Master's thesis

2020

Simon Andreas Hoff

Master's thesis

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Simon Andreas Hoff

Enhanced analysis of ultrasonic impedance logs: Improved imaging and fluid channel detection

July 2020

NTNU
Norwegian University of
Science and Technology

Enhanced analysis of ultrasonic impedance logs: Improved imaging and fluid channel detection

Simon Andreas Hoff

Cybernetics and Robotics

Submission date:	July 2020
Supervisor:	Lasse Løvstakken
Co-supervisor:	Erlend Magnus Viggen

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Abstract

The aim of this thesis is to detect vertical features in the the casing cement of oil and gas wells by using pattern recognition techniques based on machine learning on ultrasonic log data. Detecting channels is an important part of well integrity evaluations, which is the process of evaluating whether the casing cement provides a hydraulic seal of the annulus of the well. Automatic detection of channels in the casing cement can be used to make well integrity evaluations more efficient and robust, which is important for rig safety, as well as for plug and abandonment, and CO_2 injection. While automatic feature detection in well logs is common, most such detection has been focused on picking azimuthal features in the well logs. Therefore, existing methods are not suitable for detecting channels in the casing cement, as such features are mainly vertical. In this work, well log images are interpolated using corrected measurement locations and state of the art statistical interpolation techniques in order to aid the annotation process. It is shown that this process explains artefacts visible in the raw images normally displayed for well integrity evaluations. To detect channels, image segmentation is performed using deep learning. While several improvements are made compared to the similar approach used in a previous project this work is based on, the results indicate that deep learning may not be the best alternative for such detection.

Sammendrag

Målet med denne oppgaven er å detektere vertikale trekk i sementen rundt foringsrøret i olje- og gassbrønner ved å bruke mønstergjenkjenningsteknikker basert på maskinlæring på ultralydloggdata. Å oppdage kanaler er en viktig del i vurdering av brønnintegritet, som er prosessen med å evaluere om foringsrørets sement gir en hydraulisk tetning av brønnens ringrom. Automatisk deteksjon av kanaler i foringsrøret kan sement brukes til å gjøre evalueringer av brønnintegritet mer effektive og robuste. Dette er viktig både for riggsikkerhet, men også for plugging og forlating, og brønner som vurderes for CO_2-injeksjon. Selv om automatisk deteksjon av trekk i brønnlogger er vanlig, har de fleste løsninger for dette vært fokusert på å detektere trekk i horisontal retning i brønnloggene. Derfor er eksisterende metoder ikke egnet for å oppdage kanaler i foringsrørets sement, ettersom slike trekk hovedsakelig er vertikale. I denne oppgaven interpoleres brønnloggbilder ved bruk av korrigerte målelokasjoner og statistiske interpolasjonsteknikker for å hjelpe evalueringsprosessen. Det vises at denne prosessen forklarer artefakter som er synlige i bildene som normalt vises for evaluering av brønnintegritet. For å oppdage kanaler, benyttes bildesegmentering ved bruk av dyp læring. Mens flere forbedringer er gjort sammenlignet med den lignende tilnærmingen brukt i et tidligere prosjekt dette arbeidet er basert på, indikerer resultatene at dyp læring kanskje ikke er det beste alternativet for deteksjon av kanaler i brønnloggdata.

Preface

One of the initial challenges of this process was resampling the images to the same physical resolution. Previously, we used nearest neighbor interpolation to achieve this. However, we were concerned that this would cause issues for the convolutional kernels when learning features.

This led to exploring corrections for where the measurements are made, and how to reconstruct an image based on that additional information. Using kriging turned out to provide great results, and explain artefacts seen in the raw images.

This means that this thesis has ended up running down two different paths: First, I explore in detail how to localize where each measurement is collected from, and use this to reconstruct upsampled images. Second, I annotate raw images, using what I learned from the image upsampling, and use these data as input for a deep learning model.

Because this thesis combines the field of data science and machine learning with petroleum technology, I have tried to make this thesis accessible for people of either background. This means that a lot of basic material well known to people working in the corresponding fields will be presented.

