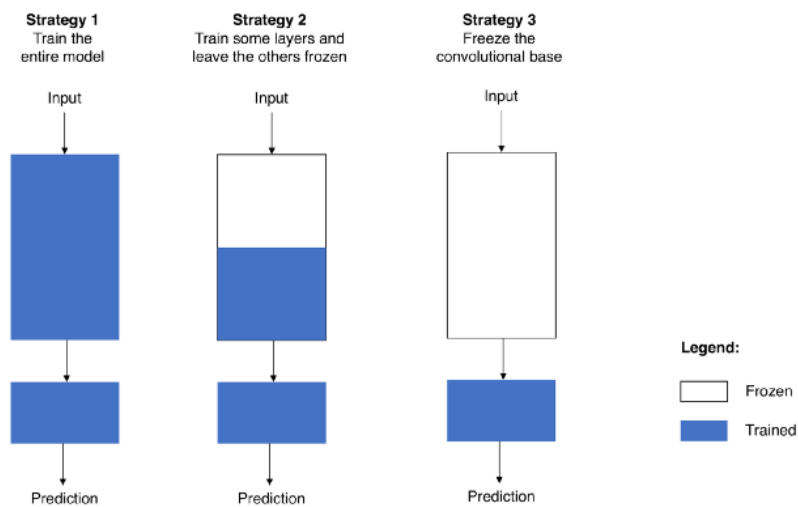


Figure 3.29: Transfer learning strategies

Chapter 4

Welding Datasets

This chapter presents the images we want to segment and how they were obtained. It also describes the segmentation masks and how they were created.

4.1 Images

The target of this thesis is to develop a universal model for weld detection. Each welding joint has a different pattern, shape and color depending on the welder's experience or the welding robot's accuracy and the environmental conditions to which it is exposed. Images with welding joints were collected from 3 sources to achieve greater diversity and to increase the model's robustness.

NTUA

The initial dataset was created using photos provided by the Marine Structures Shipbuilding Technology Laboratory (MSSTL) at the School of Naval Architecture and Marine Engineering of NTUA. The photos include welds created using the Shielded Metal Arc Welding (SMAW) or the Flux Cored Arc Welding (FCAW) method. The FCAW procedure was automated with the use of a welding robot ([Figure 4.1](#)) with 6 degrees of freedom, manufactured by IGM® Robotersysteme.

Photos from Internet

A large number of photos were collected from the Internet. In that way, different types of welding such as butt joint, corner joint were added to the training set. Intermittent welds and welds on circular tubes also used to enrich the dataset and enhance its ability to detect non-straight line and non-continuous weld seams.

Shipyard

The majority of the photos which depict welds on ship's steel joins were taken during periodical surveys or after a repair procedure. A part of them was provided to the school of Naval Architecture and Marine Engineering of National Technical University of Athens from Mr. Theodoros Panagopoulos, surveyor at Hellenic Coast Guard and the M&C Group and others were found on Shipbuilding Kapogiannatos website. Images with painted welds were not preferred as they include limited information and the weld outline sometimes can not be distinguished. Moreover, the visual or the NDtest is conducted prior to coating.

Some large-scaled photos were cropped into smaller ones to minimize the probability of the weld seam's distinctive characteristics ("information") to be vanished during down-scaling in the preprocessing phase. The final data-set consist of 300 RGB image taken with a smartphone with 24 bit depth sRGB color and are jpg formatted

Table 4.1: Images Dataset

Dataset	Number of Images
NTUA lab	54
Various Internet	148
Shipyard	98
Total	300



Figure 4.1: Robotic arm for automatic welding, MSST Lab of NTUA



Figure 4.2: Samples of weld form NTUA LAB



Figure 4.3: Samples of Images found on internet



Figure 4.4: Samples of Images from weld on ship

4.2 Image Annotation

Image annotation plays a significant role in computer vision, the technology that allows computers to gain high-level understanding from digital images or videos and to see and interpret visual information just like humans.

In this thesis, the *welding segmentation* can be interpreted as a one class-segmentation problem where the welding (white) and background (black) pixels constitute a binary mask. It is obvious that the more accurate the image annotations, the better results the model produces.

There are two primary data annotation methods; human annotation and automated annotation. Human annotation typically takes longer and costs more than automated annotation, and also requires training for annotators, but achieves more accurate results. In comparison, automated annotation is more cost-effective but it can be difficult to determine the accuracy level of the results.

During this study due to the relatively small dataset, the image annotation was created manually using two software.

The first is GIMP [36], a free and open source Image editor, which allows users to create and stack different layers on top of the initial RGB image. Labeling in such a program like GIMP is a time-consuming procedure because the welds' borders have to be defined laboriously by hand using a polyline. Therefore, GIMP was used only for the creation of a few masks for the initial testing of the neural network model.

For the rest of the annotation process, Segments.ai [37] was used. Segments.ai is a professional Superpixel tool for image segmentation that boosts labeling efficiency significantly. The platform by utilizing AI and ML algorithms, segments automatically the image into regions which may constitute candidate objects to be labeled.

Even with the usage of a specialized software like Segments.ai, the image annotation was not an easy task. During the welding procedure heat tints occur when the stainless steel is exposed to oxygen. In addition, a weld created by an amateur welder can appear defects to the joint such as spatter, lack of fusion, misalignment, underfill and concavity. Furthermore, an old weld, a painted or a corroded one displays color defects as, shown in [Figure 4.6](#). Overall, these geometrical imperfections cause a complex and wide borderline which is not feasible to be precisely delineated.

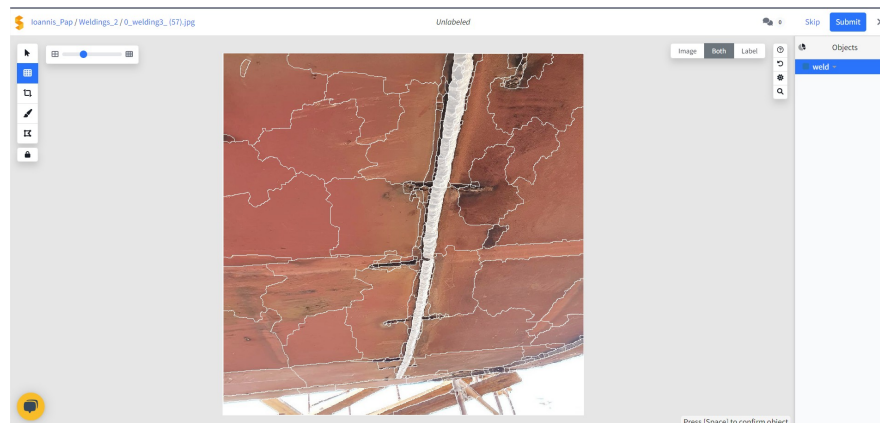


Figure 4.5: Segments.ai: The software segments the image and the weld is marked(white)



Figure 4.6: Segments.ai: Sample of quite difficult circumference to be delineated

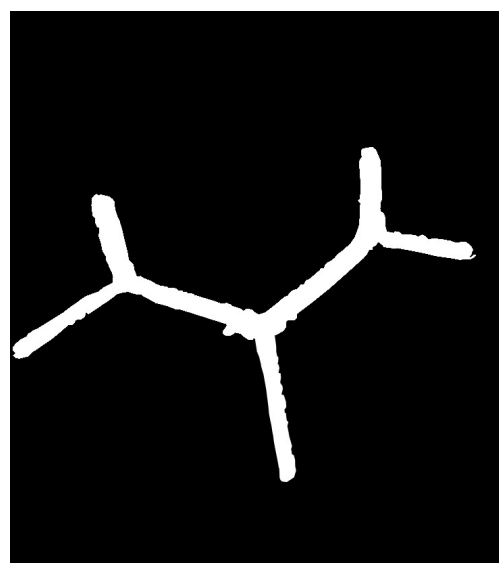


Figure 4.7: RGB Image and its Binary Mask

Chapter 5

Methodology

5.1 Hardware and software used

All preprocessing of images and model training were conducted in Julia environment [38] using an Intel Core i7-7700HQ @ 3.50GHz processor and a NVIDIA GTX mobile 1060 6GB GDDR5 graphics processing unit (GPU). The model itself was built using and modifying the UNet.jl [39] library which is built on top of Flux.jl. [40] [41]

Flux is a library for machine learning geared towards high-performance production pipelines. It comes "batteries-included" with many useful tools built in, but also lets you use the full power of the Julia language where you need it.

Flux integrates with high-performance automatic differentiation (AD) tools such as Zygote.jl for generating fast code. Flux optimizes both CPU and GPU performance and it also supports GPU accelerated computation. This provides a major performance boost compared to training the model on a CPU, see Table 5.1.

Table 5.1: Comparison of CPU and GPU

Specifications	Intel Core i7-7700HQ Processor	NVIDIA GeForce GTX 1060 Mobile
Base Clock Frequency (Boost)	2.8 GHz (3.8 GHz)	1.5 GHz (1.6GHz)
Processing units	8	1280
Memory Bandwidth	37.5 GB/s	192.2 GB/s
Floating-Point Calculations	166.8 GFLOPS	4275
Cost (Launched price)	~\$ 380	~\$ 240

5.2 Data Preparation and Augmentation

The initial RGB large images, many of them with a resolution over 3000 x 3000 cropped and downscaled to 224 x 224 which is the default size of VGG16 input. Afterwards, they imported in Julia and the initial RGB value range of [0,255] was normalized to [0,1]. In addition, the 3-channel image array was further normalized according to VGG16 requirements using the following algorithm:

Algorithm 1 VGG normalization algorithm

$$\begin{aligned}
mean &\leftarrow [0.485, 0.456, 0.406] \\
std &\leftarrow [0.229, 0.224, 0.225] \\
image_{norm} &= (image_{initial} - mean) / std
\end{aligned}$$

Due to the limited size of the dataset the technique of data augmentation was applied on testing images to multiply the dataset and to make the model more robust. On each image was randomly applied a combination of the transformations listed below:

- Rotation of $p/2rad$ or Flip across X or Flip across Y
- Color Jitter or Gaussian blur
- Elastic Distortion

where Color Jitter is an operation that adjusts both contrast and brightness (random values were used for each image).

