Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Henrik Heien

Towards Remaining Life Assessment Using Machine Learning

Master's thesis in Marine Technology
Supervisor: Sigmund Kyrre Ås
Co-supervisor: Marius Andersen
June 2021

NTNU
Kunnskap for en bedre verden

Henrik Heien

Towards Remaining Life Assessment Using Machine Learning

Master's thesis in Marine Technology
Supervisor: Sigmund Kyrre Ås
Co-supervisor: Marius Andersen
June 2021

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

NTNU

Kunnskap for en bedre verden

Summary

Mooring line failure due to fatigue and corrosion is a problem the offshore industry is facing. Multiple line failure in a permanent mooring system has an annual frequency of around 3 x 10^{-3} and most failures happen well within the mooring chains' design-life. Determining remaining fatigue life of a mooring system by conventional methods have proven to be an accurate but also tedious process. However, artificial Neural Networks have shown to produce both quick and accurate results in engineering. This thesis focuses on demonstrating the feasibility of predicting stress fields around corrosion pits by using generative adversarial networks. Generative adversarial networks have proven to be able to generate images that look very realistic to the human eye and this thesis examines their ability to recreate finite element analysis results.

To train a machine learning model was two data sets of stress fields were created using finite element modeling in ANSYS APDL, one were the target was a gray scale image of the stress field in the surface of pits and one where the target was the stress field $1\,\mathrm{mm}$ below the surface. Both data sets uses a gray scale image of the pits surface topology as an input. The element analysis was done on pits with a tension of $1\,\mathrm{MPa}$, and the 3D model of the pits were made from surface topology images of corrosion pits in mooring chains. The first principal stress in both the surface and $1\,\mathrm{mm}$ below the surface was used to generate gray scale images where the pixel vales represented stress in the particular location.

The GAN model proposed by Isola et al. (2017) was used as a basis when developing a machine learning model that generate accurate images of the stress fields. To optimize the model for finding the peak stresses in the image was generator loss function experimented with and changed to include root mean squared error. To reduce the training time and computational time of the model was the size of the model reduced. The reduction was done after experimenting with different model sizes.

The result of this thesis is the python program "Generate_images.py" that uses two trained machine learning models to predict stress fields around pits. The program accuracy is investigated by analysing 1000 pit images that were not used in the training and comparing the result with results finite element analysis. It is concluded that the program is able to accurately predict the stress fields in the surface and $1\,\mathrm{mm}$ below the surface in and around corrosion pits with depth ranging from $0\,\mathrm{mm}$ to $12\,\mathrm{mm}$.

Sammendrag

Brudd i ankerkjettinger grunnet korrosjon og utmatting er et problem offshorenæringen møter. Brudd i flere kjettinger i permanent oppankrede systemer har vist seg å ha en årlig frekvens på omtrent 3 x 10^{-3} og de fleste bruddene skjer godt innenfor kjettingens design liv. Å bruke konvensjonelle metoder til å avgjøre gjenværende levetid i kjettinger har vist seg å være en nøyaktig, men også tidkrevende prosess. Derimot har maskinlæring vist seg å gi både raske og nøyaktige resultater innenfor ingeniørfag. Denne oppgaven setter søkelys på å demonstrere mulighetene for å anslå spenning rundt en korrosjonsgrop ved å bruke "generative adversarial networks". Det har blitt vist at "Generative adversarial networks" klarer å generere bilder som ser svært realistiske ut, og denne oppgaven undersøker om de klarer å gjenskape resultater funnet ved elementmetode.

To datasett ble skapt ved å bruke elementmodelering i ANSYS APDL. Et der målet er et gråskalabilde av spenningsfeltet i overflaten av en grop og et der målet er spenningsfeltet $1\,\mathrm{mm}$ under gropen. Begge datasettene bruker et gråskala bilde av gropens topologi som input. Elementmetode analyse ble gjennomført med plant strekk av $1\,\mathrm{MPa}$. Topologibilder fra korrosjon i ankerkjettinger ble brukt da 3D modeller av gropene ble laget. Spenningen i overflaten og 1 mm under ble brukt til å generere gråskalabildene der pikselverdiene representerte spenningen i lokasjonen pikselen representerte.

GAN modellen lagt frem av Isola et al. (2017) ble brukt som grunnlag under utviklingen av maskinlæringmodellen som skulle generere bildene av spenningsfeltet. For å optimalisere nettverket for å finne høyeste spenning ble det eksperimentert med tapsfunksjonen brukt i maskinlæringen. For å redusere utregningstiden i nettverket ble det også eksperimentert med forskjellige modellstørrelser, og størrelsen på nettverket ble halvert.

Resultatet fra denne oppgaven er Python-programmet "Generate_images.py" som bruker to trenede modeller til å anslå spenningsfeltene rundt en grop. Programmets nøyaktighet er så undersøkt ved å analysere 1000 groper som ikke ble brukt til å trene modellen. Det er konkludert med at programmet er svært nøyaktig når det anslår spenninger i overflaten og $1\,\mathrm{mm}$ under overflaten rundt korrosjons groper med dybde mellom $0\,\mathrm{mm}$ og $12\,\mathrm{mm}$.

Preface

This thesis concludes my master's degree in Marine Technology at NTNU and is a part of the Sintef life-more project. The work done in this thesis is based on a model presented by Isola et al. (2017). A Tensorflow code is used with several changes made such that the model fits the problem presented in the thesis.

Another student, Håkon K. Pettersen, has worked on the same problem in their thesis, and the subject has been discussed regularly. A result of this may be that some decisions are similar between the two theses. However, the theses themselves are written separately from each other.

The work in this thesis has been performed in the following way. The finite element modeling was performed early in the semester, and the method from the project thesis was used to develop 3D models for analysis. Extracting the stress from the nodes under the surface proved to be a challenge, however, Håkon K. Pettersen found the method of defining parts, which made the process easy. Running FEM on thousands of corrosion pits was time-consuming, and a total of two weeks of computational time was used to develop the data sets used in machine learning. Handling thousands of images of pits, stress fields, STL models and FEM models proved to be challenging, and some time has been wasted as the pits have been oriented differently before and after the FEM analysis.

The process of developing the machine learning algorithm consisted of trying multiple different methods to solve the problem. A lot of time has gone into trial and error with different machine learning frameworks, and three different pix2pix versions have been used. When the desired framework was found, multiple experiments were performed with different batch sizes, loss functions, and model architecture. When the final training and generator type were chosen, the two final models trained fast, and "Generate_images.py" was developed using many of the functions used when training.

Signed: 

Henrik Heien
10.06.21 Trondheim

Acknowledgements

I want to show my gratitude to my supervisor Sigmund Kyrre Ås and co-supervisor, Marius Andersen, for excellent guidance during our weekly meetings. I would also like to thank Håkon K. Pettersen for our good discussions on the topic.

I also want to thank my mother and Father for their support, guidance, and different approaches. My mother has shown me the value of routine, hard work, and never giving up, and my Father for showing me that sometimes things are not as important as we think. The last five years would not have been the same without their support and guidance, and for that, I am very thankful.

I would also like to thank Emilie for being a large part of my five years in Trondheim, and all the time we have spent together. I am grateful for the countless hours Jon Magnus, Benjamin, and I have spent together writing our theses and the valuable inputs they have given me.

Lastly, I want to show my appreciation of Emil, Jostein, Kristian, Oscar, Stian, Øystein, and Øyvind, and our continuous conversation over the last five years.

Table of Contents

List of Tables

List of Figures

Abbreviations

ANN	=	Artificial Neural Networks
APDL	=	Ansys Parametric Design Language
CFD	=	Computational fluid dynamics
CNN	=	Convolutional Neural Networks
CPU	=	Central Processing Unit
DOF	=	Degree of Freedom
FEA	=	Finite Element Analysis
FEM	=	Finite Element Method
FPSO	=	Floationg Production, Storage and Offloading
GAN	=	Generative Adversarial Network
GPU	=	Graphical Prosessing Unit
MAE	=	Mean absolute error
MSE	=	Mean Square Error
ReLU	=	Rectified linear unit
RMSE	=	Root mean square error
SCF	=	Stress Concentration Factor
TPU	=	Tensor Prosessing Unit
UINT8	=	8 bit unsigned integer
UINT16	=	16 bit unsigned integer

Chapter 1

Introduction

1.1 Mooring line failures

Recent offshore industry studies have found that fatigue is one of the primary reasons for offshore mooring failure. Fontaine et al. (2014), studied a total of 107 reported failures in a total of 72 mooring chains. Figure 1.1 shows the distribution of different types of failures found in the study. We can observe that most of the failures result from either fatigue, corrosion, or a combination of the two.

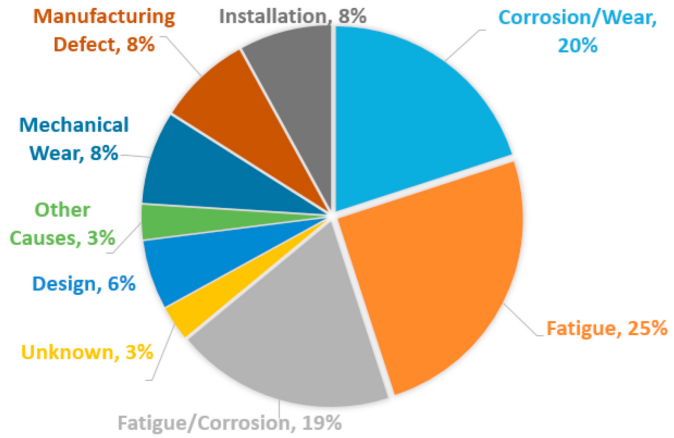


Figure 1.1: The distribution of different failures.(Fontaine et al., 2014)

These reasons have also lead to the failure of mooring chains occurring long before the end of theory design life. Fontaine et al. (2014) found that over a third of all failures happened within the first three years of fatigue life. Ma et al. (2013) studied 23 mooring chain failures between 2000 and 2011 and found that 50% of the failures happened within the first three years of life. This high infant mortality of the systems is a reason for concern in the offshore industry. The annual frequency of failure for a mooring system was extensive in both Fontaine et al. (2014) and Ma et al. (2013). They conclude with annual rates of multiple line failure per facility to be around 3.5 x 10^{-3} and 3.0 x 10^{-3} respectfully. This rate of failure is high compared to other accident scenarios for FPSOs and other moored offshore structures. Ma et al. (2013) discusses the types of mooring system components that fail more often. The paper concludes that the chain failure accounts for 50% of the failures, followed by connector and wire. The relatively high failure rate of permanent mooring systems raises a concern in the offshore industry, and fatigue analysis plays an essential role in the mooring design.

Determining the fatigue life of the mooring chain is challenging. Today's practice is to inspect mooring

chains with remotely operated vehicles and build 3D models of damaged components. The finite element method is then applied to the model to find the stresses in the component. This method is time-consuming and tedious and still includes the possibility that failure may occur in an element not chosen for further analysis.

Therefore, is there a considerable upside in developing a faster method to determine mooring chains' fatigue life. Machine learning models have recently proven themselves to find accurate results in multiple engineering practices. The possibility of using machine learning to find peak stresses in a mooring link chain is therefore very interesting.

1.2 Hypothesis

The hypothesis of this thesis is that one may utilize machine learning to calculate the remaining lifetime in a mooring chain link. To investigate this will images of corrosion pits in a mooring chain used to try to estimate peak stresses in single pits.

1.2.1 Calculating peak stresses

We may find the stress concentration factor for each pit, by multiplying the locational SCF with the geometrical SCF. The stress will vary over the surface of a mooring chain link in tension. The tension depends on the amplitude of the tension and the angle between the links. Figure 1.2 shows the stress field on the surface of a mooring chain found by Kim et al. (2019). The stress field may vary between mooring chain types.

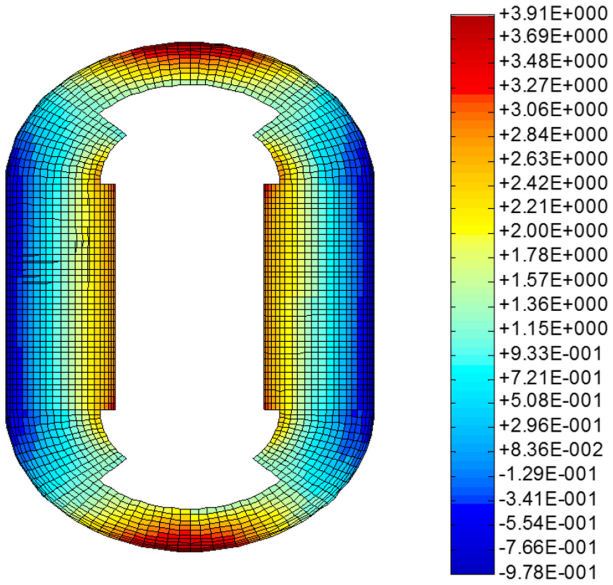


Figure 1.2: Stress field of mooring chain

With a large change in the surface topology, such as a corrosion pit, should one expect higher stresses. Determining the stress is often done with a finite element model. If we would like to determine the surface stress of the complete mooring chain link, are we required to model all surface pits with the fem model.

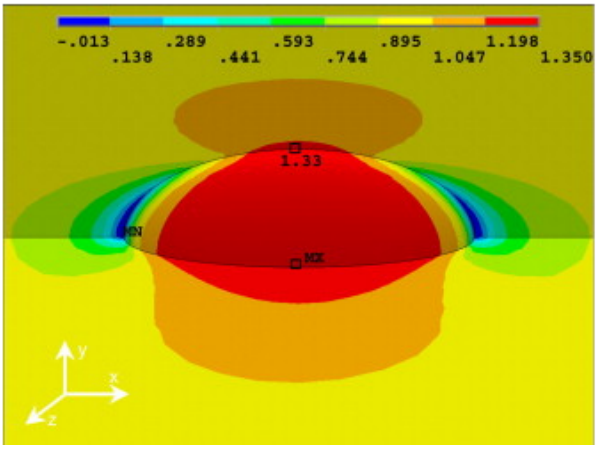


Figure 1.3: Stress field in pit

1.2.2 Goal

The goal of this thesis is to develop a program that uses machine learning models to find the stresses in and around a pit such that the stresses may be used to calculate the fatigue life of a mooring chain. A less complicated model will be valued more than a complicated one with the same accuracy.

Chapter 2

Relevant Literature

2.1 Literature on fatigue and mooring chains

Fracture mechanics have been studied for a long time. We find the Linear Elastic Fracture Mechanics (LEFM) in fracture mechanics, which is the fracture's basic theory. This theory was originated by Griffith (1921) and finalized by Irwin (1957). With this method, quantification of stress fields can be done analytically. The crack growth behavior can be evaluated by applying analytical methods based on numerical approximations or the FEM. However, the estimation of fatigue life for notched components has not been studied until recently.

2.1.1 The modeling of Mooring chains

Bergara et al. (2020) provides a detailed explanation of the calculations of stress intensity factors and relevant sources. The article compares SIF found on a chain using the analytical solutions with SIF found using Abaqus's FEM software. The article shows the validity of the analytical solution for simple geometries. However, the results could not be reproduced for more complex geometries as f.ex. mooring chains. The paper also describes a methodology for calculating SIF for mooring chains with semicircular and straight cracks.

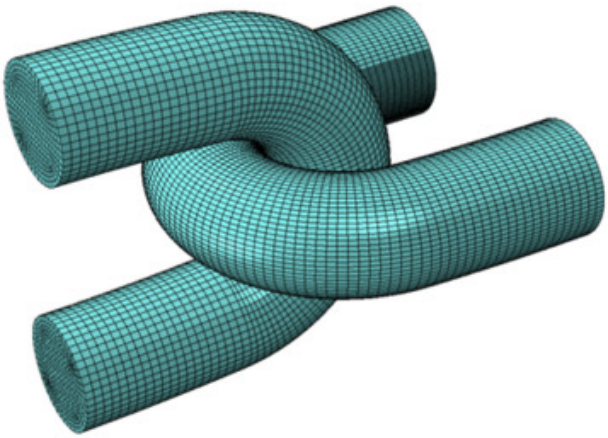


Figure 2.1: FEM modeling done by Bergara et al. (2020)

Gemiland et al. (2021) proposes a method for numerical modeling of mooring chains. The paper shows that the explicit modeling approach should be utilized for the accurate assessment of mooring chains.

This method provides the most realistic response and reduces the computational cost. There is also no convergence problems.

2.1.2 Superposition of stress fields

Paul JR and Faucett (1962) investigates how two stress-raising notches are placed in the region of maximum influence affect each other. The result they present strongly indicates that multiplication of the theoretical stress concentration factors for the individual notches will give the superposed theoretical stress concentration factor in cases where stresses are linearly related. This may be used when two corrosion pits are located close to each other.

2.2 Machine learning in engineering

Using machine learning to fast and accurately predict values that are hard and time-consuming to find using conventional methods has a huge potential.

M.K. et al. (2010) used the application of an artificial neural network to predict the corrosion of alumina plates. By using four different networks were they able to predict four different values using the same input data. The networks used in the paper were small compared to others, having between 9 and 12 nodes each. A data set of 226 samples was used for training and validation, and 62 samples were used to test the networks. The algorithms proved very accurate, and they predicted both the maximum notch diameter and pith depth with small errors. The paper shows that ANNs can predict the results of complex processes with relative ease.

Ok et al. (2007) used machine learning to develop an empirical formula describing localized pitting corrosion's effects on the ultimate strength of unstained plates. The pitting corrosion patterns were modeled as a simplified rectangular shape where the nearest individual pits were grouped and classified. These classes were pits on a single edge, pits in the center, and pits on both edges. To perform the machine learning was, four parameters used as inputs. These were the plate slenderness parameter, the ratio between pit length and plate length, the ratio between pit breadth and plate length, and lastly, the ratio between pit depth and plate thickness. The study used more than 3000 epochs, which lead to an accurate ANN-based empirical formula. The model that consisted of only one hidden layer was used in machine learning. This was done because one layer has proven to be sufficiently accurate and demanding less training data. An issue that is presented in this study is overtraining. This is a problem where the algorithm learns the data by heart rather than learning the data set the trend. To prevent over-training was cross-validation used to stop the training at an appropriate time. A total of 265 non-linear FEM analysis was used, and the total number of epochs was set to 5000. However, the cross-validation ends the learning after 100 epochs with similar results.