I would like to thank all the people who have helped me in the work with this thesis. First, I would like to thank my advisor Erlend Magnus Viggen for all the fruitful discussions, for the advice on dealing with confusing results, and for all the help in proof reading the thesis. I would like to thank Ioan Alexandru Merciu for all the insights on how to interpret the well logs, as well as sharing industry insights. Finally, I would like to thank my wonderful wife Kaja for all her love and support.

Contents

Chapter 1

Introduction

To a large extent, operations in the petroleum domain include drilling. An important part of the drilling process is placing and cementing the casing. The cement is important for securing the casing in place, and also for providing a hydraulically isolating layer. In order to verify that the cement has been placed properly, there are two options. One may perform hydraulic pressure testing of the well, or one could use well logging techniques to measure the presence of cement. Evaluating the quality of the cement job from such logs is a critical task, as it is important both for the safety of the drilling operation, but also for plug and abandonment (P&A), as well as wells considered for CO_2 injection.

Providing automated analysis tools to assist evaluators in assessing the quality of the cement job would help make evaluations better, and more consistent. Existing feature detection tools for well logs are only able to A common problem in cement jobs is that channels of mud are left through the cement, as shown by [1]. Further, over time it is common for the cement to degrade, leaving cracks and microannuli through the cement as shown by [2]. These features may be visible in well log images, but it can still be time-consuming to assess whether each such feature will allow vertical fluid flow or not, and as such whether it will influence the integrity of the well. The aim of this thesis is to improve the deep learning-based model developed in [3], to assist well integrity evaluations.

1.1 Previous work

In the past, much emphasis has been put on detecting sinusoidal shapes in the azimuthal direction in log images, as these are the results of planar features intersecting the well, such as formation fractures, or transitions between different geological layers. A common approach to this is using the Hough transform, as shown by [4], [5], and [6]. The shortcoming of such methods is however that they rely on matching pre-defined patterns to the images. This works well in the azimuthal direction, as it has a limited extent. However, features in the vertical direction can be arbitrarily long, and as such, one can no longer rely on pre-defined patterns. In recent years, advances in machine learning and particularly deep learning has enabled computer vision techniques well suited to handling such features, which will be the topic of this thesis.

This thesis is based on an earlier project at NTNU [3], meaning that some of the work presented here was performed for that project. Consequently some of the theory and methods presented will be similar or identical to those presented in [3].

This thesis has to a large extent aimed to solve some of the problems encountered in the previous project. First, a large emphasis has been put on increasing the size of the labelled data set, as well as increasing the quality of the labels. Further, semi-supervised learning has been employed to leverage unlabelled data to improve model performance. Additionally, an approach to making the model periodic in the azimuthal direction has been implemented in order to solve the problem of poor detection around the edges of the image seen in [3]. Finally, in order to support the annotation process, the measurement setup has been analyzed. This has included analysis of the locations of the measurements collected, as well as a study in how to interpolate the measurements, focusing on Gaussian processes, also known as kriging.

1.2 Acknowledgements

The author would like to thank Equinor ASA for providing the data used for this thesis, and CIUS for providing computational resources.

1.3 Thesis structure

Chapter 1 Introduction: This chapter

Chapter 2 Drilling, cementing, and logging: An introduction to drilling operations and logging. Furthermore, well integrity evaluations are introduced, and the topic of channel classification is discussed.

Chapter 3 Machine learning: A simple introduction to the foundations of machine learning, with emphasis on how computers learn from data, and how to structure the data for training and testing.

Chapter 4 Deep learning: Background on artificial neural networks, deep learning, and image classification.

Chapter 5 Log data and imaging: Methods for improving the display of well log images.

Chapter 6 Deep learning methods for fluid channel detection: Methods for training and testing the deep learning based model for use in well integrity evaluations.

Chapter 7 Channel detection results: Results from the deep learning part of the project.

Chapter 8 Discussion: Discussion of the results obtained, and what remains to be done.

Chapter 9 Conclusion: A summary of the findings in this thesis.

Chapter 2

Drilling, cementing, and logging

This chapter will provide a brief introduction to how oil wells are constructed, from drilling the wellbore, through the construction and validation phases, to the operational stage.