The [Figure 5.1](#) illustrates the different possible color and shape transformations to be applied. Visual testing to the augmented data has to be done to ensure that the transformations have not vanished the important information and to check if the images can actually exist and represent the same concept of the class.

Elastic Distortion plays a significant role by changing the shape of the weld, straight welds are becoming curved. This technique increases the capability of the model to recognize welds with various shapes.

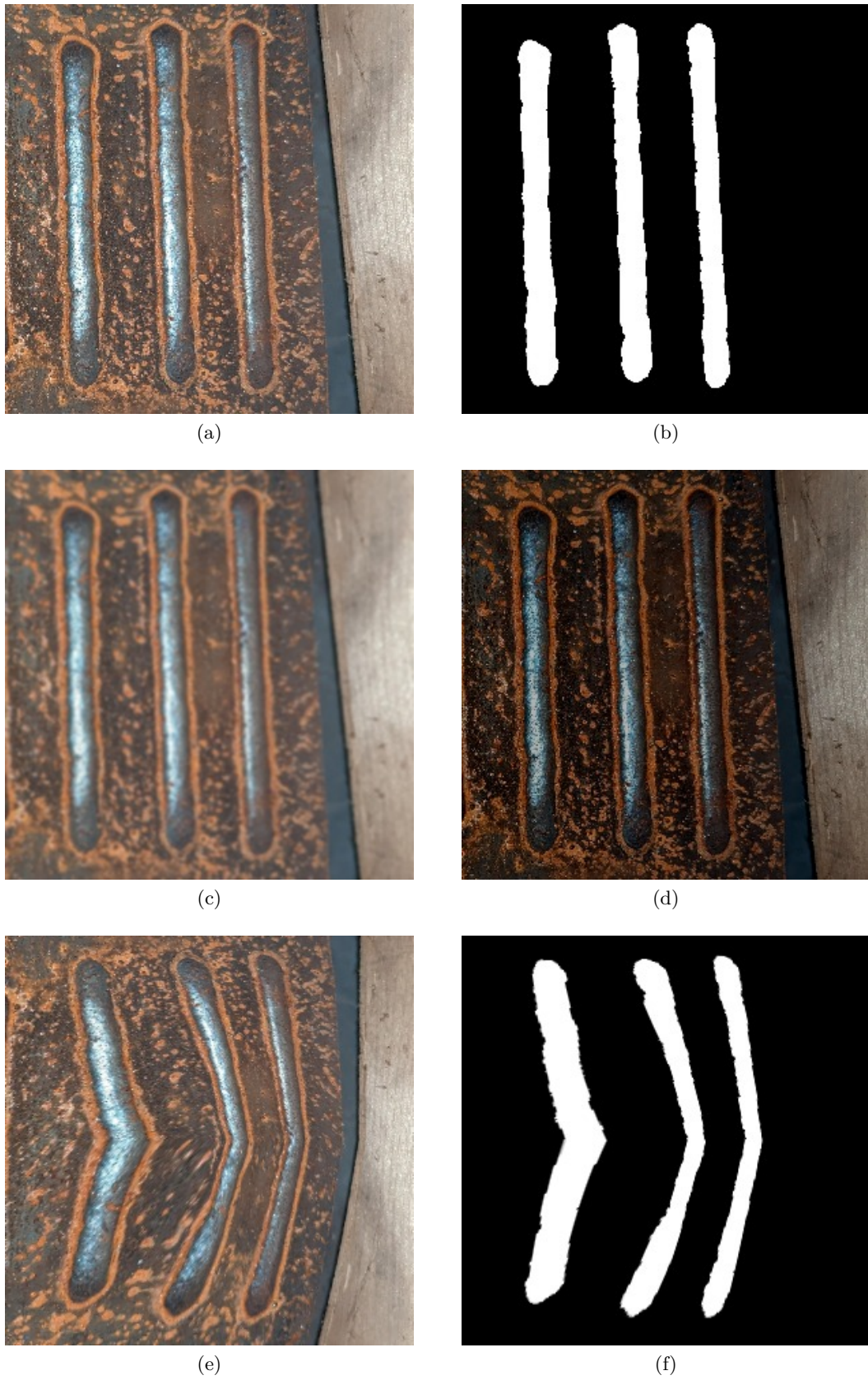


Figure 5.1: Samples of Augmentation techniques: (a)Initial Image; (b)Initial Mask; (c)Gaussian Blur (d)Color Jitter (e)Elastic Distortion + Flip Y; (f)Mask of (e)

5.3 Model architecture

The model architecture was based on the UNet architecture proposed in [42]. DhairyaL Gandhi has developed a generic UNet implementation written in Julia, on top of Flux.jl, the library UNet.jl. The source code of the library was modified to create the model's architecture, as shown in Figure 5.2.

In the proposed model, asymmetric down and up sampling paths are utilized to design the encoder-decoder structure.

The encoder structure consists of VGG16 convolutional core, as the transfer learning technique is adopted and VGG16 is employed as backbone model. VGG16 is trained on ImageNet dataset, a collection of over 14 million images belonging to 22,000 categories, achieving 92.7% accuracy.

The decoder has fewer layers than the encoder creating an weight-imbalance at first glance but according to literature, this architecture can also achieve high accuracy which is boosted further by the utilization of Batch Normalization and Dropout operations [43]. The output of the decoder is a grayscale image with values in range of (0,1), which represents the probability of a pixel to be weld or background. Prior to the output layer, a 1×1 convolutional operation occurs, followed by a sigmoid activation function. The 1×1 filter is simple as it does not involve any neighboring pixels in the input. Thus, it may not be considered a convolutional operation. Instead, it is a linear weighting or projection of the input.

The skip-connection operations were preserved to provide a positive contribution to the detailed reconstruction of the image through the decoder.

This architecture is used to reduce the computational cost without main data loss in order to build a fast and accurate model.

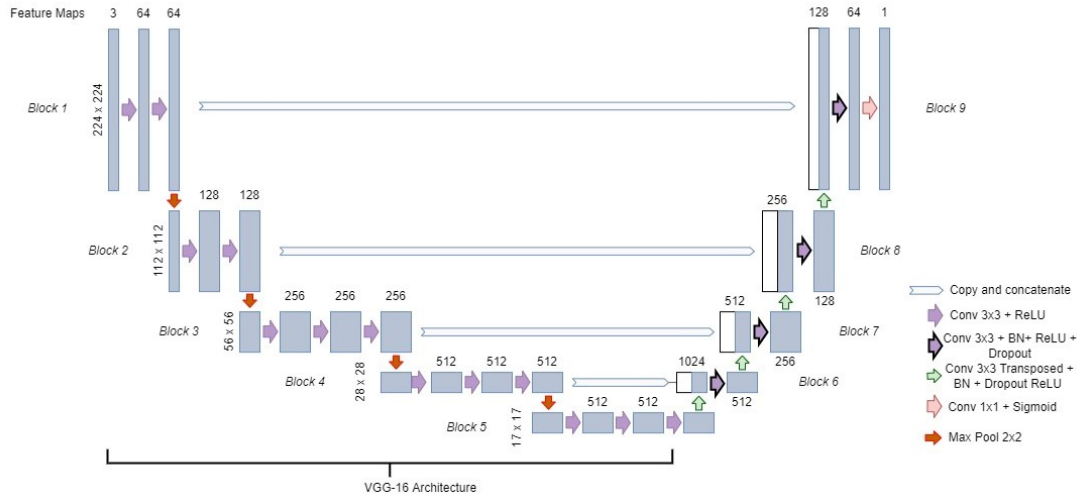


Figure 5.2: Architecture of UNet-VGG16 with transfer learning

5.4 Loss function & Optimizer

The Binary Cross Entropy (BCE) was used as the loss function during the training phase. BCE measures the difference between two probability distributions for a given random variable or set of events. It is widely used for classification objectives and

it performs well in image segmentation tasks. In many studies has been found that the predictions from models trained with cross-entropy loss were of high quality, those produced by models trained with the Dice loss appeared visually cleaner since they were almost binary [44] [45].

The Binary Cross Entropy is defined by:

$$L_{BCE}(y, \hat{y}) = -(y \log(\hat{y} + \epsilon) + (1 - y) \log(1 - \hat{y} + \epsilon)) \quad (5.1)$$

where \hat{y} the predicted value and y the ground truth.

The optimizer used in this thesis is Adaptive Moment Estimator (Adam). Adam as discussed in previous section has been proven to outperform concerning the training cost and the final accuracy of the model. The initial learning rate is usually chosen arbitrarily and is adjusted manually by monitoring training performance each time.