2.2.1 Pix2pix paper

Isola et al. (2017) developed a method of generating one image from another using general adversarial networks. The network is very robust and able to solve different problems. In the paper are areal photographs translated to maps, black and white images translated to color images, and images of scenery in daylight translated to the scenery at night. The idea of an image to image translation with color images is to translate one 3 dimensional matrix into another. When grayscale images are used is this problem more similar to 2-dimensional matrices. This method is therefore very relevant to the problem in this thesis. The authors have published their source code in multiple formats and the code has been regularly

updated after the publishing of the paper.

The r

2.3 Result from project thesis

The project thesis done as a study preceding this master thesis showed that a convolutional neural network is able to use images of surface topology to predict the SCF $0.5\,\mathrm{mm}$ under ellipsoid notches. The network architecture used to predict the SCF is found inTable 2.1.

Table 2.1: Model architecture in project thesis

Layer	Attributes	Activation function
Input Layer	150x150 Pixels	–
2D Convolutional	200 Filters of (3×3)	ReLU
Flatten		
Dense 1	64 Nodes	ReLU
Dense 2	64 Nodes	ReLU
Output	1 node	Linear

The model was trained on surface images and a corresponding SCF found using a finite element analysis. In Figure 2.2 are the predicted SCF plotted against the actual SCF.

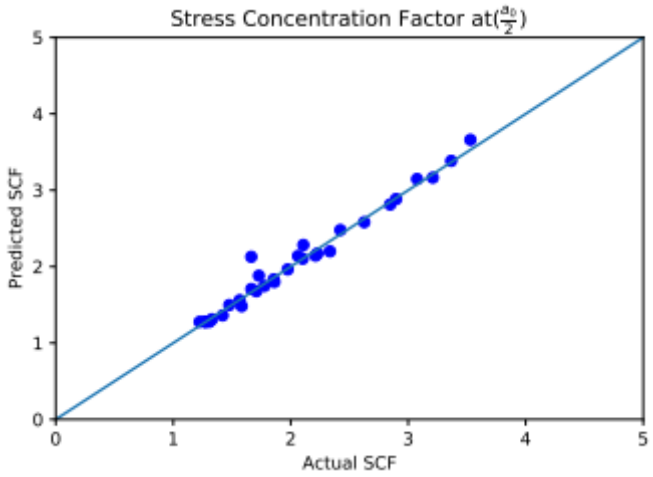


Figure 2.2: Results from project thesis

2.4 Conclusion from literature study

With the results from the project thesis, we can conclude that finding a single value using a convolutional neural network is promising. However, we may be able to predict whole stress fields by using a more advanced method like the image to image translation presented in Isola et al. (2017). Therefore will the model architecture proposed by Isola et al. (2017) be used as a basis when performing image to image translation with pit depths and stress fields. We may also conclude that it is possible to obtain accurate

results without having large data sets or complex networks.

The result from the project thesis may also be used as a sign that a convolutional neural network may be sufficient when estimating peak stresses in a pit. This will be kept in mind while trying to predict the stress field around a pit.

Chapter 3

Basic theory

3.1 Fatigue and Fracture design

There is always a possibility that cracks and defects are present in large-scale structures. These are either weld defects, fatigue cracks, or other types of damage. The influence on material strength from these defects may be significant and should be taken into account. The compendium in fatigue and fracture design, Berge and Ås (2017), provides excellent information on the subject of fatigue and fracture.

3.1.1 Pit corrosion

Corrosion is known to affect the durability of steel negatively by reducing the cross-section area of the structure. Extremely localized corrosion may form small holes in the structure. This phenomenon is referred to as pitting corrosion and may lead to high peak stresses. In this project are the pits from this type of corrosion investigated.

3.1.2 Peak stresses

At structural discontinuities under applied stresses will stress concentrations occur. These discontinuities may be cutouts, welds, notches, or corrosion pits. The relationship between the stresses at these locations and the structure's nominal stresses is the stress concentration factor (SCF). This indicates that knowing the nominal stress in a structure and the SCF is enough to estimate the peak stress in a structure. Therefore is the relevant SCFs very relevant to know.

3.1.3 First principal stress

The 1st principal stress gives you the value of stress that is normal to the plane in which the shear stress is zero. The 1st principal stress helps you understand the maximum tensile stress induced in part due to the loading conditions.

3.1.4 General Applications

Sharp notches with stress concentrations higher than five have been shown to behave like cracks when subjected to fatigue loading. This is because cracks will initiate in sharp notches during the first cycles, and the crack will grow from there if the stress is high enough. The cracks will not grow if the nominal stress is not over a certain threshold. This is similar to the behavior of cracks in the threshold region. With this in mind, we can assume a critical distance a_c ahead of the notch where the crack will either stop growing or continue to grow until failure. This critical crack length can be found by Equation 3.2.

$$\Delta\sigma(r) = \frac{\Delta K_{\text{th}}}{\sqrt{2\pi r}} = \Delta S\sqrt{\frac{a}{2r}} \tag{3.1}$$

$$a_c = \frac{a_0}{2} = \frac{1}{2\pi}\left(\frac{\Delta K_{\text{th}}}{\Delta S_l}\right)^2 \tag{3.2}$$

Here K_{th} and S_l is the long crack threshold and the smooth specimen fatigue limit. These are often found in the literature. This criterion for non-propagating cracks has been verified for various materials and geometries ranging from sharp to blunt notches. The approach is referred to as the theory of critical distances or notch mechanics. This is illustrated in Figure 3.1. The stress at the critical distance $\frac{a_0}{2}$ is found by either analytical methods or by FEM and compared to the fatigue limit. While $\Delta\sigma(\frac{a_0}{2}) < \Delta S_l$ will there be no propagation of the considered crack, which implies a safe component. Knowing the stress at this location is, therefore highly useful.

In all following analysis is the parameter $\frac{a_0}{2}$ considered to be $1\,\text{mm}$.

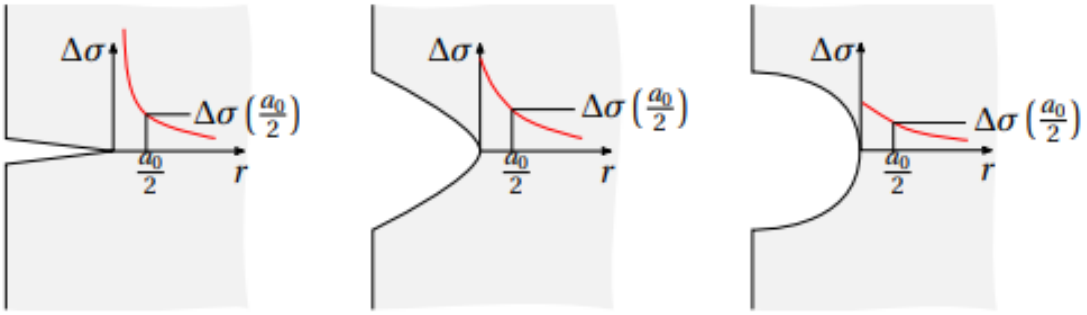


Figure 3.1: Critical distance approach illustrated by Berge and Ås (2017)

3.2 Basic Theory Machine Learning

Machine learning is a part of artificial intelligence and is the study of computer algorithms that automatically improve themselves through experience. A machine learning algorithm builds a model based on sample data, called ”training data”, to make predictions or decisions without being explicitly programmed to do so. A machine learning algorithm can find patterns in data sets with many parameters and predict a result.

In this section will the theory behind the machine learning used in this thesis be explained

3.2.1 Types of machine learning

Machine learning problems are often split into two types, classification, and regression. Classification models are models that output discrete variables, which may be used to sort things into categories, while regression models produce a continuous output variable. A classification model may be used to predict if it is going to rain or not, while a regression model may be used to predict how much it is going to rain.

In classification problems, we distinguish between binary and multi-class classification. Binary classification means we only have two categories, and in multi-class, we have more. This thesis only discusses using a binary classification used and the theory behind classification, focusing on the binary problems.

3.2.2 Artificial Neural Networks

The artificial neural network (ANN) is inspired by the biological neural network found in human brains. Like the human brain, it is based on a collection of connected units called neurons. The connections transmit signals between neurons, just like the synapses in the human brain. When a neuron receives a signal will it process the signal and then signal the neurons connected to it. In an artificial neural network are these signals real numbers are a function of the sum of the input signals to the node. (Mahanta, 2017)

3.2.3 Neurons

The neuron is the smallest entity in machine learning. Each artificial neuron has one or more inputs and produces a single output, and can send the output to multiple other neurons. The inputs may be the feature values of a sample of external data, such as images or documents, or the outputs of other neurons.

To find the output of the neuron are all the inputs weighted and summed. The input is a real number which is then multiplied with a weight and then is a bias term to this sum. This weighted sum is sometimes called the activation. This weighted sum is then passed through an activation function to produce the output. The initial inputs are external data, such as images and documents. (Mahanta, 2017)

3.2.4 layers

The neurons are structured into layers. There are many different types of layers with various features. One layer type may be well suited for image classification, and another may be well suited for regression. In the following sub-section will the layer types used in the experiments done in this thesis be described. (Mahanta, 2017)

Dense layer

The dense layer is the most frequently used neural network layer. In a dense layer is every input summed in the node and weighted by a weight function, generally followed by a non-linear activation function.

The dense layer stores information into 1-dimensional arrays, which are easy to handle but may lose information when dealing with images.

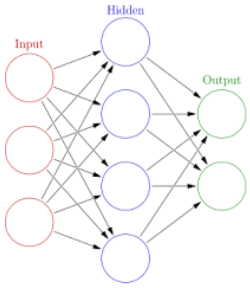


Figure 3.2: The structure of dense layers, illustrated by Wikipedia (2020)

Convolutional Layers

Convolutional layers are used when analyzing images. Using each pixel value and its placement in regards to other pixel values is the algorithm able to classify different types of animals, numbers, and more. In contrast to the dense layer will CNN's construct a layer that keeps the information between neighboring pixels, and instead of using weights and biases for each node, is a filter looking at a set of pixels. in Figure 3.3 is this illustrated. The filter, K, is multiplied with the matrix containing the value of all pixels. The filter is adjusted when training the model.

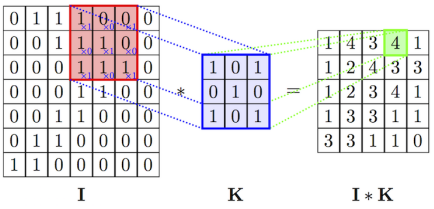


Figure 3.3: The convolutional layer illustrated by Nikhil (2020)

A parameter to note when using the Convolutional layers is the stride. Stride is a parameter of the neural network's filter that modifies movement over the image. For example, if a neural network's stride is set to 1, the filter will move one pixel, or unit, at a time.

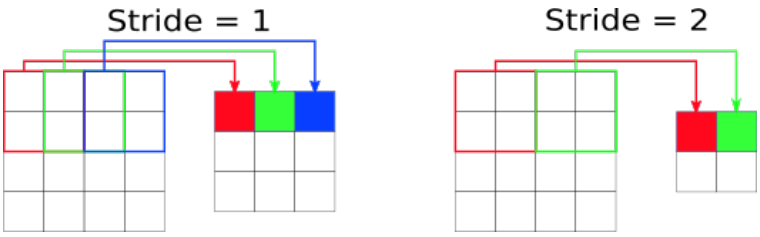


Figure 3.4: Effects of stride on convolutional layer

Padding is also a term relevant to convolutional neural networks. Padding refers to the number of pixels added to an image when the kernel of a CNN is processing it. For example, if the padding in a CNN is set to zero, every pixel value added will be of value zero.

On the other hand we have the zero-padding, which is adding a border with the value zero around the borders of the layer. If the zero-padding is set to one, will a one-pixel border be added to the image with a pixel value of zero.

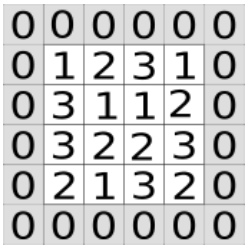


Figure 3.5: Example matrix with zero-padding

Transposed convolutional layer

Transposed convolutional networks are designed to allow for trainable upsampling, and a transposed convolutional layer produces an output feature map that is larger than the input layer. Like the convolutional layer, is the transposed convolutional layer also defined by padding and stride. It treats them as values that hypothetically were carried out on the output to generate the input. If you take the output and carry out a standard convolution with stride and padding defined, it will cause the spatial dimension same as that of the input.

A combination of convolutional and transposed convolutional layers is often used in encoder/decoders. Encoder/decoders use convolutional layers to downsample layers before utilizing transposed convolutional layers to up sample. To calculate the output layers of the two layers, may we use Table 3.1.

Table 3.1: Comparison between standard and transposed convolutional layer

Comparison					
Conv Type	Operation	Zero Insertions	Padding	Stride	Output Size
Standard	Downsampling	0	p	s	$(\mathrm{i}+2\mathrm{p}-\mathrm{k})/\mathrm{s}+1$
Transposed	Upsampling	$(\mathrm{s}-1)$	$(\mathrm{k}-\mathrm{p}-1)$	1	$(\mathrm{i}-1)^{*}\,\mathrm{s}+\mathrm{k}-2\mathrm{p}$

Pooling layers

Convolutional networks may include pooling layers to streamline the underlying computation. Pooling layers reduce the layer's dimensions by clustering the outputs from groups of neurons at one layer into a single neuron in the next layer. Local pooling combines small sets, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer. In addition, pooling may compute a max or an average. Max pooling uses the maximum value from a group of neurons at the previous layer. Average pooling ruses the average value from each of a cluster of neurons at the prior layer. (Singh, 2020)

Concatenate layer

The concatenate layer is used to connect a list of inputs. The inputs should be tensors of the exact dimensions except for the concatenation axis. Typically this means that the tensor has the same width and height and concatenate in depth. This results in the concatenate layer outputting one tensor with the same height and width as the two input tensors and a depth of the sum of the depths in the two inputs.

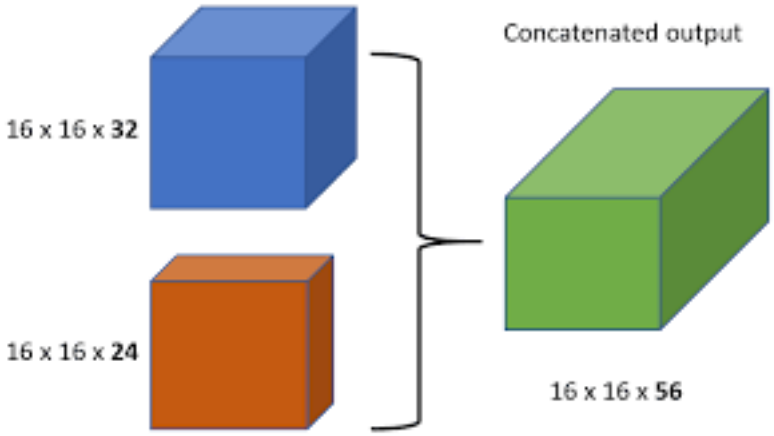


Figure 3.6: Concatenate layer illustrated by Adalogou (2020)

3.2.5 Generative adversarial networks

A generative adversarial network, GAN, is a machine learning framework designed by Ian Goodfellow and his colleagues in 2014. GANs contain two separate networks where one generates images, the generator and the discriminator, which tries to differentiate the generated images from real ones. We can guide the generator towards generating real-looking images by using the discriminator's ability to distinguish between the images as a measure of how well the generator performs. The architecture of a GAN network is illustrated in Figure 3.7.(Developers)

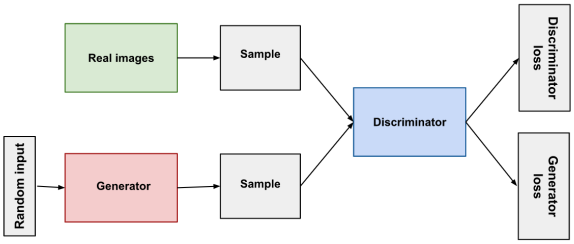


Figure 3.7: Example of GAN diagram illustrated by Google.

Generator

As mentioned is the generator the part of the GAN that creates new images from the input images. The goal of the generator is to make the discriminator classify its output as real. The generator is not trained to minimize the distance to a specific image but rather to fool the discriminator.

The generator is typically based on a convolutional neural network and uses filters to determine what is in the image and where it is.

Discriminator

The discriminator in a GAN is a binary classifier. The discriminator tries to differentiate between the data created by the generator and the ground truth output. There may be any form of network architecture.

The discriminator will train alongside the generator and improve its ability to distinguish between authentic and generated images. The training involves classifying both original and generated data. The

discriminator loss penalizes the discriminator for classifying the images. The discriminator weights are updated through backpropagation, just as another artificial neural network.

3.2.6 Activation functions

As mentioned in subsection 3.2.2 will the input values of a node be multiplied with their weight and summed. This sum is referred to as the activation of the node. This activation is then transformed with an activation function. The result of this function is the node output. A linear activation is known as the simplest of the activation functions. With this function, no transformation is applied, and the function returns the same value as the input. A model with this activation function is quickly trained. However, it cannot learn complex mapping functions. Therefore more complex functions are often used. Three non-linear functions are widely used. These are the older Sigmoid function and hyperbolic tangent function and the newer rectified linear activation function. Sigmoid and hyperbolic tangent functions mainly were used through the 1990s before the rectified linear function was introduced by Hahnloser et al. (2000).

ReLU

The Rectified Linear activation function, ReLU for short, is one of the most simple activation functions. The function will output the input directly if the input is positive and zero if the information is negative. The function is listed in Equation 3.3, and as a graph in Figure 3.8.