2.1 Well construction

An important part of petroleum rig operation is drilling. The process of drilling is illustrated in Figure 2.1, and in the following a short description is provided. First, a hole is drilled using a drillstring. In order to remove formation cuttings from the well, a drilling fluid, often referred to as mud, is sprayed from the drillbit, creating an upward flow in the wellbore which brings cuttings to the surface. After drilling the section is complete, the drillstring is extracted from the well. At this point, a steel pipe referred to as a casing or liner is inserted into the well. A casing extends from the top of the well, whereas a liner will be fixed to the bottom of the existing casing in the well. This casing serves two purposes: First, it should isolate any fluid in the wellbore from fluids in the formation. Second, it serves to stabilize the hole, preventing the surrounding formation from collapsing inward. Finally, a cement is injected in the annulus between the casing and formation to stabilize the casing, and to provide a hydraulically isolating layer to prevent any fluid flow from occuring on the outside of the casing.

In order to verify that the cement provides hydraulic isolation, testing is necessary. Common ways of testing is to do a hydraulic pressure test, in which one pressurizes the wellbore with a constant pressure, checking to see if the pressure drops, which would indicate a leaking flow on the outside of the casing. Such testing could damage the cement [7], meaning that non-destructive testing is often preferred. One such approach is to use logging tools to evaluate the quality of the cement job.

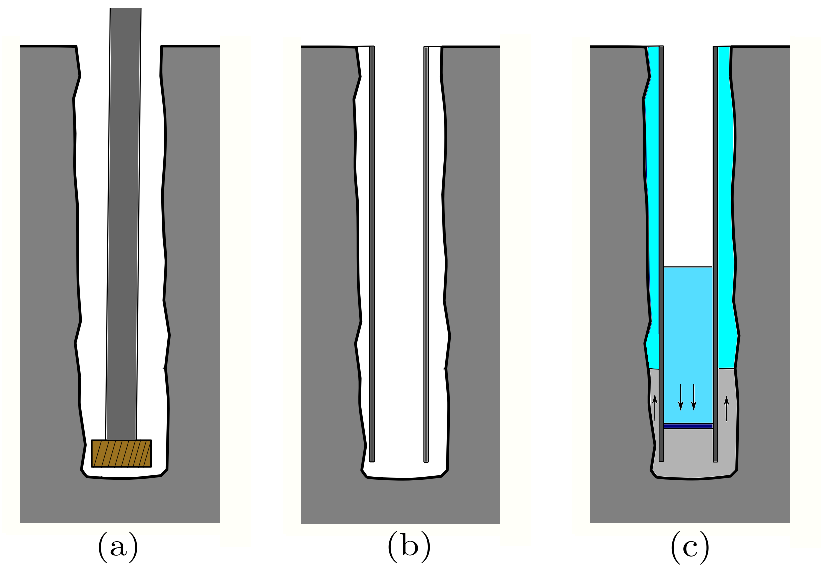

Figure 2.1: Overview of how drilling is performed. (a) The well is drilled. After reaching the desired depth, the drillpipe is pulled out of the hole. (b) A casing is inserted to stabilize the hole. (c) Cement is pumped down inside the casing to the bottom, pushing it up into the annulus [8]. Before and after the cement are spacer fluids, designed to prevent the cement from being contaminated by drilling mud.

2.2 Well logging

The first wireline well log was recorded by Schlumberger 1927, and processed using methods developed by Sabba Ștefănescu [9]. In that case the formation resistivity was recorded at a number of depths to test for the presence of hydrocarbon. In the following decades great advances have been made in measuring the borehole conditions using wireline logging techniques.

Wireline logging is performed by lowering a measurement apparatus into the wellbore. This is held up by a wire which also serves as a signal carrier for collecting the logging data. Typically the toolstring is built from a number of individual tools that are lowered into the hole together. In some cases, a well is logged multiple times. This can be for a number of reasons: There may be a need to run multiple different toolstrings in order to collect all the desired measurements. Another reason may be that an interval of the log is showing signs of poor data quality, which means that one will run the toolstring across the affected interval again, in what is referred to as a "repeat pass". For cased hole logging, it is typically routine to do a repeat pass to verify that the measurements are repeatable.