5.5 Evaluation of network performance

Evaluation of semantic segmentation can be quite complex because it is required to measure classification accuracy as well as localization correctness. The aim is to score the similarity between the predicted (prediction) and the annotated segmentation mask (ground truth). The predicted binary mask contains pixel values of 0 or 1. Each pixel in the predicted binary mask is compared with the corresponding pixel in the ground truth mask and afterwards it is classified to one of the following categories:

- True Positive (TP): pixel considered as being in the object and being really in the object
- False Positive (FP): pixel considered by the segmentation in the object, but which in reality are not part of it
- True Negative (TN): pixel outside the object both in the segmentation and the ground truth
- False Negative (FN): pixel of the object that the segmentation has classified outside

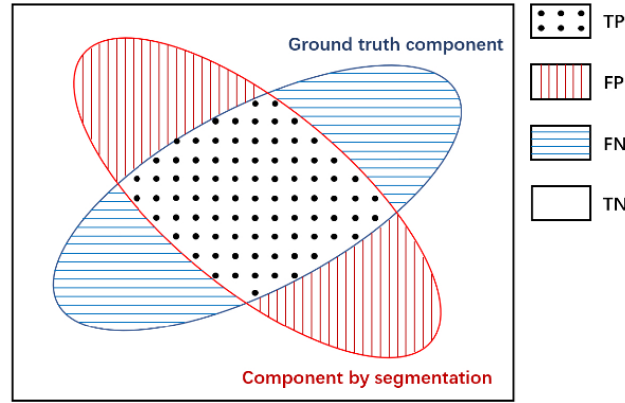


Figure 5.3: Illustration of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) of the segmentation in a binary case. The blue and red ellipses represent the ground truth component and component recognized by the segmentation method.

The efficiency and the effectiveness of the model during training can be measured in various ways:

- Pixel accuracy is the easiest and most obvious index to be calculated. It is the percentage of pixels in your image that are classified correctly. However, it might be misleading in cases where the objects are tiny.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (5.2)$$

- The Intersection-Over-Union (IoU), also known as the Jaccard Index, is one of the most commonly used metrics in semantic segmentation as it is a very straightforward metric that's extremely effective. This metric ranges from 0–1 (0–100%) with 0 signifying no overlap and 1 signifying perfectly overlapping segmentation

$$IOU = \frac{TP}{TP + FP + FN} = \frac{A \cap B}{A \cup B} \quad (5.3)$$

- Precision effectively describes the purity of our positive detections relative to the ground truth

$$Precision = \frac{TP}{TP + FP} \quad (5.4)$$

- Recall effectively describes the completeness of our positive predictions relative to the ground truth

$$Recall = \frac{TP}{TP + FN} \quad (5.5)$$

- False Positive Rate effectively describes the image noise of false positive predictions relative to background

$$FalsePositiveRate = \frac{FP}{TN + FP} \quad (5.6)$$

- Dice Coefficient (F1 Score) is a measure combining both precision and recall

$$F1\text{-score} = \frac{Precision * Recall}{Precision + Recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \quad (5.7)$$

5.6 VGG16 Feature Extraction Testing

VGG16 was chosen apriori to be used for the transfer learning strategy as it is a network tested among the community in various types of applications. Even though VGG16 is trained in thousands of images and categories of them, we can not be 100% sure that its filters work for our type of images as feature extractor. For that reason, it conducted a visual testing to the feature maps exported on each block 1-5 ([Figure 5.2](#)). It can be seen from the images below that the VGG16's neurons respond to this type of input images and the information of the weld is traveling through the convolutional core.