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{3.3}$$

Models using this activation function are often fast and accurate. This is why the ReLU activation function is the default activation function for many types of neural networks. There are many advantages to this activation function. However, the dying ReLU problem is one to have in mind when designing the network. The ReLU neurons may sometimes be pushed into states where they output 0 for virtually all inputs. This leads to no gradients flowing backward, and the neuron gets ”stuck” in an inactive state.(learning mastery, 2020)

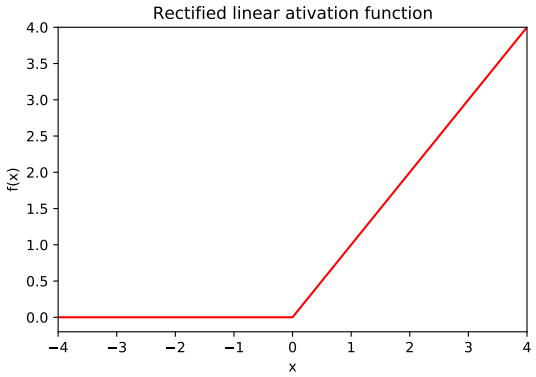


Figure 3.8: The Rectified Linear activation function

To correct the dying ReLU problem was the Leaky ReLU activation function introduced. The activation function have a slope $f(x) = ax$ where x is negative. Here $0 < a \ll 1$.

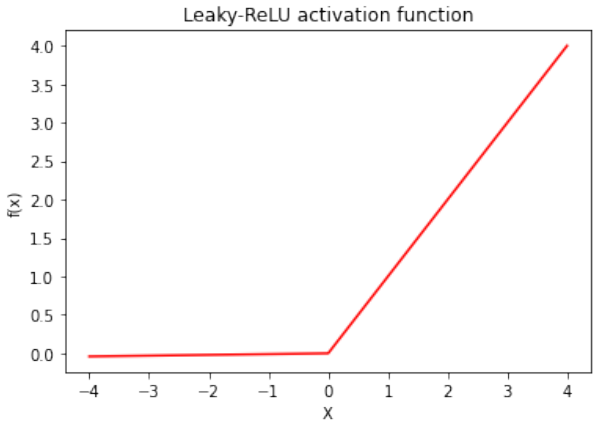


Figure 3.9: Leaky Relu

Sigmoid Activation

The sigmoid activation function is an S-shaped activation function that squeezes all values in between 0 and 1. The function is not centered around zero which may in some instances be a problem.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{3.4}$$

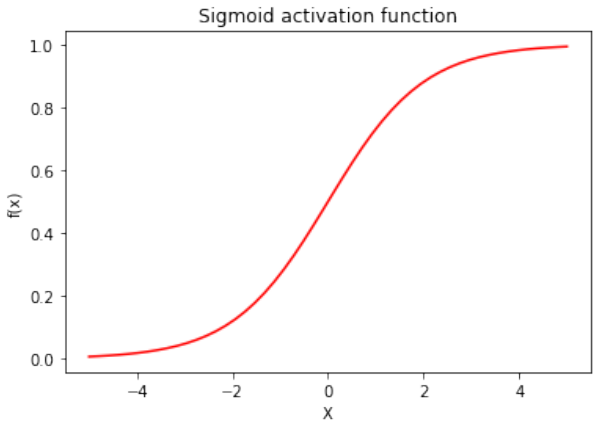


Figure 3.10: Sigmoid activation function

Tanh

The activation function is similar shaped to the sigmoid function, with outputs between -1 and 1. As we can observe in Figure 3.11 is the function zero centered. This overcomes the non-zero centering issue of the sigmoid function, this leads to a faster convergence than the sigmoid function and is therefore preferred.

$$f(x) = tanh(x) \tag{3.5}$$

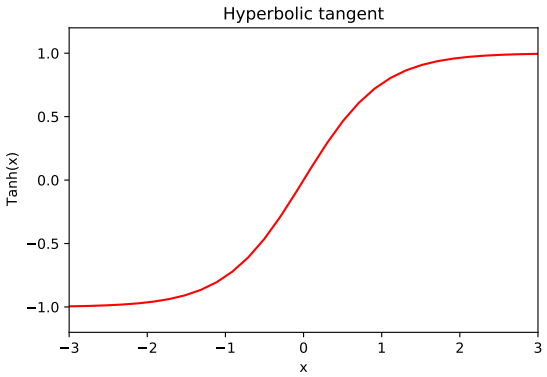


Figure 3.11: Tanh activation function

3.2.7 Loss functions used in regression problems

When the network has given an output, is it compared with the target value. A loss function gives a value for how good the network has performed. This loss function may calculate the accuracy of classification or the error of the regression. (Parmar, 2018)

In this subsection will some of the possible loss functions available when doing regression be discussed.

MAE

The mean Absolute error loss function is listed in Equation 3.6. The loss function is common in machine learning, however, it handles outliers very poorly.

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} | \, (y - \hat{y}_i) \, | \tag{3.6}$$

MSE

The mean square error loss function is listed in Equation 3.7. This loss function weighs larger errors more heavily than Mean Absolute Error and is thus sensitive to outliers. This loss function may also be referred to as L2, while MAE is referred to as L1.

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y - \hat{y}_i)^2 \tag{3.7}$$

RMSE

Root mean square error is the standard deviation of the errors which occur when a prediction is made on a data set. This very similar to MSE. However, the root of the value is considered while determining the accuracy of the model.

$$L(y, \hat{y}) = \sqrt{\frac{1}{N} \sum_{i=0}^{N} (y - \hat{y}_i)^2} \tag{3.8}$$

Pseudo-Huber Loss

The Huber loss combines MSE and MAE. The loss function uses MAE is larger than a predetermined value and MSE otherwise. This makes the Huber loss function more robust than MSE for outliers. However, this does not guaranty a perfect loss function.

$$\epsilon = \begin{cases} |y - \hat{y}|, & \text{if } |y - \hat{y}| \geq \alpha \\ (y - \hat{y})^2, & \text{otherwise} \end{cases} \tag{3.9}$$

3.2.8 Loss functions used for classification

Because of the discrete values, while solving classification problems, do we need other loss functions. The loss function uses the probability that the classification networks outputs and compares it with the label. Several different loss functions are used when solving classification problems. However, the most commons are the cross-entropy and the sigmoid cross-entropy.

Cross-Entropy

Cross-Entropy measures the performance of a classification model by giving a probability value between 0 and 1. The cross-entropy loss increases as the predicted probability diverge from the actual label. This means that if a model predicts a probability of 0.95 while the label is one will the loss be small. On the other hand, if the model predicts a probability of 0.2, will the loss be substantial.

In binary cases is Equation 3.10 used to calculate Cross-Entropy loss. Here p is the predicted probability and y is the binary indicator.

$$L = -(y \cdot log(p) + (1 - y) \cdot log(1 - p)) \tag{3.10}$$

Sigmoid Cross-Entropy

The sigmoid cross-entropy is the cross-entropy loss function that is run through the sigmoid activation function.

3.2.9 Optimizers

An optimizer updates the model in response to the output of the loss function. Optimizers assist in minimizing the loss function. There are several different optimizers used in machine learning. The goal of an optimizer is to reduce the losses in the network. When the network's loss is calculated is an optimizer used to update the neurons' attributes in the network.

Stochastic Gradient descent

Stochastic gradient descent optimizers, SGD, are the most basic and most used optimization algorithm. The method is heavily used both in linear regression and classification problems.

Gradient descent is a first-order optimization algorithm that depends on the first-order derivative of a loss function.

Momentum

Momentum is a method that accelerates SGD in the relevant direction and dampens oscillations. This is done by adding a fraction of the last vector to update the weights.

Momentum optimizing is best illustrated as a ball running down a hill. The ball accumulates momentum as it rolls down, just as the momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose slopes change direction. The momentum optimimzer results in faster convergence and reduced oscillations. The disadvantage of this method is that the hyperparameter needs to be selected manually and accurately.

RMS-Prop

RMS-Prop keeps the moving average of the squared gradients for each weight and divides the slope by the mean square's square root. The update rule looks like this Equation 3.11.

$$\begin{aligned} E\left[g^2\right]_t &= 0.9E\left[g^2\right]_{t-1} + 0.1g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \end{aligned} \tag{3.11}$$

The optimizer creator, Geoffrey Hinton, suggests $\gamma = 0.9$, with a good default for η as 0.001. (Bushaev, 2018)

ADAM - Adaptive moment estimation

Adam(Adaptive moment estimation) computes adaptive learning rates for each parameter. The method stores an exponentially decaying average of past squared gradients v_t like RMS Prop. Adam stores an exponentially decaying average of past gradients m_t, which is similar but not equal to the momentum optimizer. If the momentum optimizer is viewed as a ball running down a slope is the Adam optimizer closer to behaving as a heavy ball with friction. The friction makes it easier for the optimizer to stop in the flat minimum and not oscillate around the minimum. m_t and v_t is found as follows. (Katba)

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{3.12}$$

3.2.10 Normalization

Normalization is a part of the prepossessing of input data. The goal of the process is to have all values be on a standard scale without losing their accuracy. When normalization is not used are often the largest values more influential to the model than the small ones.

3.2.11 Backpropagation

In machine learning, is backpropagation referring to a process used as a method of training. With the backpropagation is loss from the loss functions and the gradients from the optimizers used to update the trainable parameters of the network. (McGonagle et al., 2020)

3.2.12 Epochs

The total number of epochs is defined as the number of times one input is run through the machine learning model. Having many epochs means that machine learning may train more on the same sample. Balancing the number of epochs is therefore highly important. Too few epochs will not train the model sufficiently, and too many epochs will overtrain the model. An over-trained model has learned the data set "by heart" and will not be as adaptable as a not over-trained model.

3.2.13 Batch Size

The batch size refers to the number of training images used in one iteration. Having a small batch size reduces required memory. However, a larger batch size may decrease computational time.

3.2.14 UINT gray scale images

The UINT8 and UINT16 png data types will be referenced to several times in this thesis. UINT refers to the unsigned integer value used in each pixel. The pixel value represent the colour in the pixel where 0 is black and the max value is white. UINT8 and UINT16 differs from each other as UINT8 uses 8-bit pixels and UINT16 uses 16-bit pixels. The 16-bit requires more storage space but have a higher max value. The max values in UINT8 is 255 and max value in UINT16 is 65536.

Chapter 4

Method

4.1 Creating Data set

To develop a data set that could be used to train a machine learning model was a considerable number of FEM analysis done, outputting stress in the surface and $1\,\mathrm{mm}$ below the surface. In this section is the process of developing the data sets used for machine learning described.

4.1.1 Data set of pits

Marius Andersen provided a data set containing over 63000 .png images representing the topology of pits in mooring chains. The images are the result of laser scans of mooring chains. The images of the pits are located such that the stress is in the image x-direction. The pit depth distribution of all the images is shown in Figure 4.1. From the original 63000 images are 4000 chosen at random and used when making data sets.

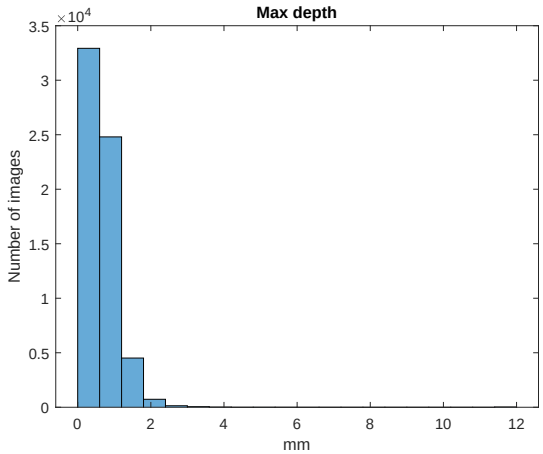


Figure 4.1: Distribution of pit depths

4.1.2 Creating Geometry

A Matlab script provided by Sigmund K. Ås was modified and used to create STL geometries of the 240x240 px topology images provided by Marius Andersen. The parameters used to create the geometries are listed in Table 4.1. The STL files created are 60x60x50 mm^3 with pits with a max depth of $11.9\,\mathrm{mm}$.

Table 4.1: Pixel values used in input images

Parameter	Value
Pixel width	$0.25\,\mathrm{mm}$
Pixel height	$0.25\,\mathrm{mm}$
Pixel depth	$0.047 \cdot V_{pixel}\ \mathrm{mm}$

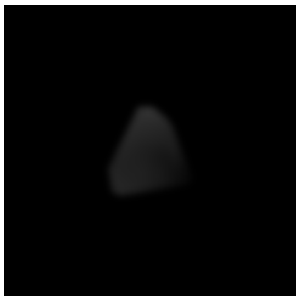
Changed input images

Because the data set of scanned pits in mooring chains was observed to have relatively small peak stresses, was this method updated. To help make the training of the model more diverse was 1000 pits scaled up with a factor of two, and 1000 scaled up with a factor of three. Because of the upscaling of the images were the images provided by Marius Andersen adjusted. The data format of the images was changed from UINT8 to UINT16 and the pixel value of the images was changed such that the value of the pixel represents the depth at that particular point in µm. This means that a pixel value of 1000 represents $1\,\mathrm{mm}$ and the deepest pit possible to represent with UINT 16 will be $65.536\,\mathrm{mm}$. This was chosen to make the pixel values more intuitive and easy to use.

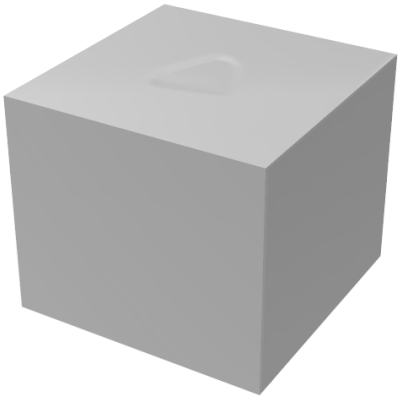
Because the machine learning method needs the size of the input images to be dividable by two are the input images also scaled to the size 256x256px. The properties of each pixel listed in Table 4.2.

Table 4.2: Values used when converting image to STL

Parameter	Value
Pixel width	$0.234\,\mathrm{mm}$
Pixel height	$0.234\,\mathrm{mm}$
Pixel depth	$V_{pixel}\ \mu\mathrm{m}$



(a) Image of PIT



(b) STL FILE

Figure 4.2: STL conversion

4.1.3 Meshing

The ANSYS program ICEM-CFD is used to prepare the STL for FEM analysis. The repair geometry function is used to create a volume of the faceted geometry. This tool automatically closes gaps between boundary edges using the best fit possible. To make the most accurate model possible, is tolerance set to 0.001 in this process, which is $\frac{1}{10}$ of the default setting. ICEM-CFD easily differentiates the edges from each other, and each edge may be defined as a part. First, is the surface containing the pit defined

as a part., then is a copy of the part made and translated $1\,\mathrm{mm}$ down. With the defined parts, it is easy to select and export stress in specific locations. Lastly is the structure meshed by tetrahedral elements. The element type and material properties are defined later when the model is analyzed in ANSYS. The mesh size is chosen to vary between $1\,\mathrm{mm}$ at the surface and $10\,\mathrm{mm}$ at the bottom of the model. The varying mesh is to reduce computational time while still maintaining accuracy and preserve the surface topology. Lastly is the meshed structure exported as an .in file, which is used in ANSYS.

While preparing the model for meshing, the record script function in ICEM-CFD was used, and a replay script was created. The script is used when running the program in batch mode and lets us mesh one STL file after another. The whole process takes around $200\,\mathrm{s}$ to complete per STL file.

The replay script is found with the attached files, however running the replay script on another computer may be difficult. Therefore is a detailed walk-through of the procedure of preparing the STL file for FEM analysis is found in Appendix C.

(a) STL-File in ICEM-CFD

(b) Finished meshed model

Figure 4.3: Model in ICEM-CFD

4.1.4 FEM-Analysis and output

The elements and parts are imported to Ansys APDL with the .in file. All elements are then given the same material parameters and element type. A temperature of $0\,^{\circ}\mathrm{C}$ is defined for the whole system.

The Element type used in this analysis is SOLID185, an eight-node element where each node has three degrees of freedom. SOLID185 Structural Solid is suitable for modeling general 3-D solid structures. It allows for prism, tetrahedral, and pyramid degenerations when used in irregular regions. Various element technologies such as B-bar uniformly reduced integration and enhanced strains are supported. When meshing the structure are the elements are given a prism shape. (ANSYS)

Table 4.3: Material properties used in FEM analysis

Parameter	Value
E module	200 GPa
Poisson's ratio	0.3 [-]

In APDL are boundary conditions and loads applied. The analysis takes about 100 s to complete.

Table 4.4: Applied boundary condition

Location	BC
(0,y,z)	u=0
(x,y,0)	v=0
(x,y,0)	w=0

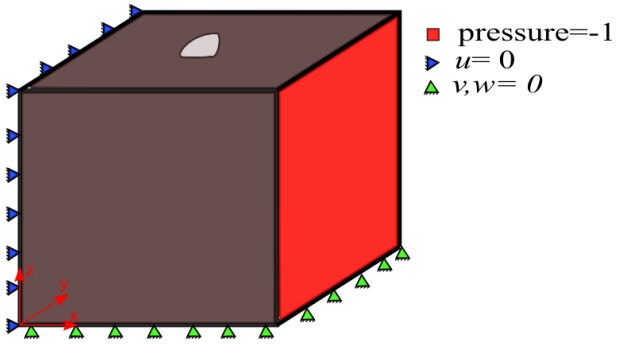
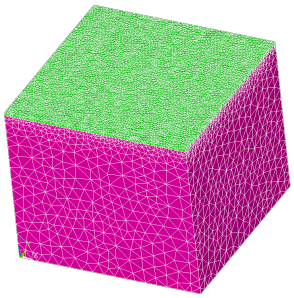
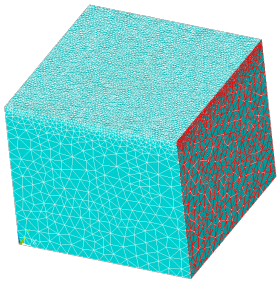


Figure 4.4: Illustration of load and BC



(a) Meshed model in APDL



(b) Meshed model with applied pressure

Figure 4.5: Model in ANSYS

The first principle stress in the surface and 1 mm nodes are collected after the analysis is finished. A Matlab script uses the node X and Y coordinates and stress to make a 256x256 matrix with columns and rows corresponding to the X and Y coordinate and cell value with the average stress of the nodes in the area. The cell values are then multiplied by 1000 and saved as a 16-bit grayscale .png. These images are illustrated in Figure 4.7a and Figure 4.7b. The pixel value in these images are scaled to have the largest values be white and the lowest to be black. This is done to show the different values of stress. The stress is multiplied by 1000 to maintain accuracy of 0.001 when saving the value as an integer.