2.3 Acoustic logging

In order to evaluate the quality of a cement job, acoustic tools are essential. Traditionally, the cement bond log (CBL) [10, 11, 12], which quantifies how well the cement has bonded to the casing, has been used for this purpose. Additionally, CBL tools typically provide the variable density log (VDL), which allows the evaluator to see the measured waveforms for each depth. CBL and VDL are considered standard logs, and most suppliers offer logging services with these tools.

Both the CBL and VDL logs have a limited ability to capture physical features that are only present on a limited angular interval at a certain depth, for instance that the cement is only present on one side of the casing. This is because the CBL is based on a sonic measurement where a wave propagates vertically along the whole circumference of the casing, meaning that the measurement lacks directivity.

The ultrasonic logging tool overcomes this limitation by making multiple measurements in different azimuthal directions at every depth. Current ultrasonic logging techniques are used to measure e.g. internal and external radii, casing thickness, and acoustic impedance. This means that the ultrasonic log can be used both to inspect the state of the casing, and to classify the material present behind the casing by measuring acoustic impedance behind the casing.

For this thesis, all data have been collected with Schlumberger's UltraSonic Imager Tool (USIT). For the remainder of this thesis, any reference to ultrasonic measurements will be referring to measurements made by the USIT tool. A typical USIT log is shown in Figure 2.2.

2.4 Well log evaluation

Well integrity evaluations are challenging due to the large volume of available data, and are highly subjective [7, 13]. To assess the integrity of a well, one must at minimum have two different physical measurements supporting the interpretation, as specified by the NORSOK D010 standard [14].

As it is impossible to know the physical reality behind the casing, often referred to as the ground truth, evaluators must rely on estimates of the physical properties behind the casing. Further, one must investigate multiple logs to make decisions, for instance one would use both casing thickness logs, radius logs, and acoustic impedance logs to make an assessment of the acoustic impedance behind the casing. This makes the decision making process more complex than if one could examine a single image.

The USIT tool is run with centralizers to ensure that the distance between the transducer and casing is the same as the focal distance of the ultrasonic transducer [13]. This means that, as pointed out by [15], tool