Figure 5.4: A sample image tested

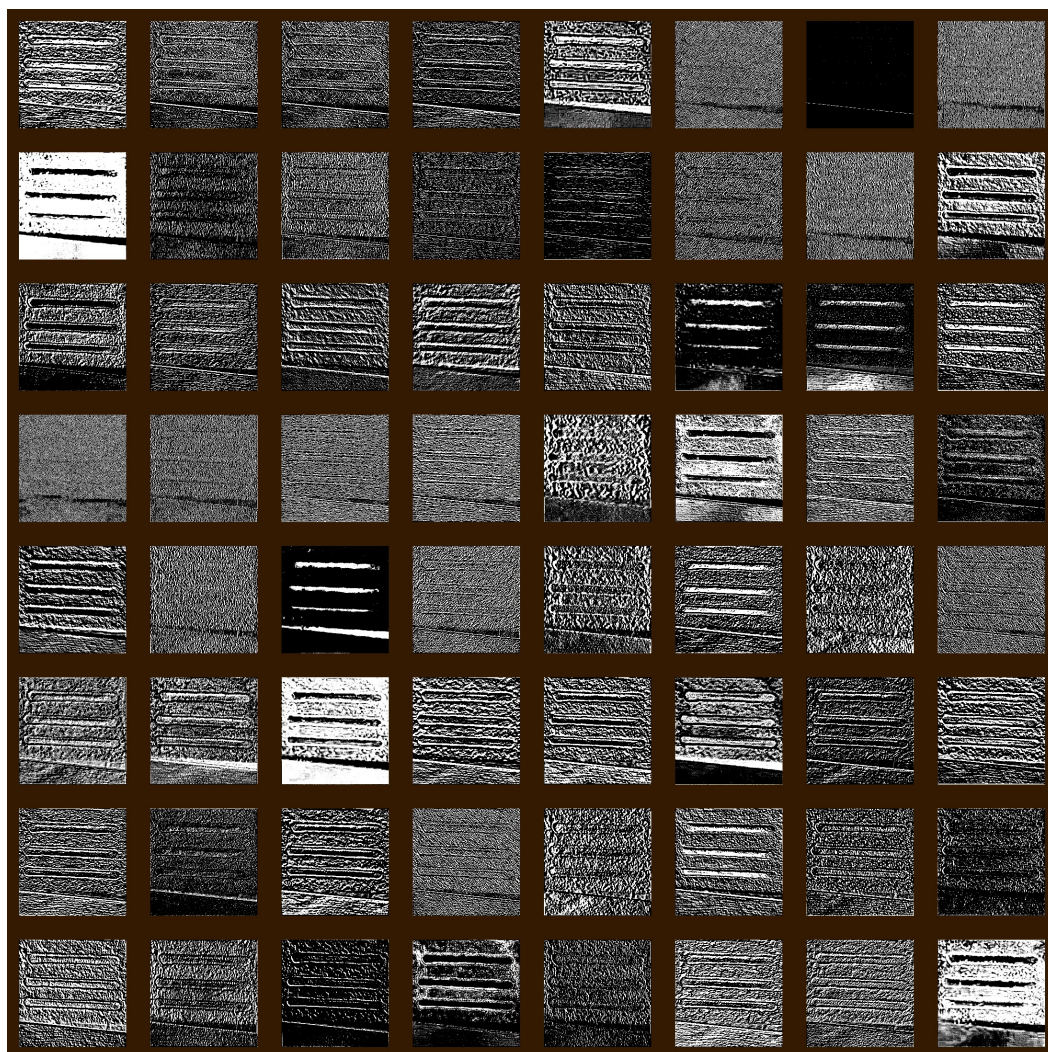


Figure 5.5: Feature maps exported on Block 1, size 224x224

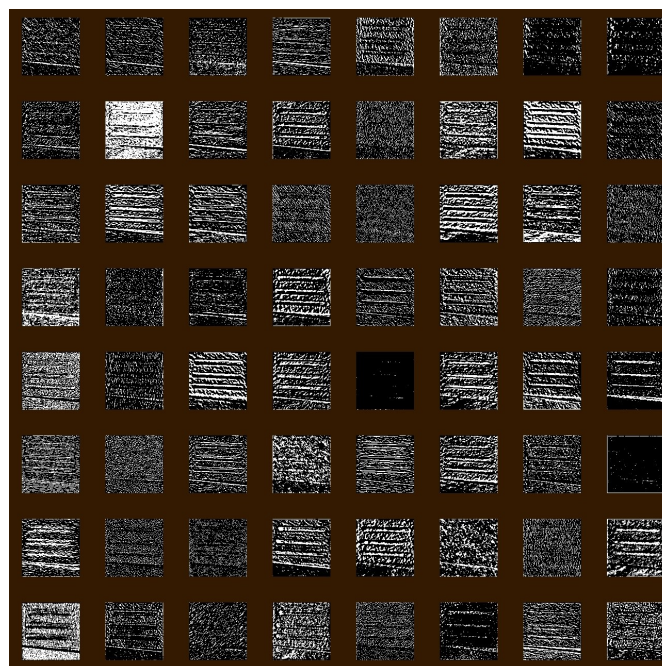


Figure 5.6: Part of feature maps exported on Block 2, size 112x112

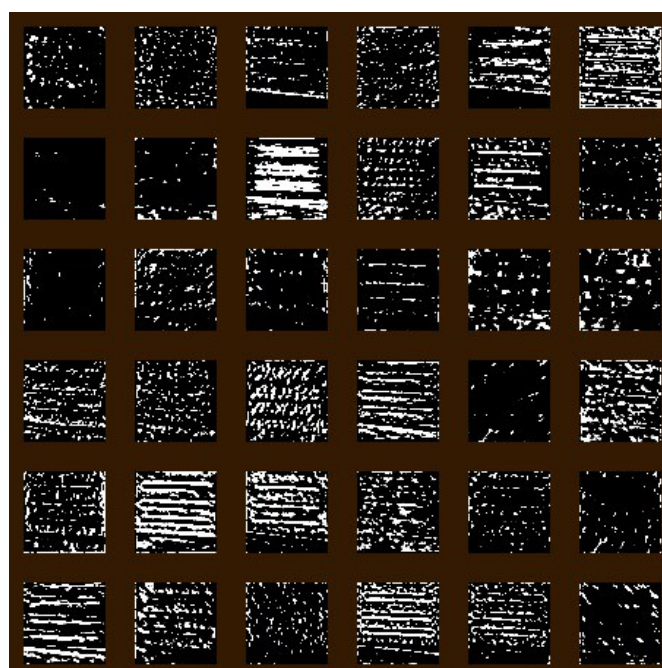


Figure 5.7: Part of feature maps exported on Block 3, size 56x56

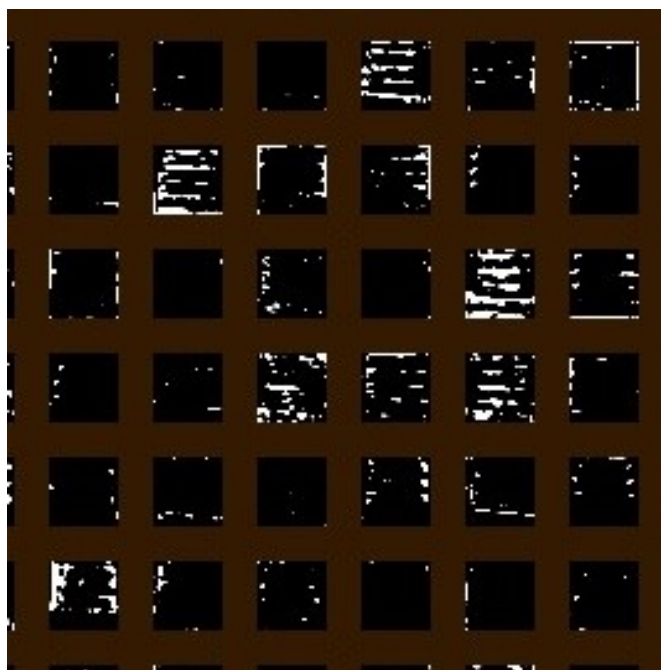


Figure 5.8: Part of feature maps exported on Block 4, size 28x28

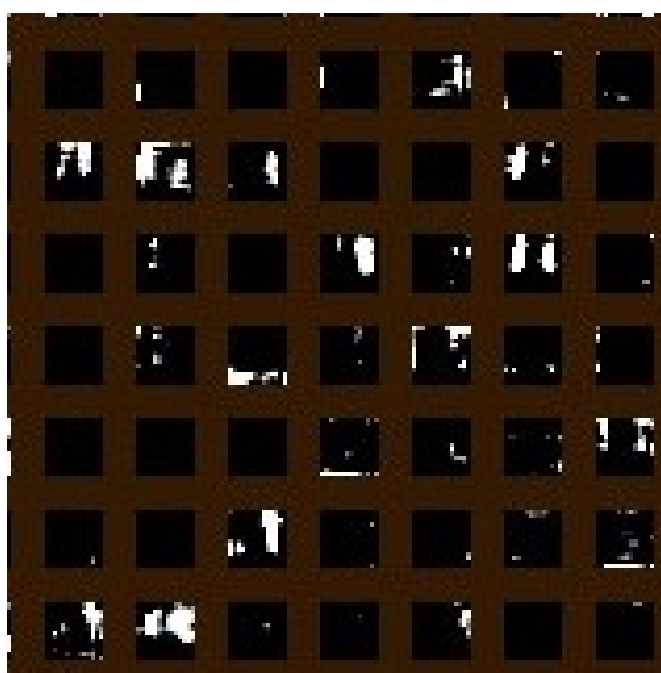


Figure 5.9: Part of feature maps exported on Block 5, size 14x14

Chapter 6

Results

6.1 General information about training

The model proposed was trained using the whole dataset which was randomly split into training and testing dataset using a ratio of $\sim 80:20$; 233 and 67 images were used for training and testing accordingly. The augmentation technique was applied 2 times on the training set, thus the total training dataset increased to $3 * 233 = 699$ images. In the proposed architecture, transfer learning was employed by freezing layers of the convolutional base of VGG16. Two strategies were tested; in the first one (Case A) the total weights on the encoder were frozen, Block 1 - Block 5, and in the second one (Case B) were frozen Block 1 - Block 4 ([Figure 5.2](#)). The total model's parameters as well as the trainable and non-trainable parameters in each case are presented on [Table 6.1](#). ADAM optimizer was used and various learning rates η tested, such as 10^{-4} , 10^{-5} and 10^{-6} to reach the best convergence with maximum possible accuracy. The loss was calculated by Binary Cross Entropy formula as defined. The data was converted to CUDA arrays of (maximum) Float32 format and the model was trained entirely on the GPU Nvidia GTX 1060 mobile 6gb increasing the speed of convolutional operations. However, the batch size was limited to 3 images ($224 \times 224 \times 3$ pixels each) due to GPU memory restrictions and the weights were updated after each batch via back propagation based on the batch-loss. In addition, the whole training data were shuffled and new batches formed at the beginning of every epoch to reduce overfitting and variance.

The time needed for a single pass of the training dataset through the model was ~ 1.5 -3 min and the performance was being monitored during the training process to stop training once the model performance stops improving.

Table 6.1: VGG16-UNet parameters

Phase	Layer	Parameters
Encoder (VGG16)	Block 1-4	7,635,264
	Block 5	7,079,424
Decoder	Block 6-9	8,007,489
Total parameters	Block 1-9	22,722,177

Case A: Trained params	8,007,489
Case B: Trained params	15,086,913

The learning rate of $\eta = 10^{-5}$ produced better results in both Case A & B models which

are presented in the following pages.

Note

The output of the model is a grayscale image, with values of pixels in a range of $[0,1]$ in which a threshold (th) is applied to become binary, where 1.0 is white (object) and 0.0 black (background). The default threshold was applied is $th = 0.5$ but in case a different value is used, this is mentioned.

6.2 Results

The first model trained according to Case A strategy with learning rate $\eta = 10^{-5}$. The spikes observed in training loss ([Figure 6.1](#)) can be attributed to a high learning rate and additionally, according to the community, to the small batch size used with Adam optimizer. But overall the mean testing loss through the last 10 epochs tends to be lower than the testing loss.

Mean BCE loss of Epochs [240:250] | Training = 0.137 | Testing = 0.154

The metrics of testing set shows in [Figure 6.2](#) that the effectiveness of the model is relatively low. The $IoU \approx 0.37$ and the other parameters can be improved as the visual results seem promising, as [Figure 6.3](#) illustrates.