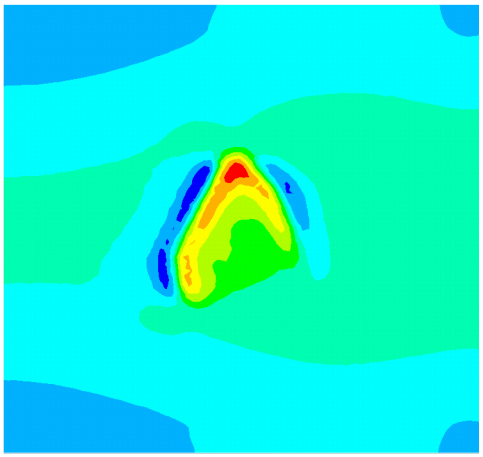
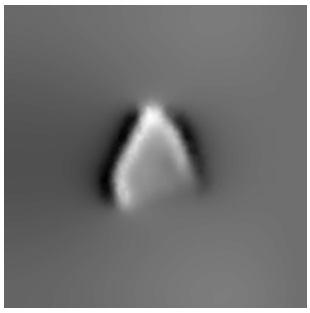
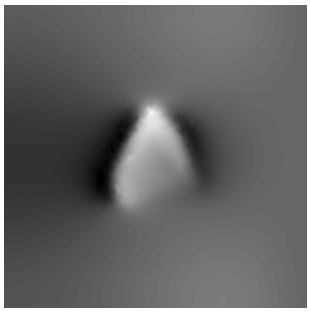


Figure 4.6: Stress on surface



(a) Stress surface



(b) Stress 1mm below surface

Figure 4.7: Stress-images

4.1.5 Batch mode

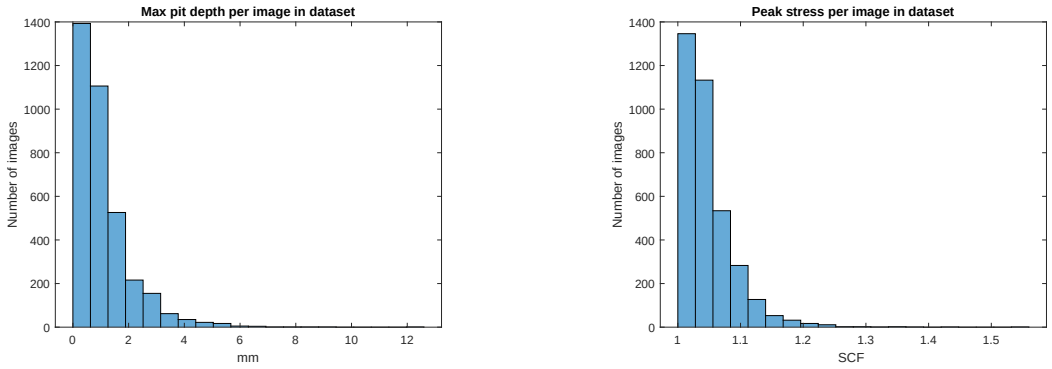
To generate a large data set is it necessary to automate the process of developing stress images. The Matlab script Generate_data.m is designed to do this. The algorithm of this program is found in algorithm 1.

Algorithm 1: Algorithm for crating data set

```
for i ← first stl file to last stl file do
    STL Filename← num2str(i) + ".stl" ;
    if STL Filename is file then
        Copy STL-filename to "STL-FILE" ;
        run ICEM-CFD batch mode ;
        run ANSYS APDL batch mode ;
        if Results files exist then
            Make image of stress field in surface ;
            Make image of stress filed 1mm below surface ;
            Copy Result files to archive folder;
            Delete Result files ;
        Delete "STL-FILE" ;
```

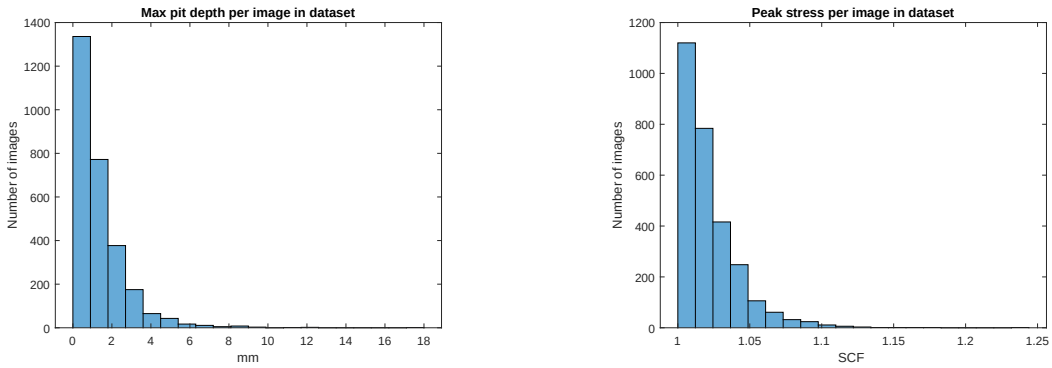
4.1.6 Data set for training

Because of the extra images is the data set shuffled before it is split into the validation, test, and training groups. The surface data set contains a total of 3586 images, while the data set with stresses $1\,\mathrm{mm}$ below the surface consist of 2816 images. Compared to the complete pit data set is the pit depth distributions in both data sets shifted towards the deeper pits. The two data sets will be referred to as the surface and subsurface data set.



(a) Pit depth distribution **(b)** Peak stresses histogram

Figure 4.8: Surface data set distribution

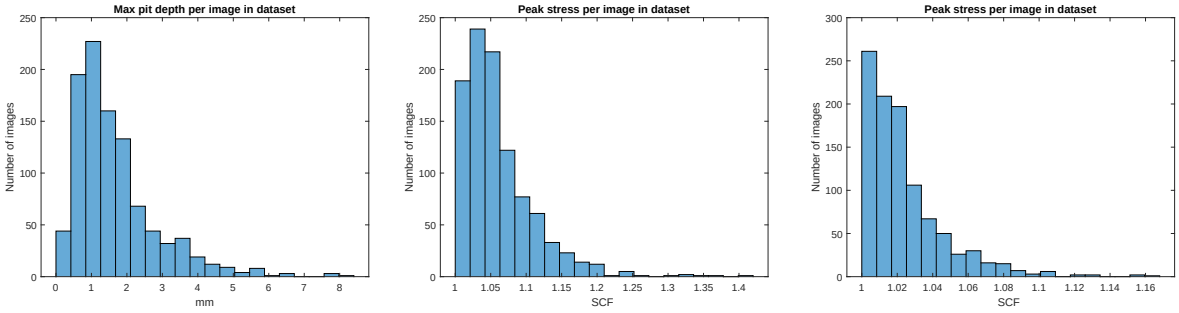


(a) Pit depth distribution **(b)** Peak stresses histogram

Figure 4.9: Subsurface data set distribution

4.1.7 Validation data set

In addition to the training data set is a validation data set created. This data set consists of 1000 pits. These pits will not be used in the training process and are only used to test the finished models.



(a) Pit depth distribution **(b)** Peak stresses in surface **(c)** Peak stresses 1mm below surface

Figure 4.10: Validation data set distribution

4.2 Method for Image to image translation

Previous experiments found that a convolutional neural network can predict the peak stresses in a pit accurately. Finding stress fields, however, hasn't been done previously. To do this does is one matrix of depths to be translated into a matrix of stresses. Isola et al. (2017) developed a machine learning algorithm able to translate one picture to another. The model is very versatile and can solve problems ranging from colorizing images to making maps from satellite images. Because of the model's versatility, we are hoping that the code may be used to estimate stresses in and around a pit.

The pix2pix model will be presented in this section, along with the changes that have been made to the model.

4.2.1 Code type

There are several pix2pix versions available for use, with the program written in different languages and using different libraries. Before deciding on the python version using the Tensorflow library was the Lua language using the torch library and python using Pytorch library tested. The choice fell on the code using the Tensorflow library. This code was chosen as it easy to read and make the necessary changes. The original code can be found and downloaded from GitHub[1].

4.2.2 Data set

To group input and output images, are they combined into one image. This is done with combine_images.m, which is found with the attached files. The input image is placed in the A position, and the output is placed in the B. The input image is scaled from 240x240 to 256x256, and the pixel values are scaled. The resulting image is a 512x256 .png image with datatype UINT16.

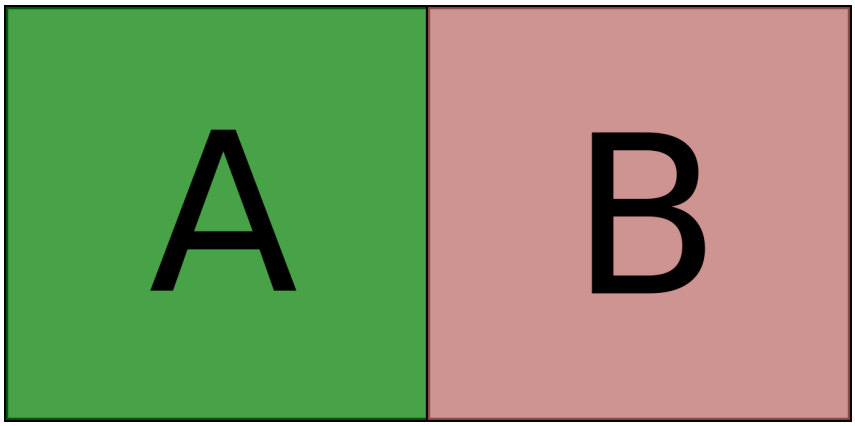


Figure 4.11: Combined Data

This is done with both data sets described in subsection 4.1.6 before the images are shuffled. Then are the data set split into three parts, training, validation, and test data sets. The validation and test data sets each contain 100 images, and the remaining images are in the training data set.

4.2.3 Preprocessing

The data set is preprocessed for each epoch. The original program uses a function they refer to as random jitter for preprocessing. Random jitter is a concept where each image is scaled up to 286x286 px and then

[1] https://github.com/affinelayer/pix2pix-tensorflow

crops a random patch of 256x256 pixels of the image to use for training. Because all pits are centered in the images, will this not be done in the experiments. There will be no random jitter, and the whole image is used for training.

The original preprocessing process also includes a random rotation of the image. Because the images show stress in plane tension in the x-direction, this is also not done in these experiments. However, a feature that flips 50% of the images horizontally is kept. This is because of the plane tension in the x-direction that will give similar stress fields if the pit is mirrored in the x-direction.

4.2.4 Generator architecture

The Tensorflow library handles data in the generator as tenors of rank 3. Shapes of tensors are referred to as (width, height, depth) where the depth in the image represents the number of channels in the image. The original model is built to handle RGB images that uses three channels, one for each color. The number of channels in the input and output layer is reduced to 1. This results from the training being done on gray scale images rather than three channel RGB images. The input and output layers have the shape (256,256,1).

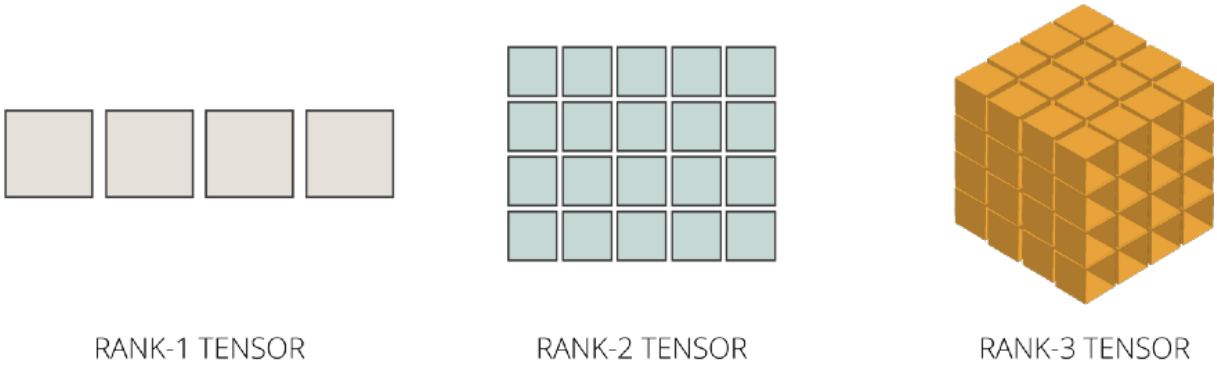


Figure 4.12: Rank of tensors illustrated by G. Yalçın (2020)

The model analyzes the images by reducing the width and height of the tensor while increasing the depth until the width and height of the tensor are 1. Then the process is reversed such that the width and height of the layers increase to the original size. This process is illustrated as encoder-decoder in Figure 4.13. The downsizing finds ”what” is in the image, and the upsizing finds where it is.

To give the generator a means to keep information from the original image, skip connections are added. This gives the U-net shape shown in Figure 4.13. Specifically, skip connections are added between layer i and layer n -i, where n is the total number of layers. Each skip connection concatenates all channels at layer i with those at layer n -i. The skip connections use the leaky ReLU activation function between each other.

The original generator uses filters with size 4x4 and a stride length of 2 in all convolutional and transpose convolutions layers. The whole generator architecture is listed in Table 4.5.

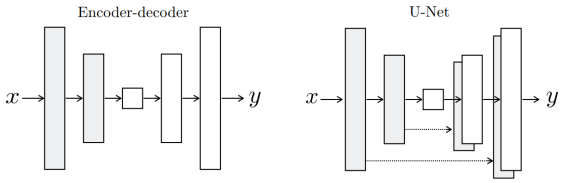


Figure 4.13: Architecture principle illustrated by Isola et al. (2017)

Table 4.5: Generator architecture

Layer	Input Shape	Output Shape	Layers	Layer type	Sends to	Receives from
1	(256,256,1)	(256,256,1)	64	Input	-	-
2	(256,256,1)	(128,128,64)	128	Convolutional	23	-
3	(128,128,64)	(64,64,128)	256	Convolutional	21	-
4	(64,64,128)	(32,32,256)	512	Convolutional	19	-
5	(32,32,256)	(16,16,512)	512	Convolutional	17	-
6	(16,16,512)	(8,8,512)	512	Convolutional	15	-
7	(8,8,512)	(4,4,512)	512	Convolutional	13	-
8	(4,4,512)	(2,2,512)	512	Convolutional	11	-
9	(2,2,512)	(1,1,512)	512	Convolutional	-	-
10	(1,1,512)	(2,2,512)	512	Convolutional	-	-
11	(2,2,512)	(2,2,1024)	-	Concatenate	-	8
12	(2,2,1024)	(4,4,512)	512	Transpose Convolutional	-	-
13	(4,4,512)	(4,4,1024)	-	Concatenate	-	7
14	(4,4,1024)	(8,8,512)	512	Transpose Convolutional	-	-
15	(8,8,512)	(8,8,1024)	-	Concatenate	-	6
16	(8,8,1024)	(16,16,512)	512	Transpose Convolutional	-	-
17	(16,16,512)	(16,16,1024)	-	Concatenate	-	5
18	(16,16,1024)	(32,32,256)	256	Transpose Convolutional	-	-
19	(32,32,256)	(32,32,512)	-	Concatenate	-	4
20	(32,32,512)	(64,64,128)	128	Transpose Convolutional	-	-
21	(64,64,128)	(64,64,256)	-	Concatenate	-	3
22	(64,64,256)	(128,128,64)	64	Transpose Convolutional	-	-
23	(128,128,64)	(128,128,128)	-	Concatenate	-	2
24	(128,128,128)	(256,256,1)	1	Transpose Convolutional	-	-

4.2.5 Discriminator

The discriminator is a Patch GAN network. It has two input layers, one using the generator's input image and one with either the target image or the generated image. Each input is a (256,256,1) tensor, and the two are concatenated together into a (256,256,2) tensor.

The output of the model is a (30,30,1) tensor containing values between 0 and 1, where values close to 0 indicate produced images and values close to 1 indicate target images.

As discussed in section 3.2.5 will the discriminator try and distinguish real images from the generated images. The discriminator helps the model find the most real-looking images. Blurry-looking images

will not be accepted by the discriminator, which leads the generator to avoid producing blurry images.

Table 4.6: Discriminator Architecture

Input 1 Shape		Input 2 shape		
(256,256,1)		(256,256,1)		
Concatenate				
Layer	Input Shape	Output Shape	Filters	Layer Type
1	(256,256,2)	(128,128,64)	64	Sequential
2	(128,128,64)	(64,64,128)	128	Sequential
3	(64,64,128)	(32,32,256)	256	Sequential
4	(32,32,256)	(34,34,256)	-	Zero Padding
5	(34,34,256)	(31,31,512)	512	Convolutional
6	(31,31,512)	(33,33,512)	-	Zero Padding
7	(33,33,512)	(30,30,1)	1	Convolutional

The architecture has shown to produce accurate results and we will therefor keep the architecture as it is.

4.2.6 Activation functions

Between the layers are leaky ReLU between the convolutional layers. The last layer uses the hyperbolic tangent as activation function. The last activation function makes sure the output values are between -1 and 1, which fits well with the normalization used. As the two activation functions are widely used and work well with the training, no other activation function is looked into.

4.2.7 Loss functions

Generator loss

The loss function of the pix2pix can be found in Equation 4.1. The loss function consists of two parts, the mean absolute error($\lambda\mathcal{L}_{L1}(G)$) and the generator error (L_{cGAN}). The generator loss is a two-dimensional variant of the sigmoid cross-entropy that uses the discriminator output to score the generator's performance.

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda\mathcal{L}_{L1}(G). \tag{4.1}$$

λ is a value used to weigh the MAE loss, and the original model uses $\lambda = 100$. The loss function has shown to give very accurate results, however, it may not be the perfect choice when estimating the stress field around a corrosion pit. We may observe in Figure 4.7 that the average value of the stress fields images are dominated by the area around the pit. This may reduce the models' ability to estimate the highest peak stresses.