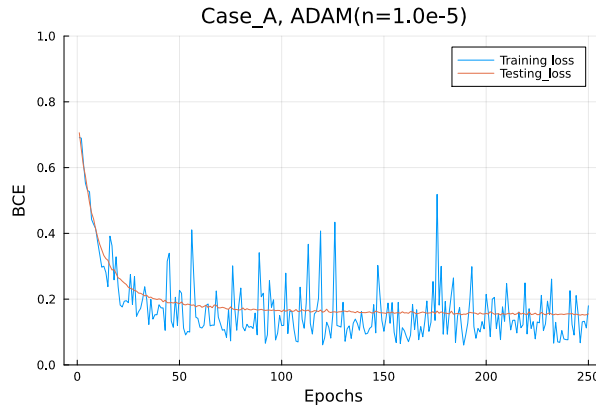


Figure 6.1: Model - Case A: BCE loss

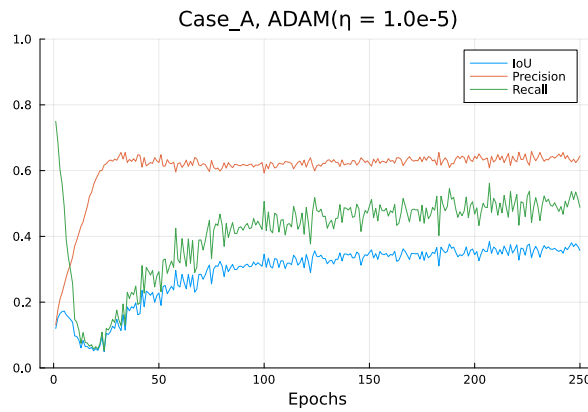


Figure 6.2: Model - Case A: Testing Set - Mean values; IoU, Recall, Precision

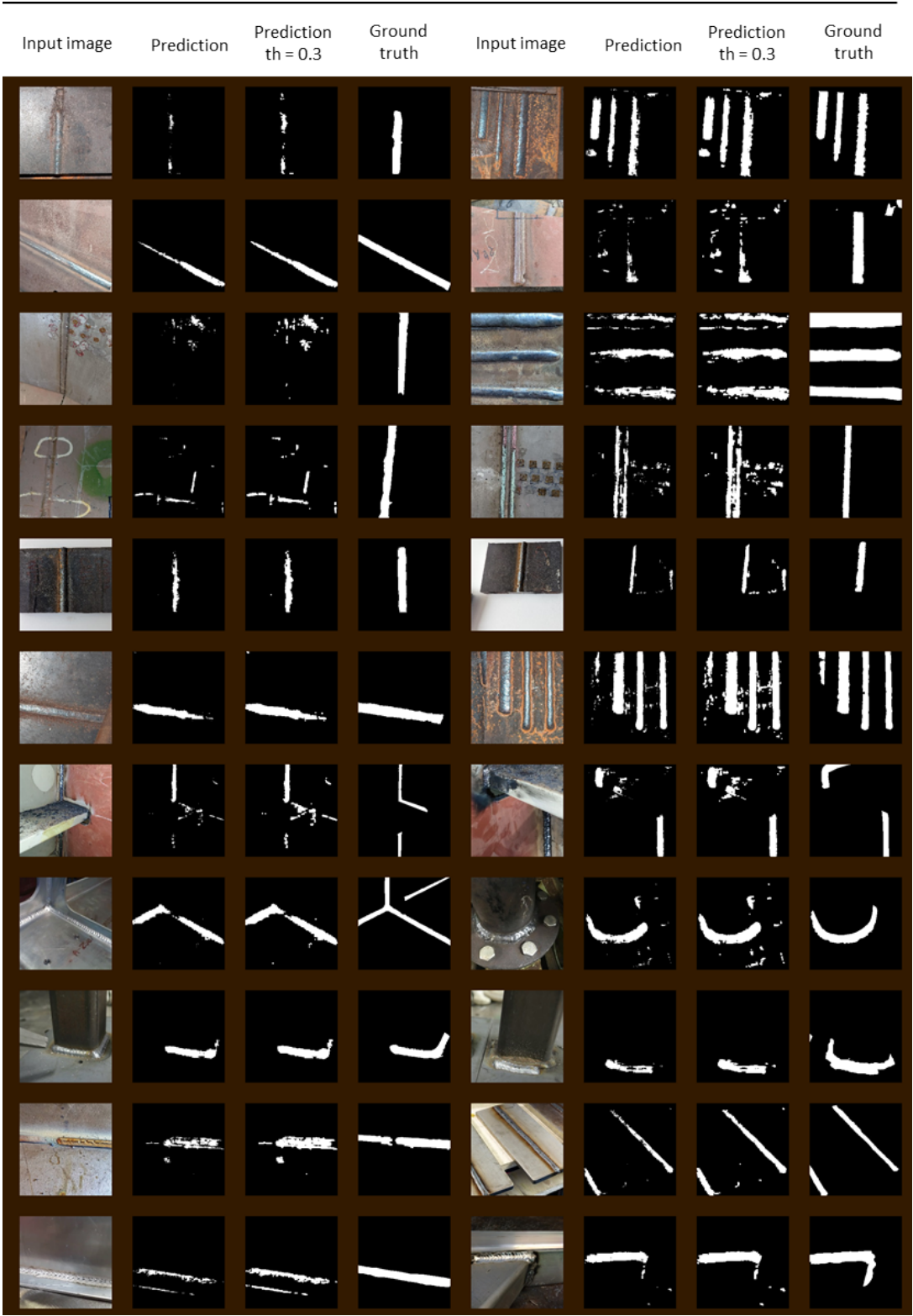


Figure 6.3: Model - Case A results

As discussed in the transfer learning section high-level feature representations are extracted from the final layers of the convolution core and in some cases, it is suggested to retrain the last layer of the encoder, in order for the model to be adapted in accordance to the training set. For that reason the Case A strategy was abandoned and the following models trained implementing the Case B.

The results presented below show a significant enhancement due to the increment of the trainable layers. The [Figure 6.4](#) demonstrates the losses during training process. The spikes of training loss which occurred in Case A ([Figure 6.1](#)) tend to be eliminated. As far as the deviation between training and testing loss after Epoch 150 is observed, it could be a sign of overfitting but it does not seem to decay the mean value of the other metrics as shown in [Figure 6.5](#). Nevertheless, the results presented below are exported from the Epoch 150.

In addition, in [Figure 6.5](#) a comparison of the usage of threshold 0.3 instead of 0.5 is demonstrated. Based on this plot, by applying the $th = 0.3$ it is observed an 1.7% increase of IoU. The Recall is also increased by 13% which is an advantage but there is an 9% decrease in Precision (more noise produced).

The mean value of $IoU = 0.46$ achieved in the testing set is an adequate result given that for the training set the mean value is $IoU = 0.59$.

Another significant positive result is that the mean Recall values can be higher than 70% if $th = 0.3$ is applied, which means that the network has learnt in great extent the pattern and geometry of weld. However, the F1-score remains almost the same regardless the threshold value applied.

From the visual results presented below we can also conclude that the model does not produce lots of false predictions (red pixels) [Figure 6.7- 6.13](#). Despite the fact that a relatively small amount of images was used for training and that they were not taken from a controlled industrial environment, most of False positive pixels appeared mainly near the circumference of the weld. In cases of high percentage of noisy pixel, the model seems to be distracted by "uncommon" objects for it, such as spray paints, cables, etc.

Table 6.2: Summary of metrics at Epoch 150

Dataset	th	BCE Loss	IoU	F1-score	Precision	Recall
Training set	0.5	0.086	0.59	0.70	0.66	0.82
	0.3		0.58	0.69	0.61	0.89
Testing set	0.5	0.15	0.46	0.60	0.64	0.64
	0.3		0.47	0.61	0.59	0.72

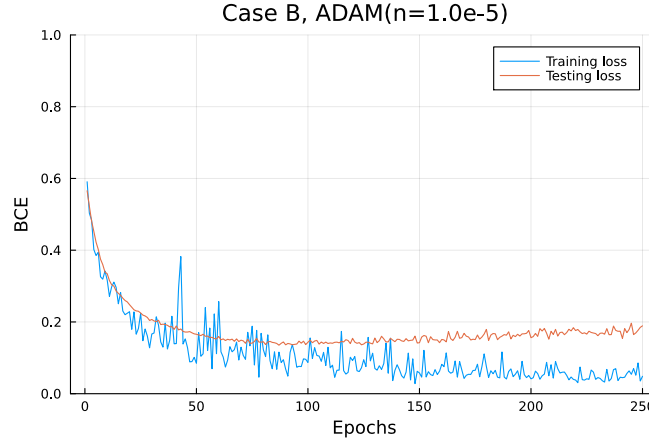


Figure 6.4: Model - Case B: BCE loss

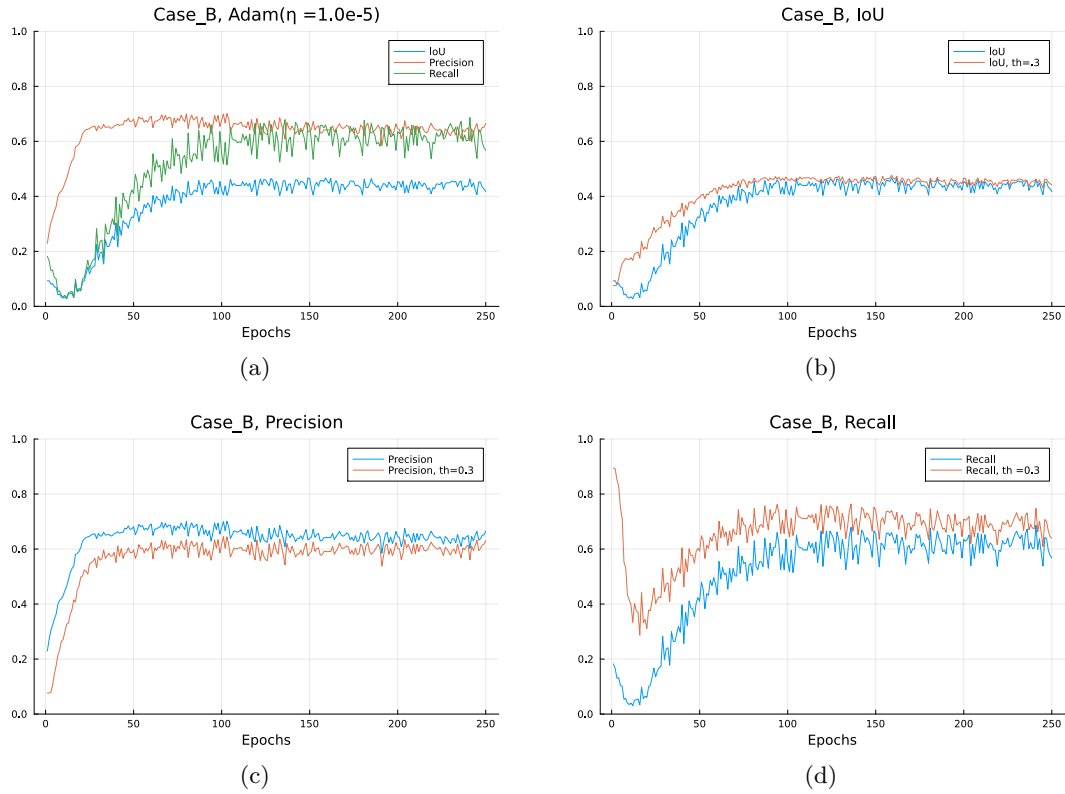


Figure 6.5: Model - Case B: Testing Set - Mean values (a)Summary of Metrics; (b)IoU(th=0.3 and 0.5); (c)Recall(th=0.3 and 0.5), (d)Precision(th=0.3 and 0.5)

It is also worth discussing these interesting facts revealed by scattering for each image the Precision vs Recall at Epoch 150 in [Figure 6.6](#).

- There are images with high Recall and low Precision. The model predicted the weld but there is a large percentage of noise (either in the circumference of the weld or elsewhere).
- There are images with high Precision and low Recall. The model made mainly correct predictions but did not predict the entire surface of the weld.

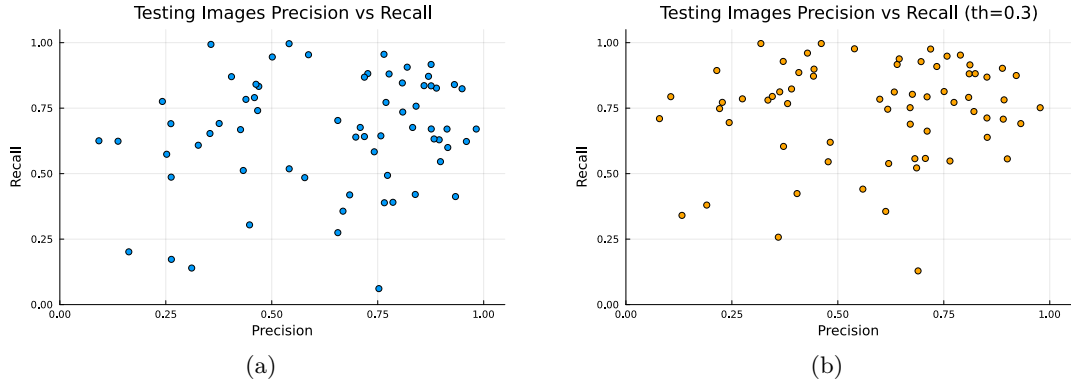


Figure 6.6: Model - Case B: Testing Set - Epoch 150 (a)Precision vs Recall ; (b)Precision vs Recall (th=0.3)

In the following pages are presented visual results of the testing images ([Figure 6.7-6.13](#)). For the sake of completeness are presented and samples of training set. In *Ground truth & Prediction* column, the two individual masks were overlapped and the pixels are depicted with different colors depending on the category they are classified:

- White pixels: True Positive
- Cyan pixels: False Negative
- Black pixels: True Negative
- Red pixels: False Positive