To investigate the impact of the loss function are several models are trained, and the accuracy of the models are analyzed. This process is described in subsection 4.3.1.

Discriminator loss

The discriminator uses the sigmoid cross entropy as its loss function when classifying between the real and generated images. The loss function evaluates the two outputs from the discriminator by comparing

the result from the target image with a (30,30,1) tensor of all ones and the generated image with a tensor of the same size containing all zeros. This has proven to be a very effective means of training the discriminator and will not be changed.

4.2.8 Optimizers

The code uses the mini-batch SGD and the application of the ADAM optimizer to update both the image generator and the discriminator. The learning rate is set to 0.0002 and momentum parameters of β_1 = 0.5 and $\beta_2 = 0.999$.

Because the ADAMS optimizer has been shown to give fast convergence, will there be made no changes to the optimizer.

4.2.9 Batch Size in the model

As discussed in subsection 3.2.13, will a larger batch size lessen the learning time of the machine learning model. However, the pix2pix paper suggests a batch size between 1 and 10. This is due to the fact that larger batch sizes have been shown to give poorer results. To save computational time will a batch size of 4 be used when experimenting with different models and then use a batch size of 1 when training the final models. Because multiple models are going to be trained, is the reduction in computational while still maintaining accuracy important. The batch size of 4 will give us an idea of how the different models differ from each other and not take up too much time.

4.2.10 Normalization

Because the hyperbolic tangent activation function is used as the activation function before the output layer, do we need to normalize the data set to fit between -1 and 1. As a consequence of this do we need to normalize the data. The original model uses Equation 4.2 to do this. However, to fit the 16-bit pixel value is Equation 4.3 a better option.

$$V_{Norm} = \frac{V_{Original}}{\frac{255}{2}} - 1 \tag{4.2}$$

$$V_{Norm} = \frac{V_{Original}}{\frac{2^{16}}{2}} - 1 \tag{4.3}$$

This way of normalization will include all possible values of UINT16. However, we may observe that the pit depths used in the model range from 0 to 18 mm. This means that the maximum possible input value is 18000. Therefore may the normalization of Equation 4.4 be a better option when training on this specific data set. On the other hand is an adaptable model desired, and using Equation 4.4 will limit the models' ability to analyze deeper pits.

$$V_{Norm} = \frac{V_{Original}}{\frac{18000}{2}} - 1 \tag{4.4}$$

4.2.11 Training the model

Computer resources

The training is done in the web-based python notebook google Colaboratory, colab for short, which is a product from Google Research. When running code in Google colab is the code executed in a virtual machine private to the account running. Google colab is chosen as the environment to train the models in as they give their users access to GPUs, TPUs, and high RAM computers. The GPUs available in Colab

often include Nvidia K80s, T4s, P4s, and P100s.

One thing to keep in mind is that virtual machines are deleted when idle for a while, which may result in the termination of the code running.

Machine learning algorithm

The machine learning algorithm used is the one presented by Isola et al. (2017). Algorithm 2 shows how the training is performed. The whole script may be found online in google colab[2].

Algorithm 2: Algorithm for Machine learning

```
for epoch← 1 to end do
    Start timer;
    Generate and show example image;
    for image ← 1 to end do
        predicted image ← generator(input image);
        real discriminator output ← discriminator(target image);
        predicted discriminator output ← discriminator(predicted image);
        discriminator loss ← calculate discriminator loss(real discriminator loss,predicted
          discriminator loss);
        generator loss(predicted discriminator output,generated image, target image) ;
        calculate generator gradients ;
        calculate discriminator gradients;
        update generator;
        update discriminator;
    if modulus(Epoch,20) == 0 then
        Save checkpoint;
    End timer ;
    Print time ;
```

[2] https://colab.research.google.com/drive/1Wl462lHNyoYAp2tliSd-9m1JoQ0_SliU?usp=sharing

4.3 Experimentation with the machine learning method

Due to the limited experience with machine learning, it is deemed necessary to experiment with model parameters. To find the best way of training a model was a number of models trained. During this process was three different error functions used and three different model sizes.

The normalization of the model is not experimented with. This is because we want the model to be as adaptable as possible. Lowering the max values used in the normalization will limit what kind of pits we may use in the model.

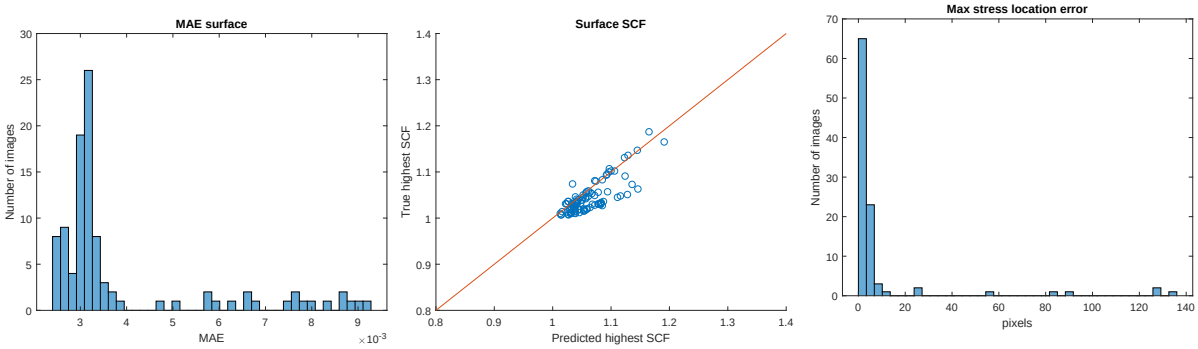
To compare the different trained models is MAE, MSE, and the value and location of the peak stress for the same 100 images in the test data sets used as a reference. MSE and MAE give us an idea of how similar the images are to each other, however, we are most interested in the errors in peak stresses.

4.3.1 Loss function

As described in subsection 4.2.7 will the loss function define what we are looking for when training. As discussed in subsection 1.2.2 are we most interested in predicting the highest SCF accurately. Isola et al. (2017) states that models trained with loss functions not using a generator loss term usually produce results that have indistinct edges and look blurry. Because of this is the generator loss term kept and not adjusted. To find the best option for loss function is three models trained, one using MAE, one using MSE, and one using the RMSE.

Pseudo-Huber loss and the adaptive loss functions are not experimented with. This is because the loss function is complicated and challenging to implement and scale with generator loss.

MAE

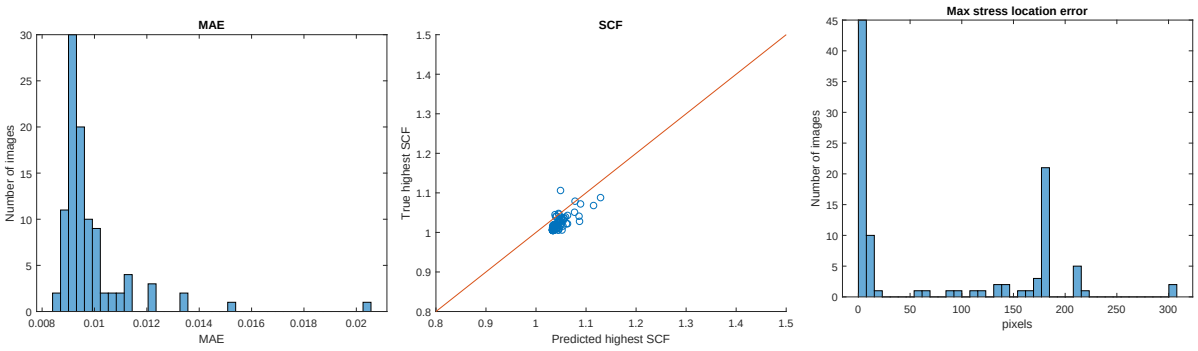


(a) MAE distribution

(b) Accuracy of peak stress value

(c) Distance between GT and predicted max stress

Figure 4.14: Accuracy of surface model trained with MAE



(a) MAE distribution

(b) Accuracy of peak stress value

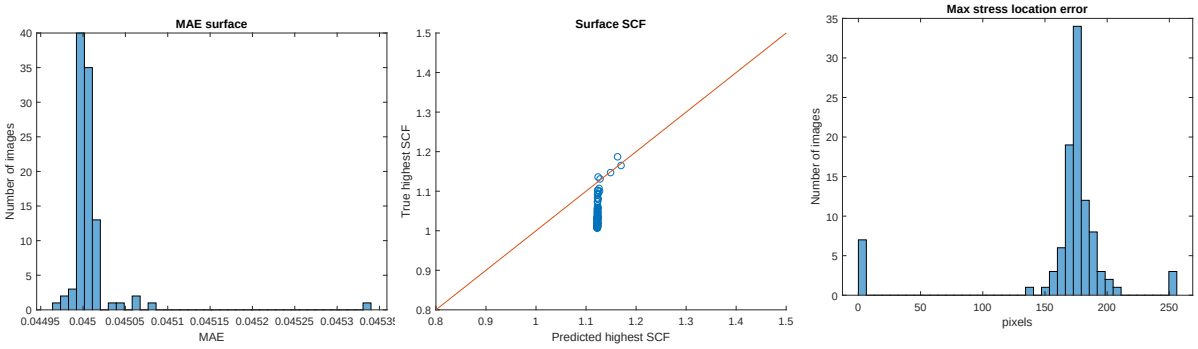
(c) Distance between GT and predicted max stress

Figure 4.15: Accuracy of subsurface model trained with MAE

MSE loss function

As discussed in subsection 4.2.7 are we interested in including MSE in the loss function rather than MAE. However, finding an appropriate λ value proved to be a challenge. Four models were trained with $\lambda \in [0.1, 1, 10, 100]$. The best result was found when using $\lambda = 10$. Still, the results were not very accurate. Because the best result didn't come while training on either $\lambda = 0.1$ or $\lambda = 100$ we will not continue experimenting with the value of λ. There is also not necessary to train a model with the subsurface data set as that data set has shown to train a model that performs poorer than the surface data set.

Because we did not achieve good results with the surface data set is it no point in training a model with the more challenging subsurface data set, and this is not done.



(a) MAE distribution

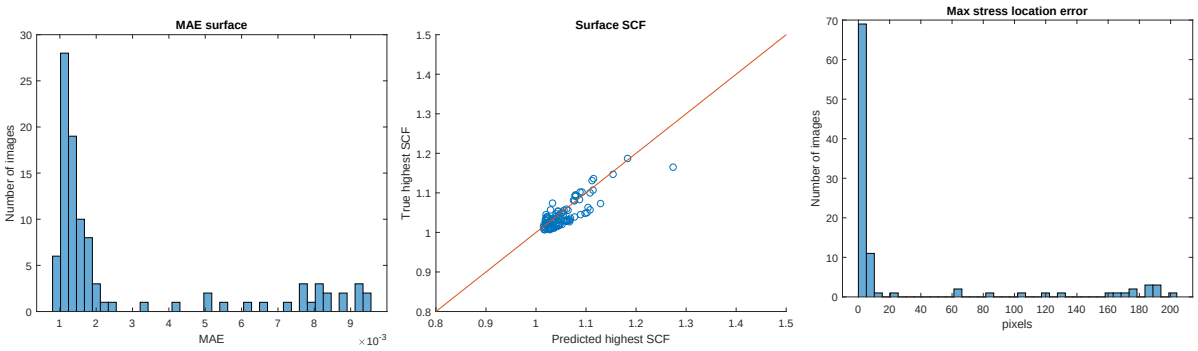
(b) Accuracy of peak stress value

(c) Distance between GT and predicted max stress

Figure 4.16: Accuracy of surface model trained with MSE

Root Mean Square Error

Lastly was a model trained with RMSE as the loss function. This was done because no appropriate λ was found when experimenting with MSE, and the model produced no accurate results. RMSE will be in scale with the original loss function while still weighting the outliers heavily. As the loss function is in scale with MAE is λ set to 100.



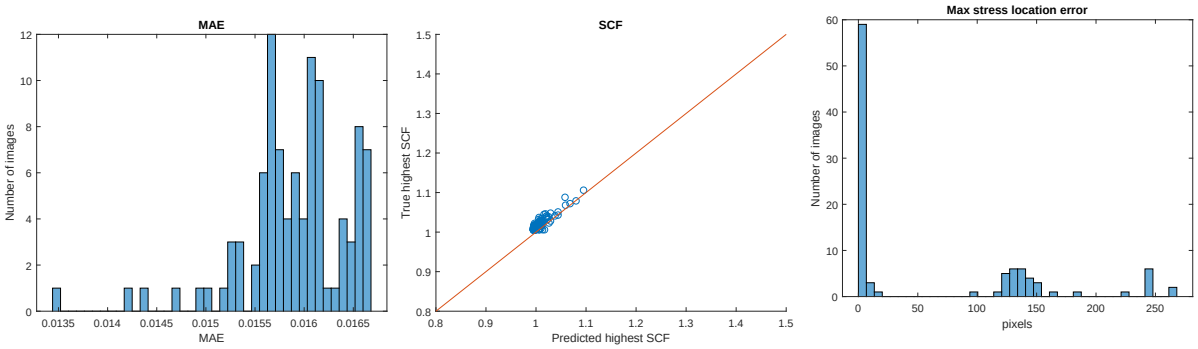
(a) MAE distribution

(b) Accuracy of peak stress value

(c) Distance between GT and predicted max stress

Figure 4.17: Accuracy of surface model trained with RMSE

Because the predicted images from the training proved accurate, is a model trained on the subsurface data set.



(a) MAE distribution

(b) Accuracy of peak stress value

(c) Distance between GT and predicted max stress

Figure 4.18: Accuracy of subsurface model trained with RMSE

4.3.2 Experiments with model architecture

The pix2pix architecture is extensive and designed to handle very complex images with a lot of details. The large model causes the training to be time-consuming, and the checkpoint files large, over 500MB each. The images used in the experiments in this thesis are not as complex or detailed as those used in Isola et al. (2017), and we should thus examine if we may reduce the size of the model. We continue using the RMSE loss when training the model.

Removing trainable parameters may reduce the accuracy of the model. However, a smaller model also reduces the chance of overfitting the model. This process aims to find how we can reduce the size of the model while still having the model predicting sufficiently accurate results.

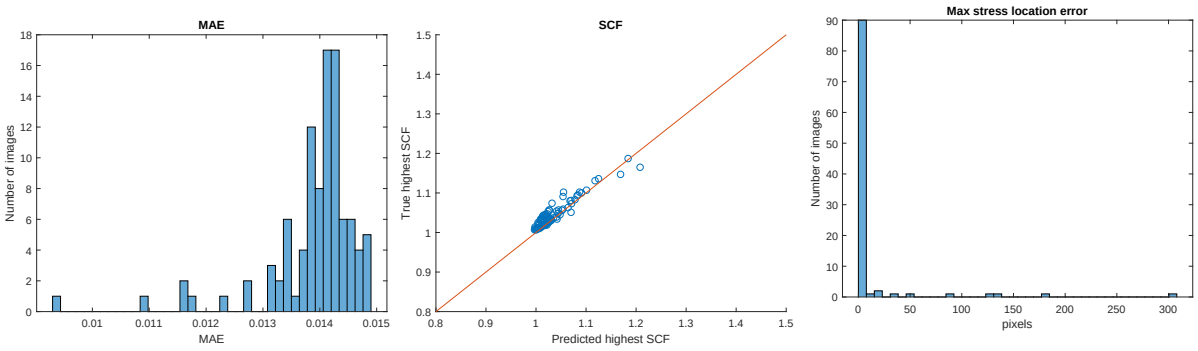
To examine the use of a smaller model was two different reductions were proposed. One where the center layers, 9,10,11 and 12 from Table 4.5, were removed, and one where the CNN strides were increases from 2 to 4. The RMSE model trained in the previous chapter was used as a reference to measure how the reduction of the model affects the results, storage use, and time consumption. To judge the models on similar grounds will a batch size of 4 be used when training the models.

Model with center layers removed

By removing the four center layers the model, we remove 8704 trainable nodes and 1536 filters. This results in the file containing the model data being reduced to 366MB. That the layers with the smallest dimensions have are (4,4,512). As discussed in subsection 4.2.4 may removing the middle layers restrict the models' ability to find "what" is in the image.

Each epoch takes 70 seconds when training the reduced model, which reduces the 85 seconds it took to train the original model. As predicted does the reduced model saves some computational time and predicates the value and placement of SCF accurately. We may observe that the model gives some higher MSE and MAE losses than the original model. However, the reduction of the model size shows some potential.

To experiment further was another experiment performed with a model where six layers were removed. The model did not produce good results and thus was no more experimentation done on models with center layers removed.



(a) MAE distribution

(b) Accuracy of peak stress value

(c) Distance between GT and predicted max stress

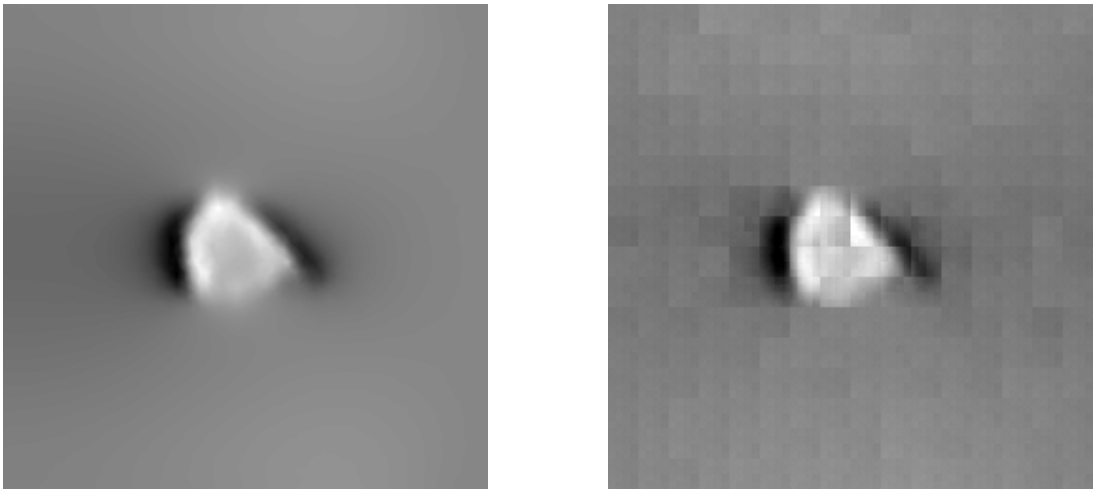
Figure 4.19: Accuracy for model with middle layer removed

Using stride length of 4

Another method of reducing the model size is to increase the stride length in the convolutional layers when up and downsizing. Doubling the stride length from 2 to 4 will cause the downsizing to reduce the layer to $\frac{1}{16}$ of the size of the previous, and the number of layers in the model architecture becomes half the number of the original architecture.

A stride of 4 means that each 4x4 filter will only influence each pixel once. This results in the predicted image being visually split into squares of 4x4 pixels. Figure 4.20 shows how the image is split into chunks. To try and solve this issue was an experiment was performed using 8x8 filters. This model with larger filters performed worse than the model with 4x4 filters, and 4x4 filters were accepted as the best choice.

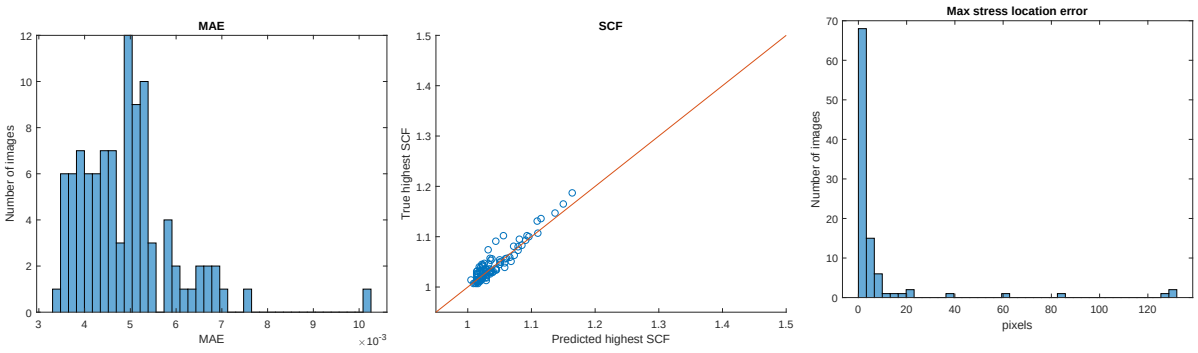
The model uses 25 seconds per epoch which is very fast compared to the previously trained models. As we may observe in the loss plots in Figure 4.21 will the model also gives very accurate results.



(a) Scaled Ground Truth

(b) Scaled Predicted Image

Figure 4.20: Comparison between predicted image and ground truth with stride = 4



(a) MAE distribution

(b) Accuracy of peak stress value

(c) Distance between GT and predicted max stress

Figure 4.21: Accuracy for model with stride 4

4.3.3 Conclusions from experiments

Loss functions

The L1 and RMSE model perform very similar. However, we observe that the RMSE model performs just slightly better when estimating the value and location of the highest stress value. This becomes especially clear when training the model on the subsurface dataset. On the other hand, the model trained with MAE loss function performs better overall with lower MSE and MAE values. Because we value the accuracy of the highest stress concentration factor the most, will we use Equation 4.5 as loss function when training the final model.

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda\sqrt{\mathcal{L}_{L2}(G)}. \tag{4.5}$$

Generator architecture

The model estimates accurate results with the smaller model architectures. Using four as the stride length decreased the training time and the size of model checkpoints substantially and is a good option for use in the final model. Removing the four layers in the middle of the architecture also predicted accurate

results and reduced the model by just over 100MB. Therefore will a combination of the two reductions be used in the final model. The stride length will still be four, and the two center layers will also be removed. Removing the two center layers with the double stride length will give us the same center layer size as the experiments with four removed center layers. The final model file size is 48MB, which is a 90% reduction from the original model size.

4.4 Training final models

With the knowledge gained from the experiments detailed in section 4.3 are we training two final models. One trained on the surface stress and one trained on the stress 1mm below the surface. The batch size used to train the final models are 1, and the model is trained for 300 epochs. However, the objective of this training is to find the best possible result, and we, therefore, observe the training carefully. If the model performs very well and then begins performing worse, we may end the training early.

Because of the smaller batch size is the training slower than when doing the experiments, and each epoch takes 55s.

4.4.1 Final model architecture

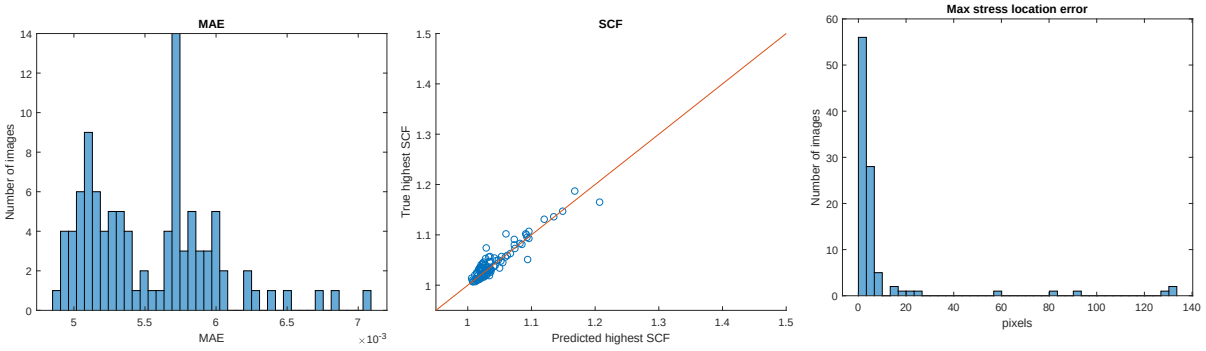
The final model architecture is listed in Table 4.7. The number of layers has been halved, and 4928 trainable filters are removed from the original model.

Table 4.7: Final model Generator Architecture

Layer	Input Size	Output Size	Filters	Layer type	Sends to	Receives from
1	(256,256,1)	(256,256,1)	-	Input	-	-
2	(256,256,1)	(64,64,64)	128	Convolutional	9	-
3	(64,64,64)	(16,16,128)	256	Convolutional	7	-
4	(16,16,128)	(4,4,256)	512	Convolutional	-	-
5	(4,4,256)	(4,4,512)	-	Concatenate	-	-
6	(4,4,512)	(16,16,128)	128	Transpose Convolutional	-	-
7	(16,16,128)	(16,16,256)	-	Concatenate	-	3
8	(16,16,256)	(64,64,64)	64	Transpose Convolutional	-	-
9	(64,64,64)	(64,64,128)	-	Concatenate	-	2
10	(64,64,128)	(256,256,1)	1	Transpose Convolutional	-	-

4.4.2 Plots of accuracy of after training the final models

The plots of accuracy for the 100 test images show that the final models are very accurate. To further test the models, will the validation data set be used.

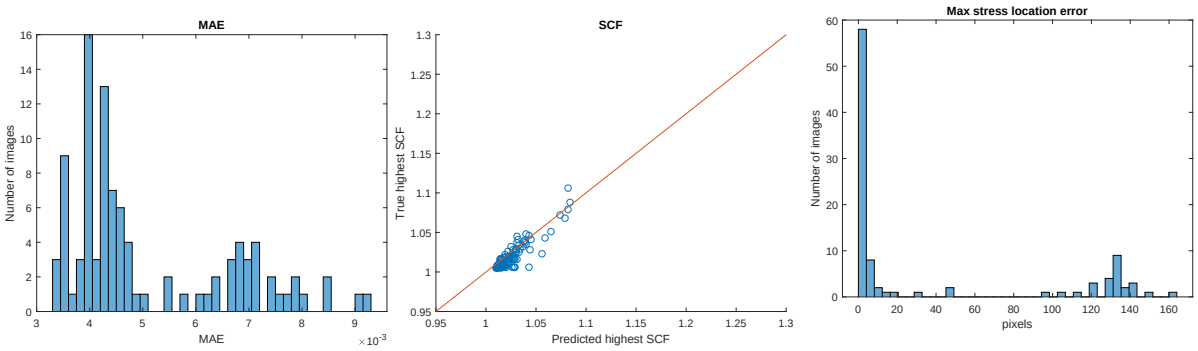


(a) MAE distribution

(b) Accuracy of peak stress value

(c) Distance between GT and predicted max stress

Figure 4.22: Accuracy of final surface model



(a) MAE distribution

(b) Accuracy of peak stress value

(c) Distance between GT and predicted max stress

Figure 4.23: Accuracy of final subsurface model

4.4.3 Using the trained models

After completing the training of the models, are we interested in being able to use them to predict images. To do this do we need to load the trained models and use them to generate images. Two methods of sharing the models are developed, one that may be run locally and one that may be run on Google colab.

Checkpoint files

During the training, checkpoints are stored. Checkpoints capture the exact value of all parameters used by a model. The checkpoints contain the weights biases and filter values from both generator and discriminator and are therefore large files. However, the checkpoints do not contain any description of the computation defined by the model and are therefore only useful when source code that will use the saved parameter values is available.

The checkpoints from the finished trained models are used when developing the python program Generate_images.py. This script loads the checkpoint files from the trained model and analyzes all images in a selected input folder.

To facilitate for analyzing single images instead of the type described in subsection 4.2.2 is the data set loader from the training program adjusted to import 256x256px images instead of 512x256px. To maintain the input pits number or name is the data set loader is adjusted to import the input images in alphabetical order. By collecting an alphabetical list of png files in the input folder, are we able to give

the predicted images names that correspond with the input image.

Algorithm 3: Algorithm for creating images with final model

input : Input Data path, Surface model path, surface-1mm model path, Result path
output: Surface stress images, Surface-1mm stress images

Initialize model architecture;
Filenames ← Image names in input folder;

Create a surface folder in result path;
Load surface model file ;
i ← 1 ;

for *All images in the input folder* **do**
    Generate image ;
    Generated image name ← ”Predicted image”+Filename(i);
    Save Generated image as Generated image name in surface folder;
    i ← i+1

Create surface-1mm folder in result path;
i ← 1;
for *All images in input folder* **do**
    Generate image ;
    Generated image name ← ”Predicted image”+Filename(i);
    Save Generated image as Generated image name in surface-1mm folder ;
    i ← i+1

Tensorflow model

The whole model may also be saved as a TensorFlow file. By using a single command, may the model with the associated weights and biases be loaded. The saved model requires much less space and does not require the original architecture to analyze the images. However, this method has proven difficult to run on systems without GPU when the model has been trained on GPU.

The trained models may be found on google drive [3]. The folder also contains input images and their corresponding stress images. A notebook designed to run the models are found in google colab [4].

[3] https://drive.google.com/drive/folders/1V1A87veYLSmBJSJL16pRpnfnRR1W_0Ot?usp=sharing

[4] https://colab.research.google.com/drive/1qywXADDKr7MIdkWFA6GK4jNJWcFo9Csp?usp=sharing

Chapter 5

Results and discussion

5.1 Results

The result of this thesis is the finished models and the scripts described in subsection 4.4.3. To analyze the accuracy of the models was the Generate_images.py program used to analyze the validation data set. The result from the validation data set. The program is run locally without a GPU, which produces ten images per second. Considering that one mooring chain link often have 2000 pits will it take 200s to complete an analysis of one mooring link.

Generate_images.py and the validation data set may be found with the attached files.

5.1.1 Example images

To get an understanding of how the generated images look are some images scaled. The ground truth images are scaled such that the highest value is completely white and their lowest value is black. The corresponding predicted image is scaled with the same parameters as the ground truth. The images illustrated in Figure 5.1 are chosen at random and do not represent any particularly good or poor result.

5.1.2 Accuracy and loss for the validation data

To understand how the complete validation data set performs, may we look at both models' mean error and variance. This may be found at Table 5.1.

Table 5.1: Mean error and Variance

-	Surface	Surface-1mm
Mean error	0.0080	0.0056
Variance	$5.7572 \cdot 10^{-5}$	$4.6614 \cdot 10^{-5}$

To get a picture of the accuracy of the model may we look at the MAE and the predicted peak stresses of the validation data.

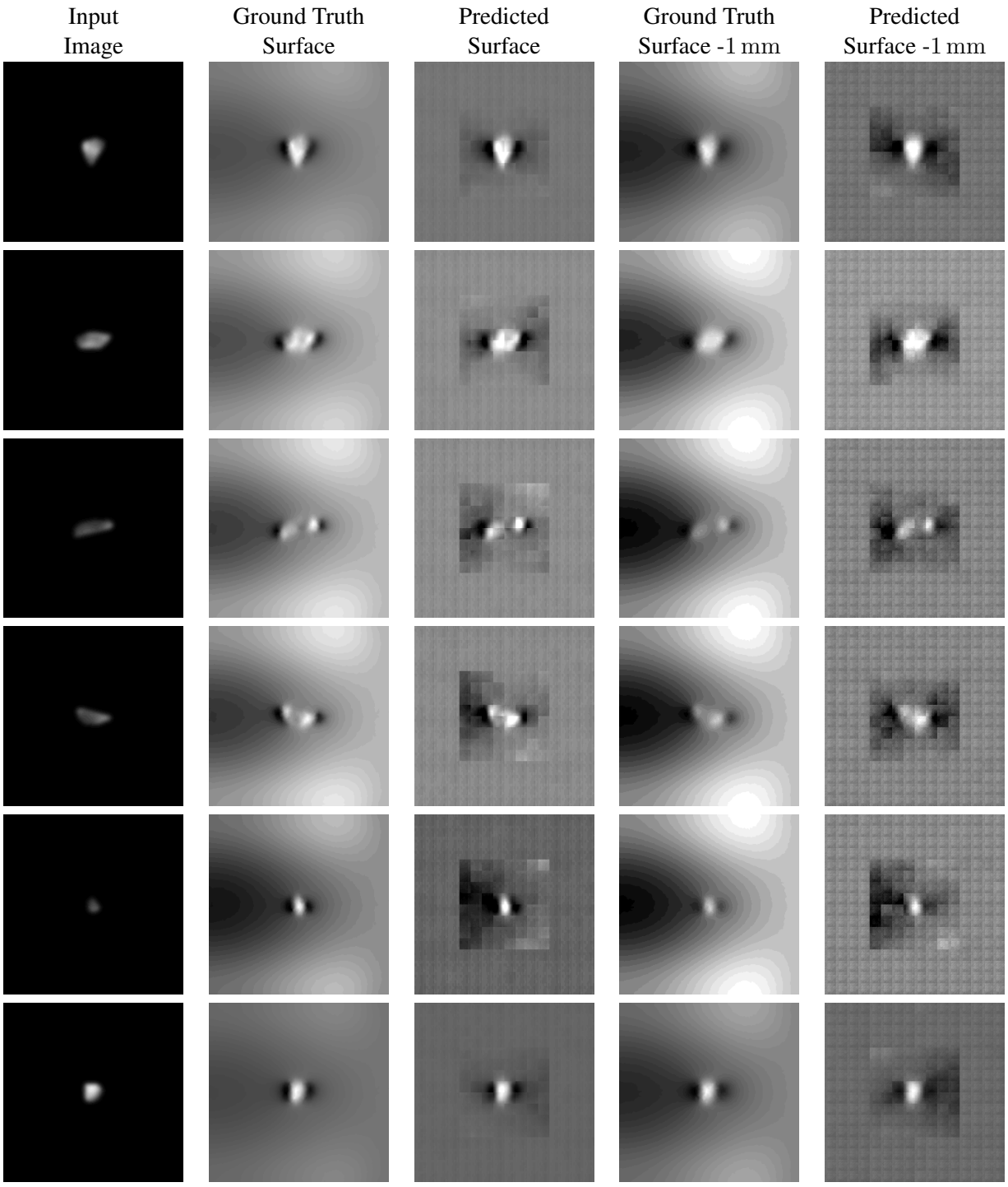
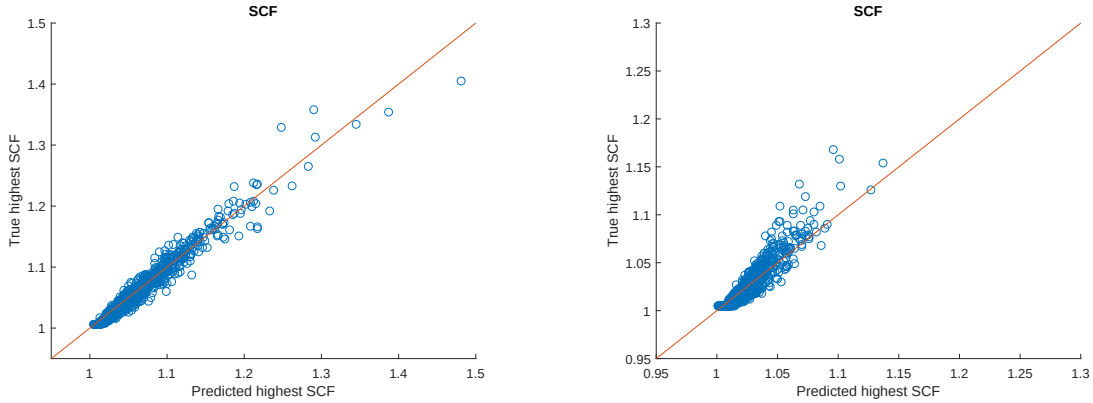
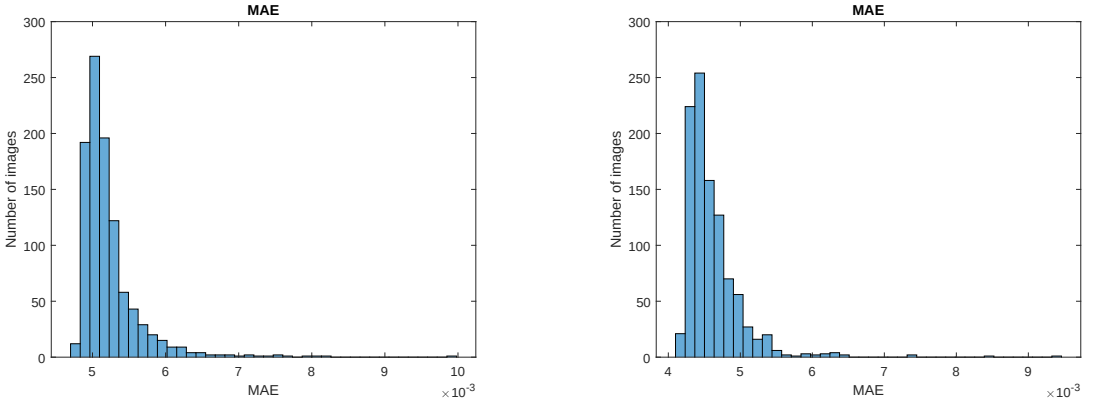