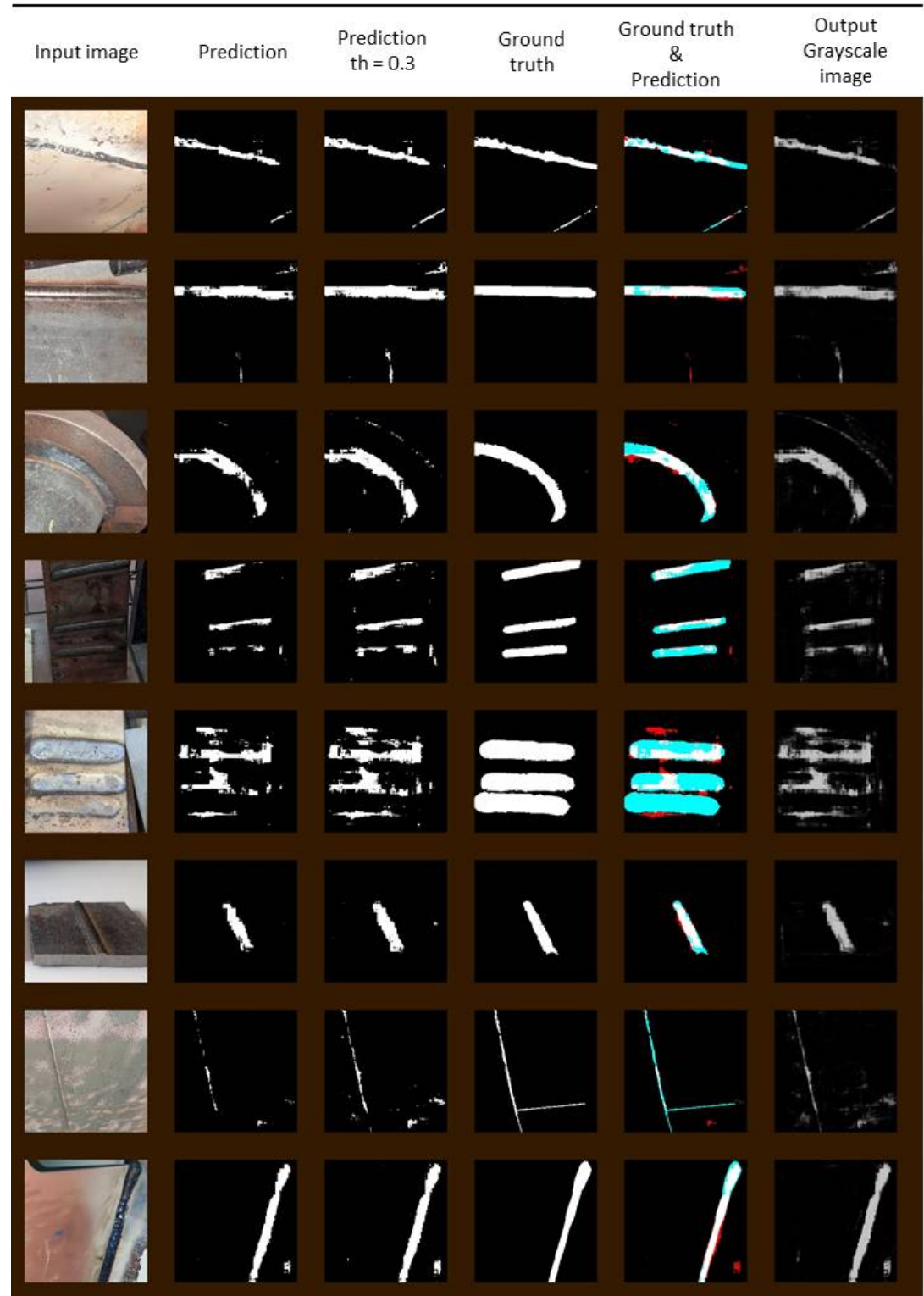


Figure 6.7: Model - Case B Testing set results 1

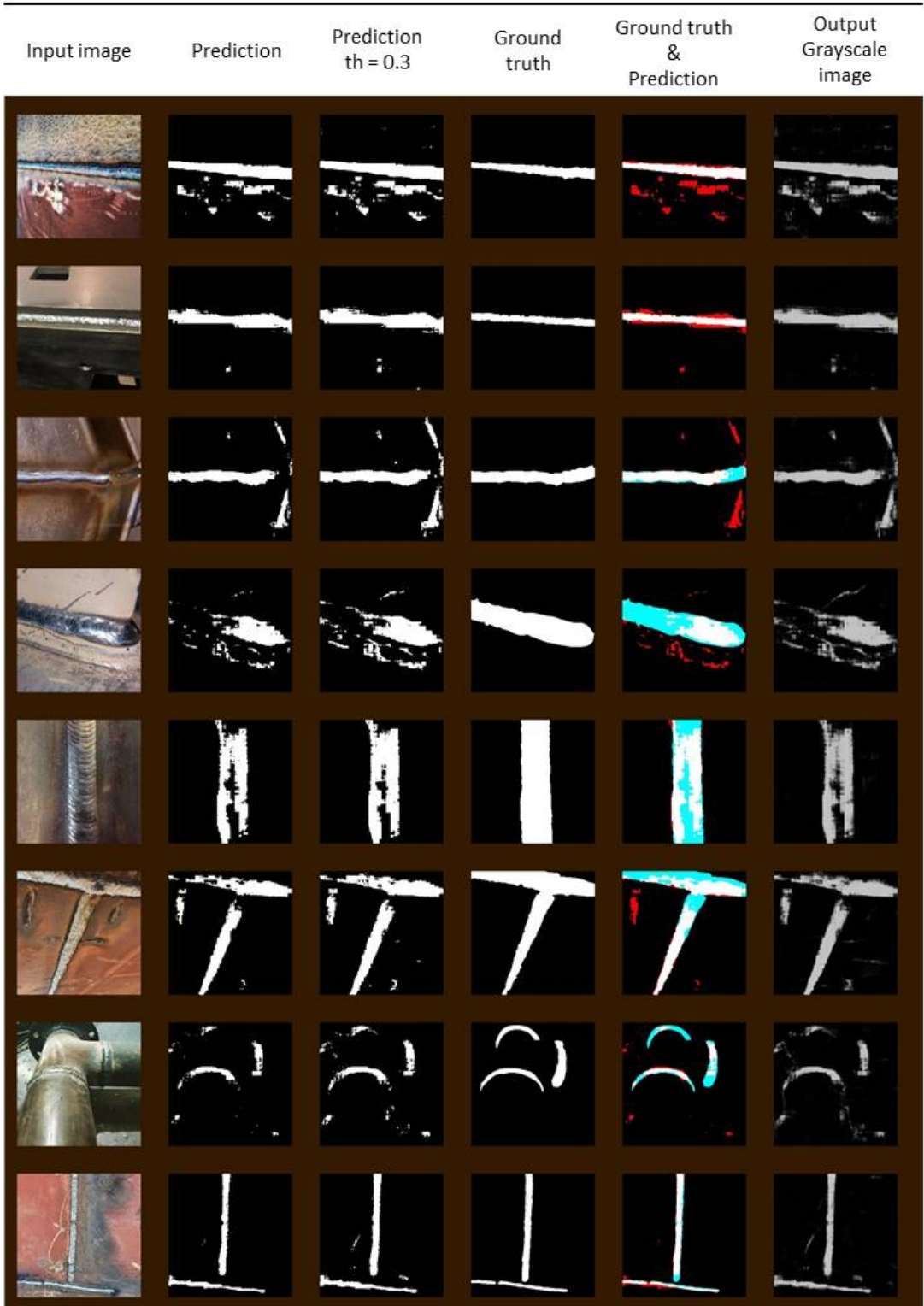


Figure 6.8: Model - Case B Testing set results 2

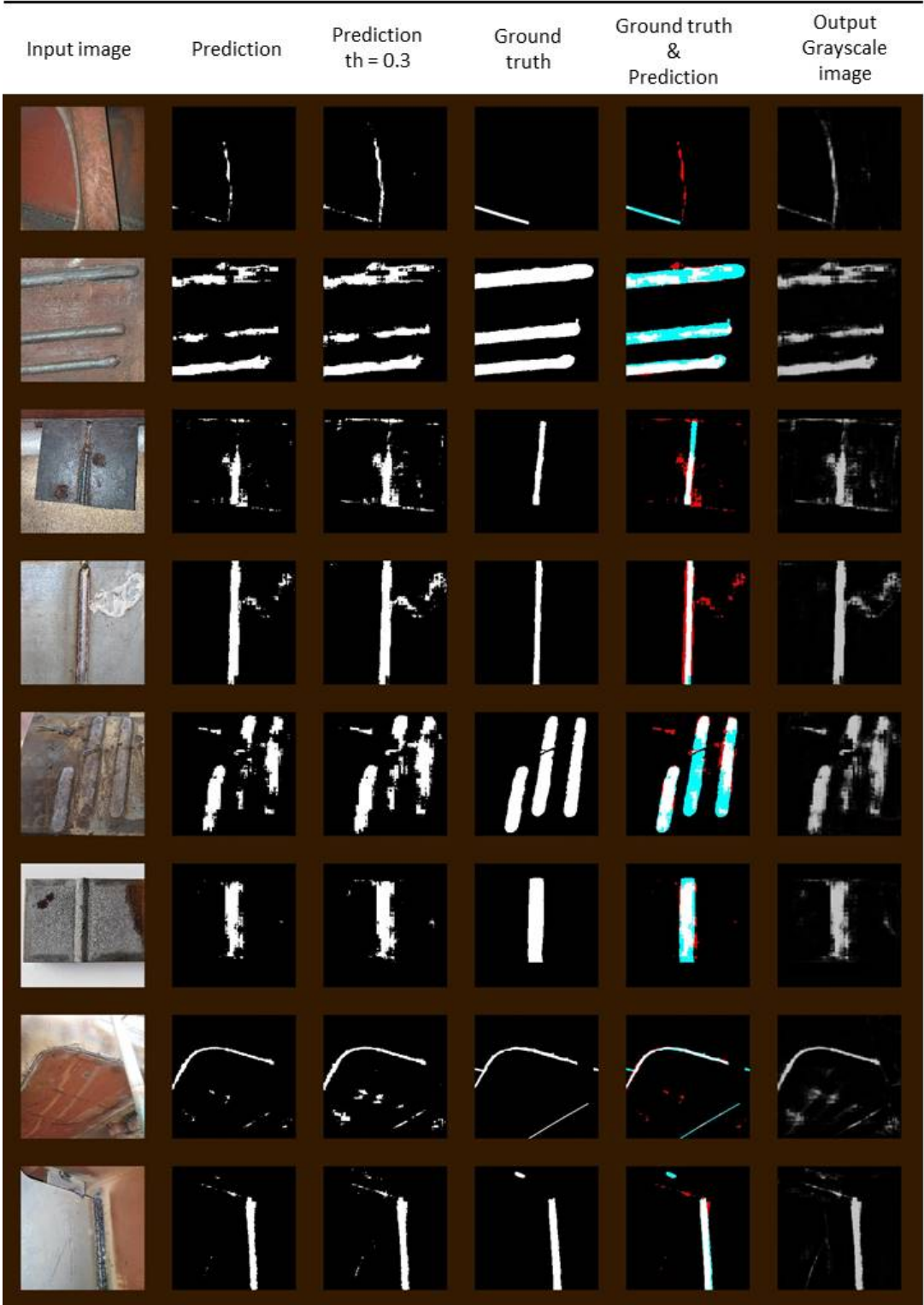


Figure 6.9: Model - Case B Testing set results 3

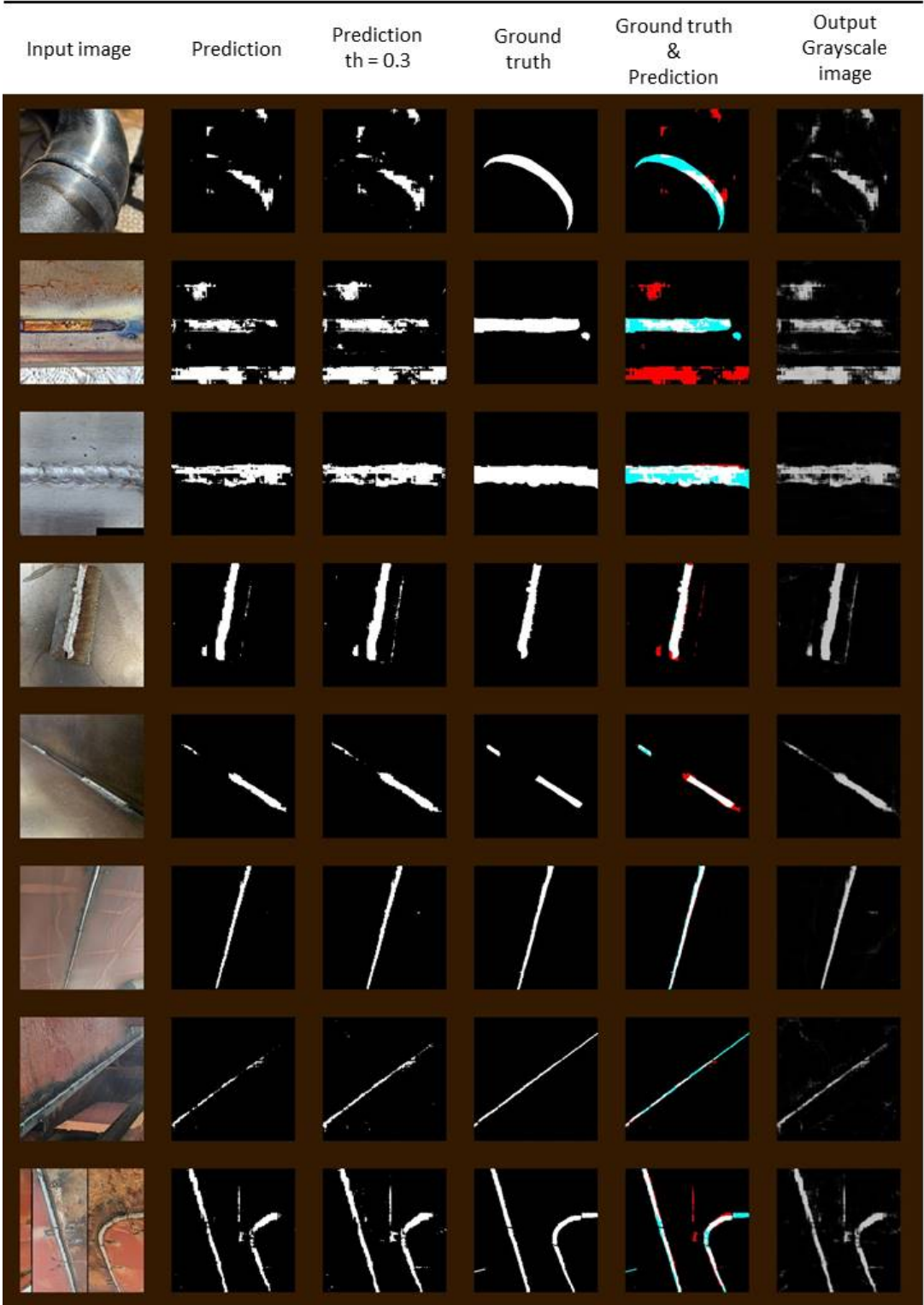


Figure 6.10: Model - Case B Testing set results 4



Figure 6.11: Model - Case B Testing set results 5

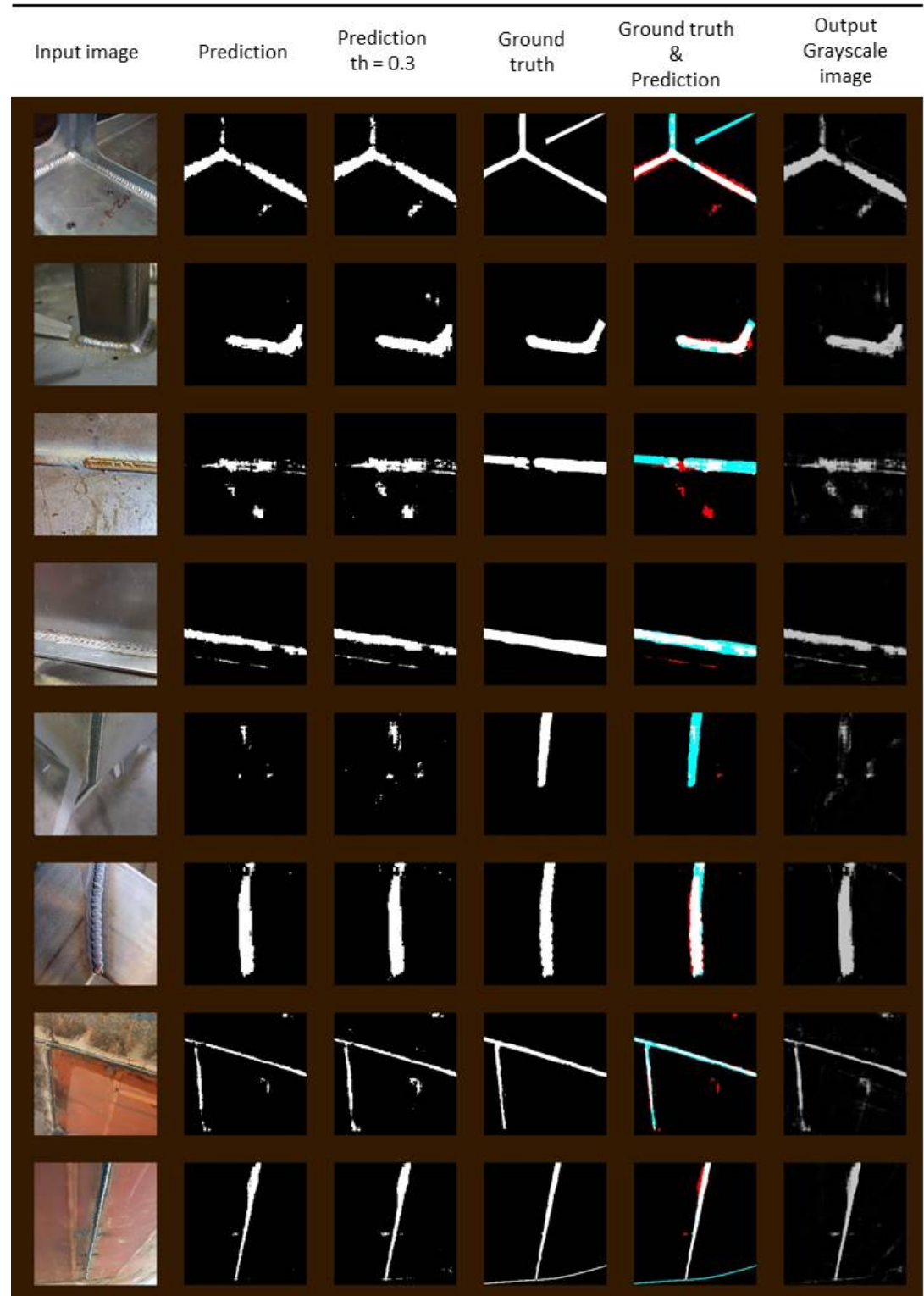


Figure 6.12: Model - Case B Testing set results 6



Figure 6.13: Model - Case B Training set

Chapter 7

Conclusion

In this thesis, an automated algorithm for semantic segmentation of welds was provided. The initial thought was the model to be generalized, therefore images from different sources were collected for the dataset. The model was trained using transfer learning techniques, thus overfitting was avoided and finally produced satisfactory results for both training and testing sets (Testing set: IoU = 0.46, F1-score = 0.6, Precision = 0.64 and Recall = 0.64). The threshold of IoU to be considered an acceptable prediction is 0.5 and in our case the model, despite the small amount of dataset, almost achieved the acceptable threshold. The visual results of the testing set illustrate that the model can predict almost in all images the existence and the location of the weld. According to these values and the visual results the model can be used for weld localization and existence tasks. For further analysis of the predicted welding surface, the model could be sufficient if it produces higher IoU and F1-score values. This can be achieved by enriching significantly the dataset with more images.

Bibliography

- [1] B.J. Moniz and R.T. Miller. *Welding Skills*. American Technical Publishers, 2010.
- [2] P.L. Moore. *The importance of welding quality in ship construction*, pages 357–363. 01 2009.
- [3] Laura Poggi, Cesare Mario Rizzo, Tomaso Gaggero, Marco Gaiotti, and Enrico Ravina. Assessment of ship robotic inspections. pages 1495–1502, 09 2020.
- [4] Drones herald in new era of inspections - industry insights - dnv. <https://www.dnv.com/expert-story/maritime-impact/Drones-herald-in-new-era-of-inspections.html>, 06 2019. (Accessed on 11/05/2022).
- [5] Dnv gl’s remote surveys surge with 15,000 completed since launch. <https://www.dnv.com/news/dnv-gl-s-remote-surveys-surge-with-15-000-completed-since-launch-171041>, 03 2020. (Accessed on 11/05/2022).
- [6] MultiMedia LLC. Robohop, 2020.
- [7] Miguel Carrasco and Domingo Mery. Segmentation of welding defects using a robust algorithm. 62, 01 2004.
- [8] A. Nirmala V. Sridevi. Inspection of welding images using image segmentation techniques. 62, 03 2013.
- [9] Zhifen Zhang, Guangrui Wen, and Shanben Chen. Weld image deep learning-based on-line defects detection using convolutional neural networks for al alloy in robotic arc welding. *Journal of Manufacturing Processes*, 45, 07 2019.
- [10] Chenhua Liu, Shen Chen, and Jiqiang Huang. Machine vision-based object detection strategy for weld area. *Scientific Programming*, 2022:1–12, 04 2022.
- [11] Tianyuan Liu, Jinsong Bao, Junliang Wang, and Yiming Zhang. A hybrid cnn-lstm algorithm for online defect recognition of co2 welding. *Sensors*, 18(12), 2018.
- [12] Sergey Shevchik, Christoph Kenel, Christian Leinenbach, and Kilian Wasmer. Acoustic emission for in situ quality monitoring in additive manufacturing using spectral convolutional neural networks. *Additive Manufacturing*, 21:598–604, 05 2018.
- [13] Yang Lei, Huaixin Wang, Benyan Huo, Fangyuan Li, and Yanhong Liu. An automatic welding defect location algorithm based on deep learning. *NDT E International*, 120:102435, 03 2021.

- [14] Christian Nowroth, Tiansheng Gu, Jan Grajczak, Sarah Nothdurft, Jens Twiefel, Jörg Hermsdorf, Stefan Kaierle, and Jörg Wallaschek. Deep learning-based weld contour and defect detection from micrographs of laser beam welded semi-finished products. *Applied Sciences*, 12(9), 2022.