Figure 5.1: Example results from the models



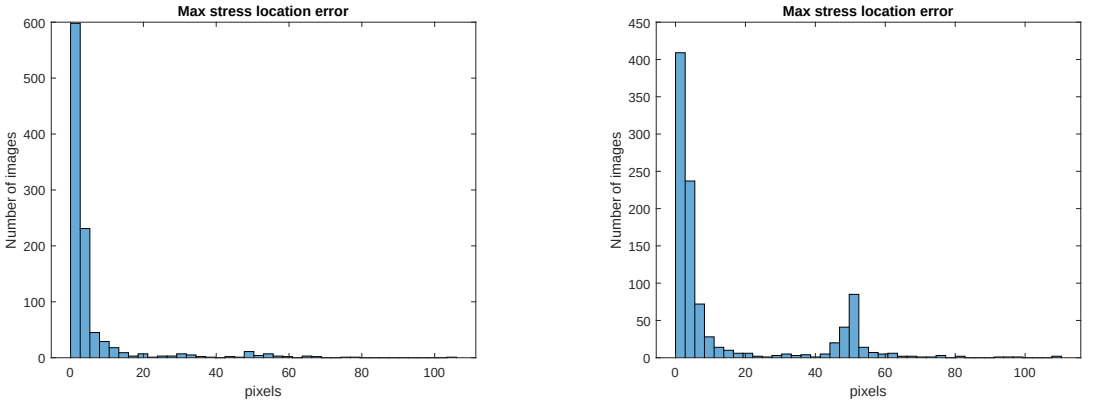
(a) Surface **(b)** Subsurface

Figure 5.2: Difference between the value of actual and predicted peak stress



(a) Surface **(b)** Subsurface

Figure 5.3: MAE of the validation data set



(a) Surface **(b)** Subsurface

Figure 5.4: Distance between predicted and actual highest stress

5.2 Discussion of results

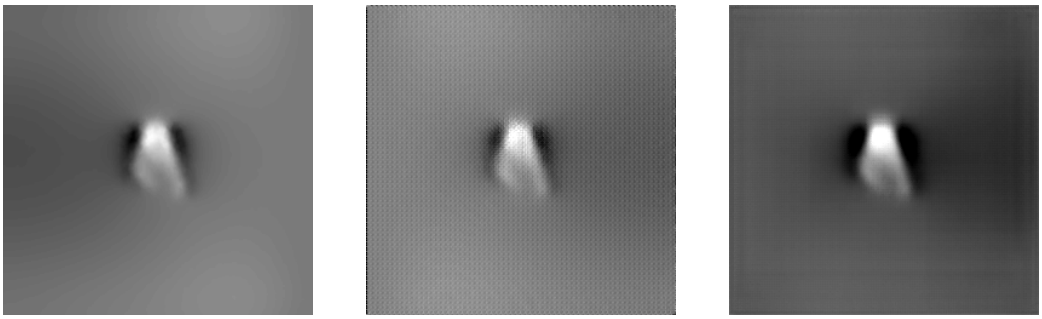
The result presented in this thesis suggests that machine learning and general adversarial networks may be used to emulate FEM results. However, the results are also quite limited. Both models presented in the results are trained and tested on pits with max depths smaller than $12\,\mathrm{mm}$ and peak stresses with SCF smaller than 1.5. The inaccuracy observed has been rather small, and utilizing a safety factor when calculating will make the error negligible. Still, we want to discuss the results and their inaccuracies.

5.2.1 Causes of errors

To investigate what causes the models to be less accurate are histograms made where losses are put into context with pit dimensions and predicted stress. The plots may be found in Appendix B. The plots reveal that the model is not as accurate when estimating the SCF value for deeper pits and the peak stress location for small pits. This is especially clear with the model trained on stresses $1\,\mathrm{mm}$ below the surface. In this model, we may observe very accurate results in pits with depth shorter than $5\,\mathrm{mm}$ and accurate but not as accurate results pits between $5\,\mathrm{mm}$ and $12\,\mathrm{mm}$. The loss in accuracy when finding SCF in deeper pits is most likely because the data set used has been dominated by small pits. The distance error in more shallow pits is most likely because the stresses are so small that a pixel outside of the pit just happens to have a larger value. The two errors are very well summed up in the plots where predicted SCF is plotted against the errors. We see high SCF give poorer results for peak stress value and low SCF give poor results for peak stress location.

5.2.2 Comparing result to other models

The focus of this thesis has been to build a network that is effective and still accurate. The result of this has been the pixelated images we may observe in Figure 5.1. If we compare the images with Figure 5.5 is it clear that both the full-size model and the model with removed center layers produce smoother images. However, when judging the model by MAE and value and location of SCF is the better models not as obvious. To better understand how the reduction of the model affected the results was a full-size model trained with a batch size of 1 and 300 epochs. The model then used the validation data set to generate images. Accuracy plots from the model may be found in Appendix A. A comparison of the plots shows that the accuracy of the reduced model is not worse than the full-size model, even though the images are not as clear.



(a) Ground Truth **(b)** Prediction Full size model **(c)** Prediction model w.o. 4 layers

Figure 5.5: Output of other models

Discriminator loss

The images not looking authentic may be a result of the discriminator not producing large enough losses. This may result from λ not being fine tuned when using RMSE or that the discriminator cannot distinguish between real or fake images.

5.2.3 Generality and accuracy

As discussed in subsection 4.2.10 may the normalization of the model not be the best fit when training on the data sets presented in this thesis. Another source of inaccuracy may be the pixel values used in the images. The pixel values may have been scaled such that the deepest pit had the highest possible value in the input image, and the highest possible value represented the highest SCF in the target image. This could have helped with the accuracy of the model. However, it would have limited the generality of the model.

In this thesis has a general model valued higher than the absolute accuracy of the model. This may have lead to more inaccurate results. This has been chosen as the model architecture may be used in later research. As the model is sufficiently accurate to prove the concept has the choice of a general model proved to be well.

5.2.4 Distance under surface

The value of a_0 was chose to be $1\,\mathrm{mm}$. The results predicting stress $1\,\mathrm{mm}$ below the surface have shown to be not as accurate as the stress in the surface. This may come from the fact that the stresses $1\,\mathrm{mm}$ below the notch is not as sensitive to the notch geometry, and a more appropriate value for this may be $0.5\,\mathrm{mm}$.

5.2.5 Predicting complete stress field

The necessity of predicting the complete stress field around the pit should be discussed. The models presented in this thesis have been able to predict stress fields around a pit very accurately. However, the model is large and time-consuming to train. Furthermore, we only need the highest stress to calculate the fatigue limit of a component as described in subsection 3.1.4. Predicting the complete stress field may be an overly complicated solution to the problem. Simple convolutional networks are shown to be able to predict peak stresses in ellipsoid notches and $0.5\,\mathrm{mm}$ under the notch.

On the other hand, may the stress field around the pit be very useful if pits are located close to each other. If we may use a superposition approach to the stress-fields, as Paul JR and Faucett (1962) presented, could the stress fields be placed on top of each other, and total peak stress may be found.

Chapter 6

Conclusion and further work

6.1 Conclusion

Two generative adversarial networks have been trained on data sets developed using finite element analysis. The data set contain images of pit topology and stress both in the surface and $1\,\mathrm{mm}$ below the surface. The results presented in section 5.1 suggest that using generative adversarial networks is a very promising approach for approximating finite element method results. The models presented in this thesis are in pits in plane stress with depths larger than $1\,\mathrm{mm}$ and smaller than $12\,\mathrm{mm}$ able to accurately predict the value and location of the peak stresses.

Numerous models were trained to experiment with the models loss function and architecture. When training the last model, the loss function is RMSE because the nominal stress dominates the stress images, and RMSE weighs outliers heavily.

The model size was reduced by increasing the stride used in the convolutional layers in the generator and removing two layers from the center of the model. The reduction of the model lead to the images looking less authentic while still maintaining good accuracy. The reduction of model size was successful. The smaller model needs less time to train and less storage space while producing accurate results as the full-size model. The model reduction shows that a smaller model is able to perform just as well as a larger one, yet it is still uncertain if the smaller model is a better choice than the original size when analyzing more diverse data sets.

The combination of changing model architecture and loss function may have led to the generator error not being weighed heavily enough. The generator has produced images that do not look like target images. Still, it is concluded that the images are still sufficiently accurate.

This thesis aims to design a model that may be trained on more diverse data sets. Consequently, the pixel values used in the input and target chosen to be as general as possible. The image parameters are found in Table 6.1. These pixel values give the user a precision within $10^{-3}\,\mathrm{mm}$ in the input image and 10^{-3} for SCF in the output image. The UINT16 datatype limits the input to a max depth of $65.536\,\mathrm{mm}$ and output to max SCF of 65.536, which is acceptable.

Table 6.1: Pixel parameters

Parameter	Value
Pixel width	$0.234\,\mathrm{mm}$
Pixel height	$0.234\,\mathrm{mm}$
Pixel value Input	Depth in $\mu\mathrm{m}$
Pixel value Target	SCF $\times 10^{-3}$

The training script is available online in Google colab[1] and may be used to solve similar problems. The two trained models developed in this thesis are available in the attached program generate_images.py and online in Google colab[2].

[1] https://colab.research.google.com/drive/1Wl462lHNyoYAp2tliSd-9m1JoQ0_SliU?usp=sharing

[2] https://colab.research.google.com/drive/1qywXADDKr7MIdkWFA6GK4jNJWcFo9Csp?usp=sharing

6.2 Further work

This section contains the author's recommendation for further investigation of the application of machine learning when analyzing corrosion pits.

Before initiating any further work, is it necessary to decide if the whole stress field is needed when analyzing single pits or if the peak stress is sufficient. If it is desired that the whole stress field around the pit is desirable, should the application of the superposition principle of pit stress fields be investigated. If a superposition principle may be used, then the result for a whole mooring chain link could be produced quickly.

As the data sets used when training the models have been small, a more diverse data set should be developed. The data set should contain deeper pits that may give larger peak stresses. This may help the model be more robust and diverse such that model that can handle deeper pits and more significant stresses.

With a more diverse data set may the generator proposed in this thesis be inadequate. Therefore, further work should investigate the model architecture and if the small model is large enough to solve more diverse problems.

Bibliography

Adalogou, N., 2020. Intuitive explanation of skip connections in deep learning.

ANSYS, I., . Solid185 element description.

Bergara, A., Arredondo, A., Altuzarra, J., Martinez-Esnaola, J., 2020. Calculation of stress intensity factors in offshore mooring chains. Ocean Engineering 214, 391–402.

Berge, S., Ås, S.K., 2017. Fatigue and Fracture Design of Marine Structures. volume 3.

Bushaev, V., 2018. Understanding rms-prop faster neural network learning.

Developers, G., . Overview of gan structure.

Fontaine, E., Kilner, A., Carra, C., Washinton, D., Ma, K., A, P., 2014. Industry survey of past failures, pre-emptive replacements and reported degradation for mooring systems of floating production units. Offshore technology conference OTC , 14.

G. Yalçın, O., 2020. Mastering tensorflow tensors in 5 easy steps.

Gemiland, G., Reed, P., Sobey, A., 2021. Selection of appropriate numerical models for modelling the stresses in mooring chains. Marine Structures 75.

Griffith, A., 1921. The phenomena of rupture and flow in solids. Philosophical Transaction of the Royal Society of London 221, 163–198.

Hahnloser, R.H.R., Sarpeshkar, R., Mahowald, M.A., Douglas, R.J., Sebstian, H., 2000. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature 406, 947–951.

Irwin, G., 1957. Analysis of stresses and strains near the end of a crack traversing a plate. J. Appl. Mech. 24, 361–364.

Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A., 2017. Image-to-image translation with conditional adversarial networks. CVPR .

Katba, C., . Code4pa: Deep learning neural networks to address the opioid epidemic.

Kim, Y., Kim, M.S., Park, M.J., 2019. Fatigue analysis on the mooring chain of a spread moored fpso considering the opb and ipb. International Journal of Naval Architecture and Oncean Engineering .

Ma, K., Duggal, A., P, S., L'Hostis, D., Shu, H., 2013. A historical review on integrity issues of permanent mooring systems. Offshore technology conference OTC , 14.

Mahanta, J., 2017. Introduction to neural networks, advantages and applications.

learning mastery, M., 2020. MS Windows NT a gentle introduction to the rectified linear activation function.

McGonagle, J., Shalkouski, G., Williams, Christophe rand Hsu, A., Khim, J., Miller, A., 2020. Backpropogation.

M.K., C., R.G., B., N., N.B., 2010. Modeling the environmental dependence of pit growth using neural network approaches. Corrosion Science 52, 3070–3077.

Nikhil, T., 2020. Why use concolutional layers.

Ok, D., Pu, Y., Incecik, A., 2007. Artificial neural networks and their application to assessment of ultimate strength of plates with pitting corrosion. Ocean Engineering 34, 2222–2230.

Parmar, R., 2018. Common loss functions in machine learning.

Paul JR, F.W.P., Faucett, T.R., 1962. The superposition of stress concentration factors. Journal of engineering for industry 5, 129–132.

Singh, R., 2020. Wavelet scattering network -just to begin.

Wikipedia, 2020. Artificial neural networks.

Appendix A

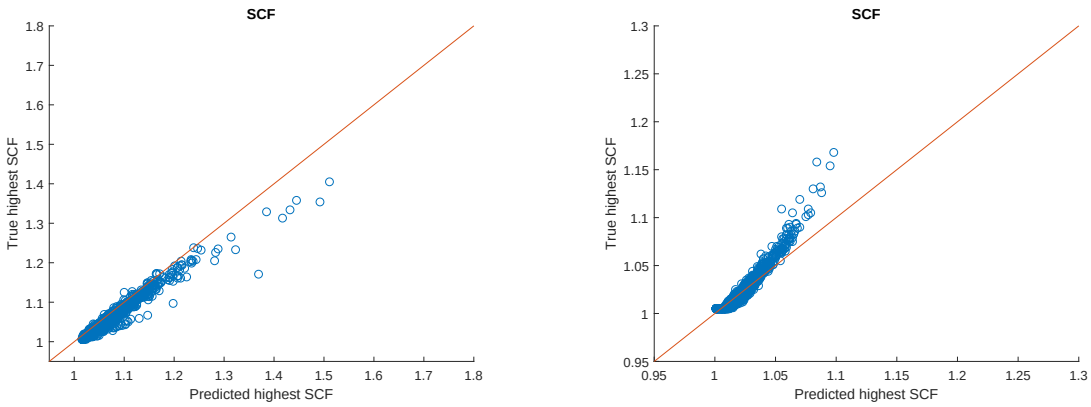
Results full size model

A.1 Results

Table A.1: Mean error and Variance

-	Surface	Surface-1mm
Mean error	0.0158	0.0049
Variance	$1.856 \cdot 10^{-4}$	$4.3954 \cdot 10^{-5}$