- [15] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon. A PROPOSAL FOR THE DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE. <http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html>, 1955.
- [16] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.
- [17] Dan Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. pages 1237–1242, 07 2011.
- [18] Richard Szeliski. Computer vision algorithms and applications, 2011.
- [19] Bo Sun, Abdullah Iliyasu, Fei Yan, Fangyan Dong, and Kaoru Hirota. An rgb multi-channel representation for images on quantum computers. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 17:404–417, 05 2013.
- [20] P.Y. Simard, D. Steinkraus, and J.C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pages 958–963, 2003.
- [21] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [22] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 07 2011.
- [23] Nitish Srivastava Geoffrey Hinton and Kevin Swersky. Divide the gradient by a running average of its recent magnitude, lecture 6e.
- [24] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.
- [25] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- [26] keywords = 2017 book machine-learning oreilly tensorflow textbook publisher = O'Reilly Media timestamp = 2018-04-06T05:59:31.000+0200 title = Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems year = 2017 Geron, Aurelien isbn = 978-1491962299. Sebastopol, CA.
- [27] Shallu Sharma and Dr. Rajesh Mehra. Implications of pooling strategies in convolutional neural networks: A deep insight. *Foundations of Computing and Decision Sciences*, 44:303–330, 08 2019.

- [28] Mars Xiang. Convolutions: Transposed and deconvolution | by mars xiang | medium. <https://medium.com/@marsxiang/convolutions-transposed-and-deconvolution-6430c358a5b6>, 07 2020. (Accessed on 11/05/2022).
- [29] Chengxi Ye, Matthew Evanusa, Hua He, Anton Mitrokhin, Tom Goldstein, James A. Yorke, Cornelia Fermüller, and Yiannis Aloimonos. Network deconvolution, 2019.
- [30] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 02 2015.
- [31] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.
- [32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.
- [33] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3523–3542, 2022.
- [34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. volume 9351, pages 234–241, 10 2015.
- [35] Lisa Torrey, Jude Shavlik, Trevor Walker, and Richard Maclin. *Transfer Learning via Advice Taking*, volume 262, pages 147–170. 12 2009.
- [36] The GIMP Development Team. Gimp.
- [37] Otto Debals and Bert De Brabandere.
- [38] The julia programming language. <https://julialang.org/>. (Accessed on 11/06/2022).
- [39] DhairyaLGandhi. Unet.jl v0.2.0, Sept 2020.
- [40] Michael Innes, Elliot Saba, Keno Fischer, Dhairya Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. Fashionable modelling with flux. *CoRR*, abs/1811.01457, 2018.
- [41] Mike Innes. Flux: Elegant machine learning with julia. *Journal of Open Source Software*, 2018.
- [42] Anindya Pravitasari, Nur Iriawan, Mawanda Almuhayar, Taufik Azmi, Irhamah Irhamah, Kartika Fithriasari, Santi Purnami, and Widianana Ferriastuti. Unet-vgg16 with transfer learning for mri-based brain tumor segmentation. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 18:1310, 06 2020.
- [43] Iman Kazerouni, Gerard Dooly, and Daniel Toal. Ghost-unet: An asymmetric encoder-decoder architecture for semantic segmentation from scratch. *IEEE Access*, PP:1–1, 07 2021.
- [44] Shruti Jadon. A survey of loss functions for semantic segmentation. 06 2020.

- [45] Michal Drozdal, Eugene Vorontsov, Gabriel Chartrand, Samuel Kadoury, and Chris Pal. The importance of skip connections in biomedical image segmentation. *CoRR*, abs/1608.04117, 2016.