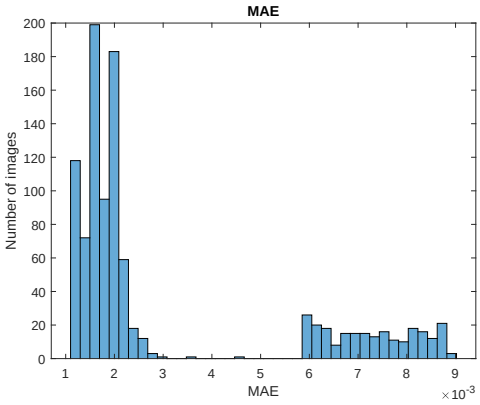
A.1.1 Loss plots for full size model



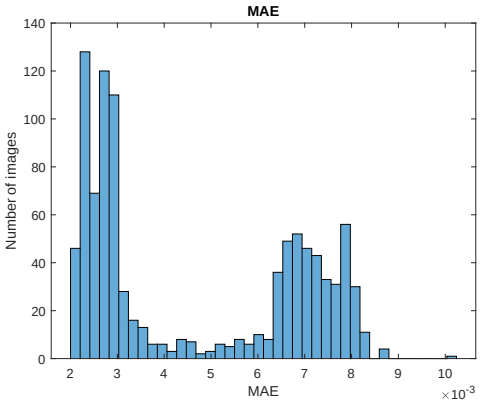
(a) Surface

(b) Subsurface

Figure A.1: True peak stress plotted against predicted full size model

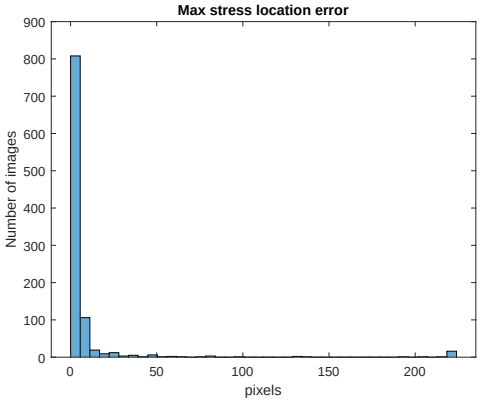


(a) Surface

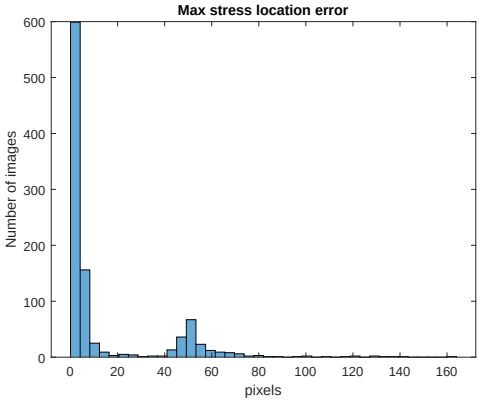


(b) Subsurface

Figure A.2: Mean average error in full size model



(a) Surface



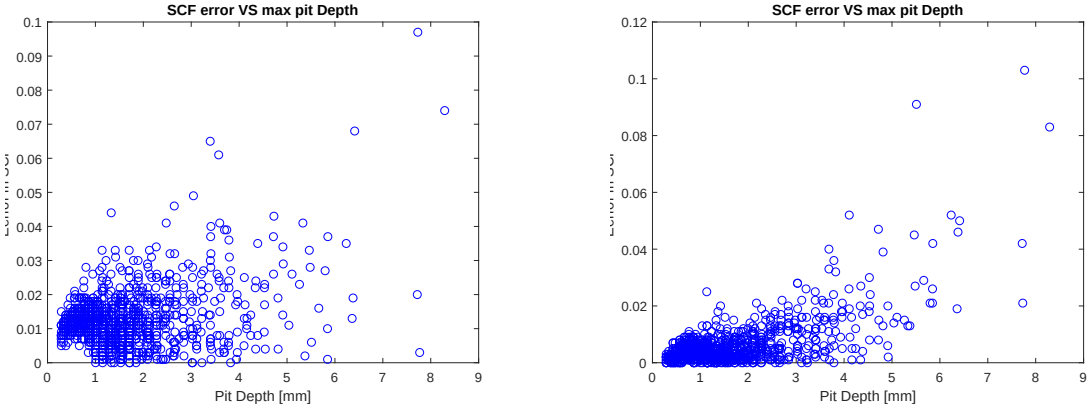
(b) Surface

Figure A.3: Distance error in full size model

Appendix B

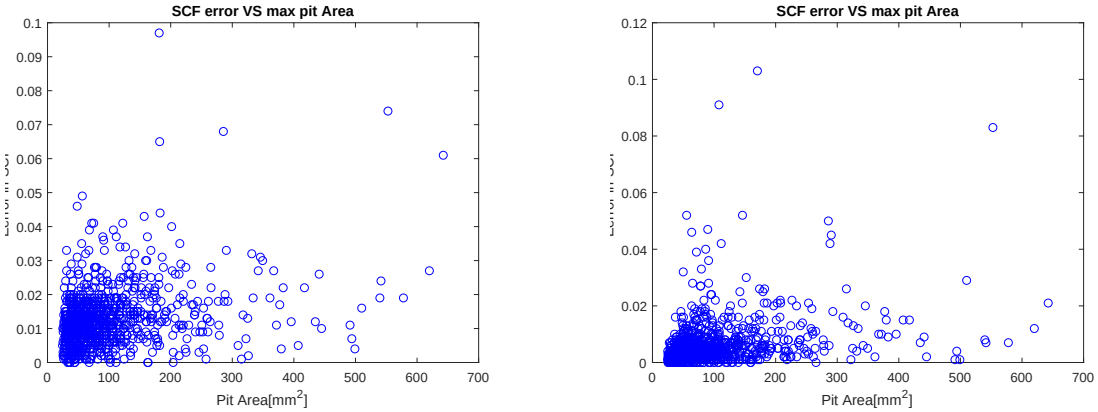
Error patterns final model

B.1 SCF value error



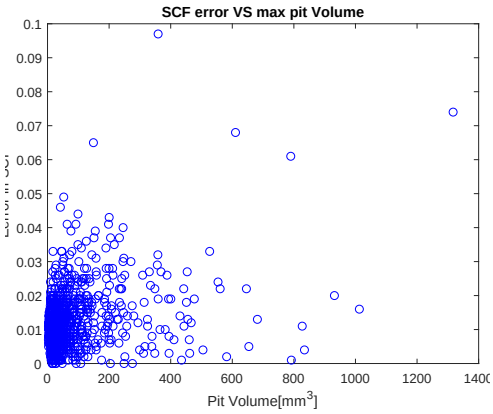
(a) Surface **(b)** Subsurface

Figure B.1: SCF error VS max pit depth

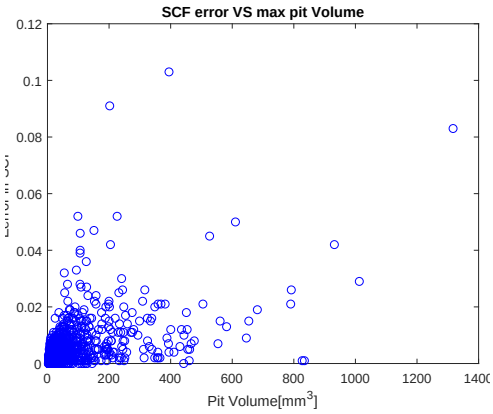


(a) Surface **(b)** Subsurface

Figure B.2: SCF error VS pit area

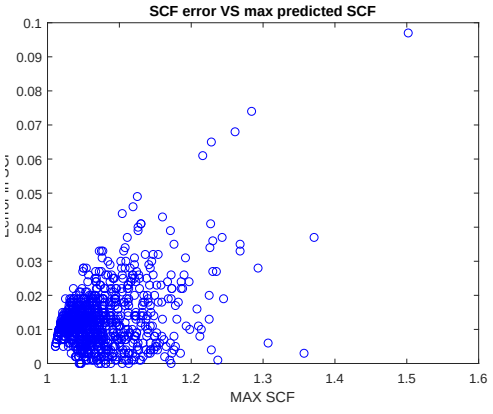


(a) Surface

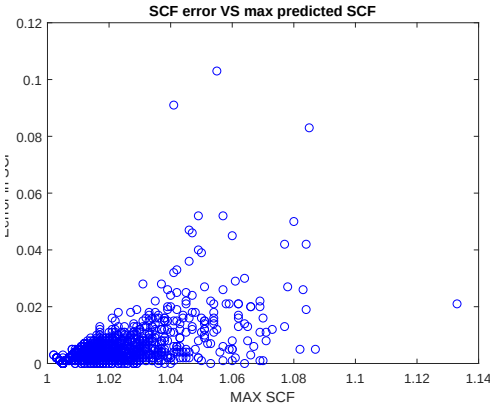


(b) Subsurface

Figure B.3: SCF error vs pit volume



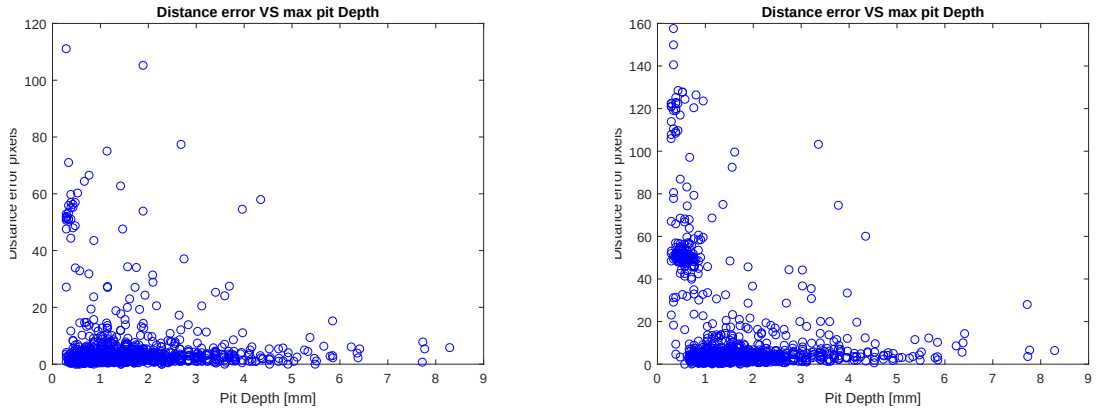
(a) Surface



(b) Subsurface

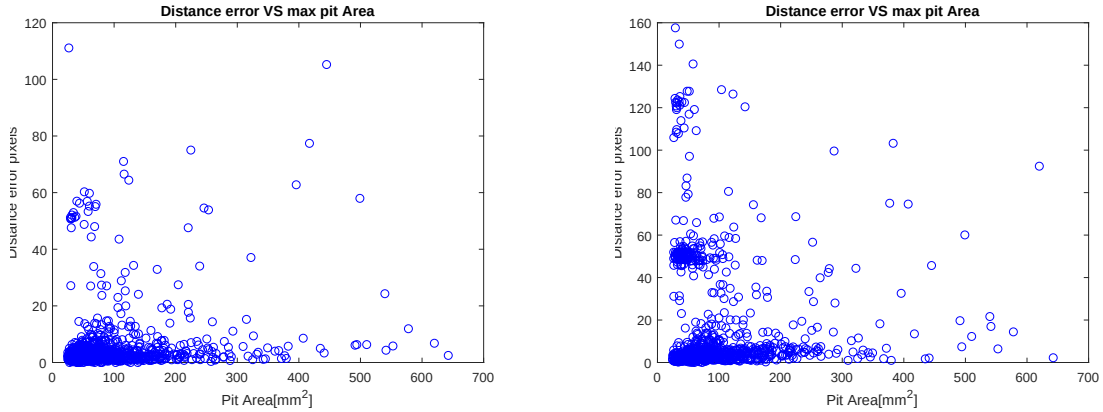
Figure B.4: SCF error VS predicted SCF

B.2 SCF Distance error



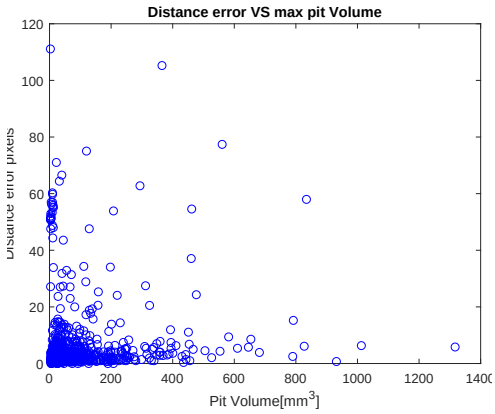
(a) Surface **(b)** Subsurface

Figure B.5: Distance error VS max pit depths

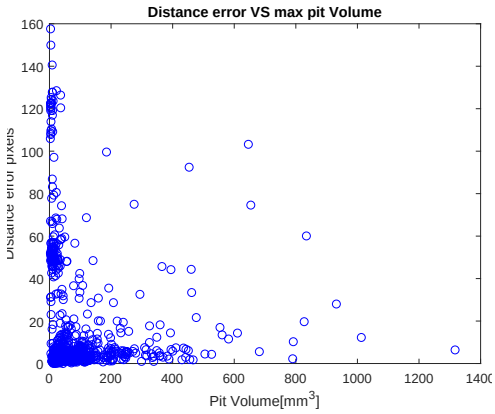


(a) Surface **(b)** Subsurface

Figure B.6: Distance error VS pit area

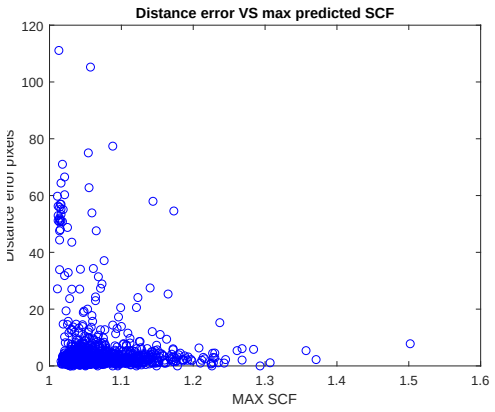


(a) Surface

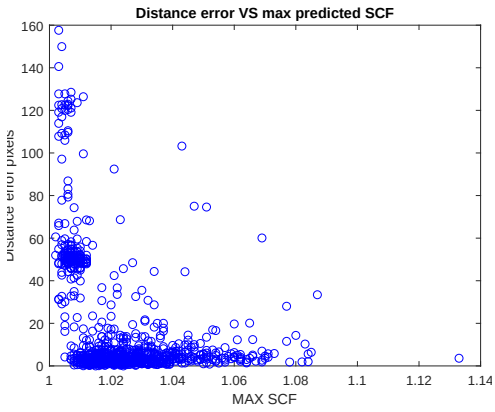


(b) Subsurface

Figure B.7: Distance error VS pit volume



(a) Surface



(b) Subsurface

Figure B.8: Distance error VS predicted max SCF

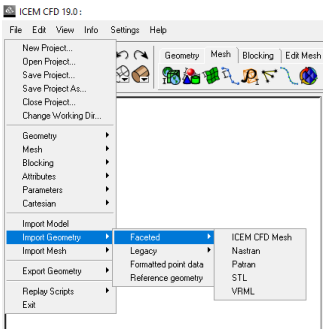
Appendix C

ICEM CFD

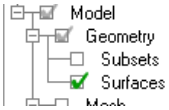
Meshing STL file in ICEM-CFD

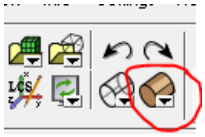
Importing

- Click “Change working dir” and select desired directory
- Click “Import Geometry”-“Faceted”-“STL” and select desired file



- Check “Surfaces” box and solid simple display to visualize the model

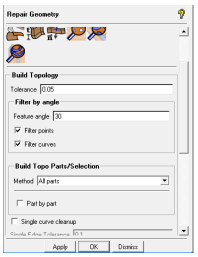




Repairing geometry

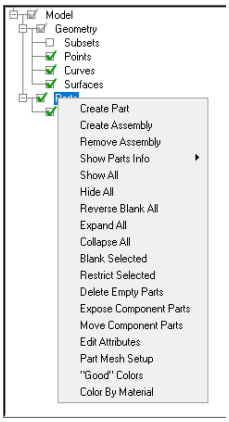
- Select the repair geometry tool

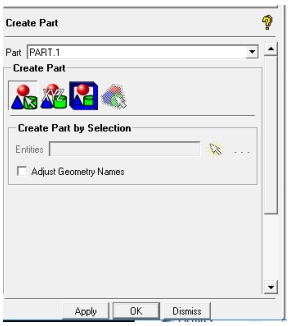
- Select tolerance and feature angle to filter by.
- Check both “filter points” and” filter curves” to mitigate the risk of having unnecessary lines and points in your model.
- Click “OK”



Creating Parts

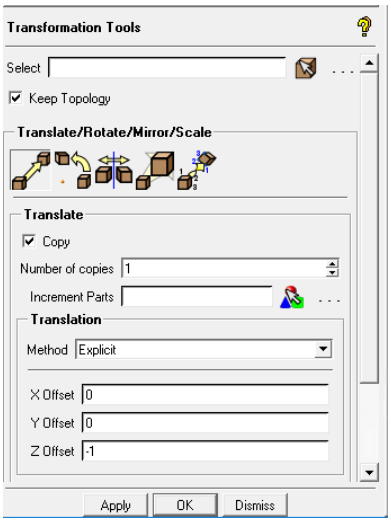
- Right click on “Parts”in the model list
- Select “Create Part”
- Click on “Create part by selection”
- Click on the desired surface and confirm by clicking middle button
- Write part name
- Click OK





Translating part

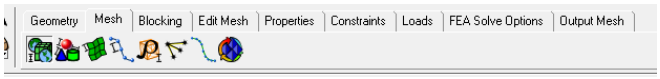
- Select the transform geometry tool

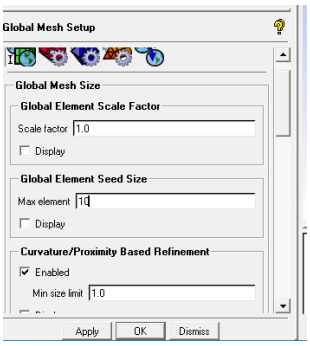


- Select part to transform with the “select geometry” button
- Select Translate geometry (key=m)
- Check “copy” box to keep the original part
- Select the desired offset
- Click Apply

Mesh

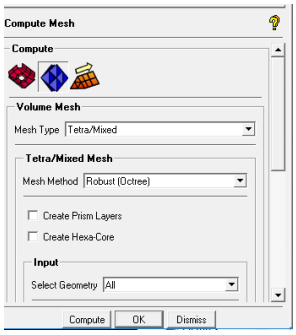
- Select “Global mesh setup”





- Check “curvature/proximity based refinement” box
- Select max element size and min Element size
- Click ok

- Select Compute mesh

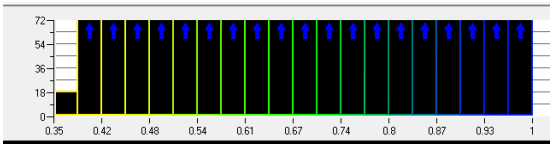


- Select mesh type and mesh method
- Click “Compute”

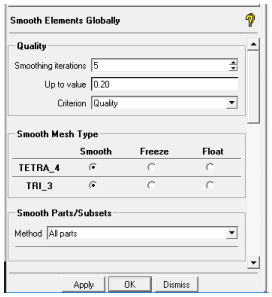
Smooth Mesh

- If the mesh is not looking good smooth the mesh with “Smooth mesh globally tool”

- The element quality is shown in the lower right corner

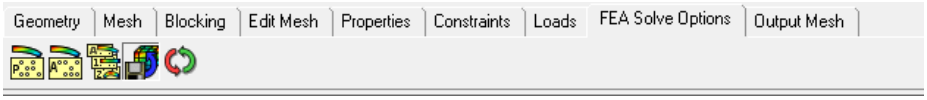


- Select the smoothing parameters
- Click OK
- Check mesh quality

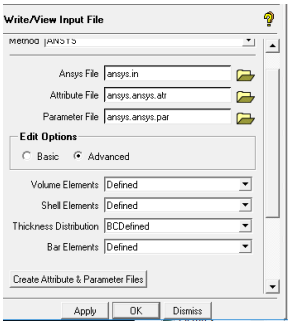


Export model

- Select “Write/view input file”



- Select advanced options
- Change volume elements from defined to “all”
- Click “Create Atribute and parameter file”
- Click OK.



NTNU

Kunnskap for en bedre verden