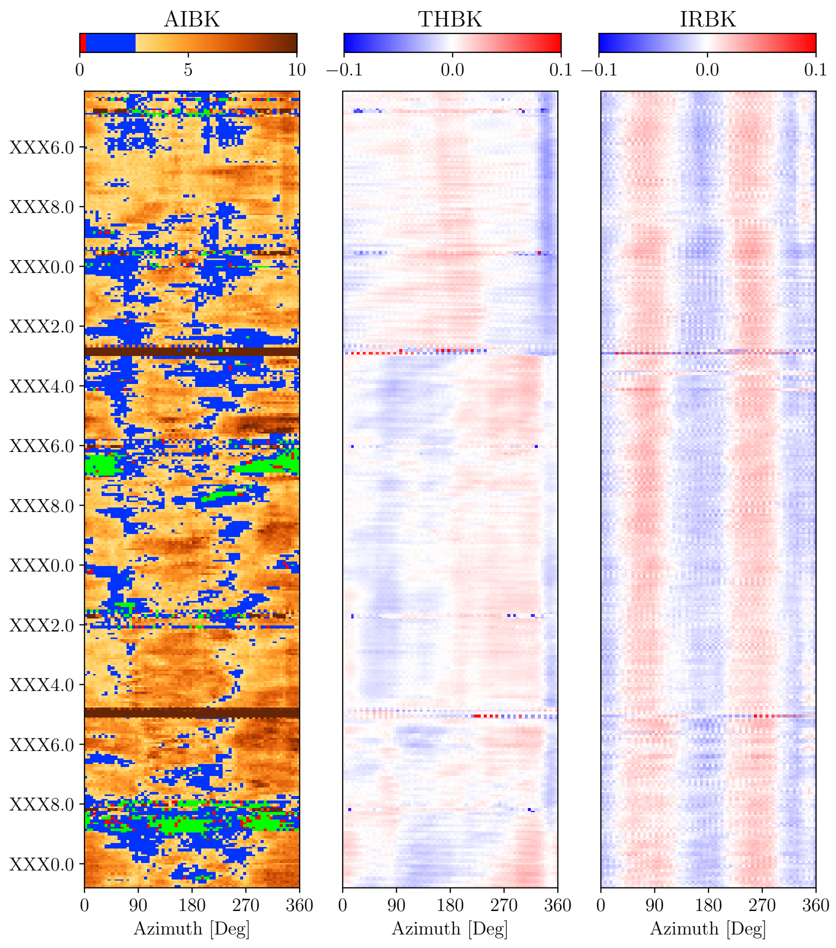

Figure 2.2: Example of an ultrasonic log from the dataset provided by Equinor. AIBK is the impedance behind casing, measured in MRayl, THBK is the thickness deviation from average thickness per depth, measured in inches, and IRBK is the internal radius deviation from average per depth, measured in inches.

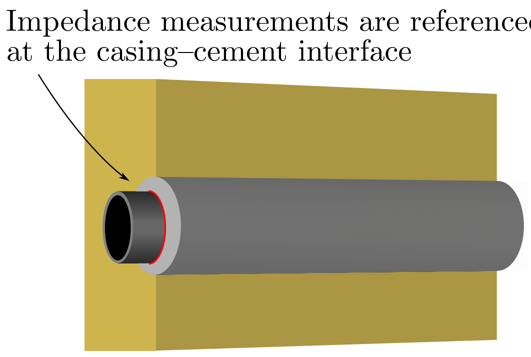
Impedance measurements are referenced at the casing–cement interface

Figure 2.3: A cemented casing section. The acoustic impedance map is referenced at the interface between casing and cement.

eccentering may lead to wrong acoustic impedance estimates, which may cause certain regions of good cement to appear as fluid patches or channels.

Common practice well integrity evaluations is to consider any impedance below 2.6 MRayl to be a fluid, where the interval from 0 to 0.3MRayl to be gas, and the interval from 0.3 to 2.6 MRayl to be water or mud [13, 16].

To make the evaluation easier a custom colormap can be used for displaying the images. In the following gas will be labelled as red, water/mud as blue, and higher impedances will follow a yellow/brown colormap. Further, impedance values below 0 are labelled as green. This can either mean that the data point was missing, or that the correction for nonplanar geometry [15] applied to the impedance estimate has pushed the value below 0. While this ambiguity could cause problems when performing evaluations, it can be argued that in the case of a missing measurement next to a fluid, it is best to assume that there is fluid in the location of the missing measurement as well, as this leads to a pessimistic rather than optimistic evaluation of the well integrity.

2.5 Channel classification

Classifying channels in ultrasonic logs is in the first instance a simple task: In the evaluation we look for any area that can contribute to hydraulic flow. This is possible both for what is typically referred to as a channel, i.e. a vertically connected area of fluid behind the casing, but also for so-called fluid patches of significant size. It is important to note that ultrasonic logging tools are only able to detect the acoustic impedance directly behind the casing as shown in Figure 2.3, meaning that it is possible that channels can exist further into the cement than the location where the acoustic impedance measurement is referenced.

A challenge when classifying channels just based on the acoustic impedance behind casing (AIBK) is that various conditions may affect the acoustic impedance estimate. An example of this is casing grooves. If the well has been drilled further before the logging run, the drillstring may have scraped the casing, creating a small groove. This groove can often lead to lower acoustic impedance estimates than the ground truth. Because the groove will form a stripe, this will have the appearance of a channel even though it is likely not one, as seen in Figure 2.4. Notice that the "channel like feature" in the acoustic impedance image matches with the groove seen clearly in the thickness image. The thickness image is considered more robust than the acoustic impedance image [15, 13]. As such, the acoustic impedance values in the area of the casing groove cannot be trusted. However, there may still be a channel behind this area. To evaluate this, one must consider the appearance of the surrounding areas, looking for fluid patches and channels where the AIBK estimates are reliable, and use this to make a decision on whether there may be a channel behind the casing in the area of the casing groove.

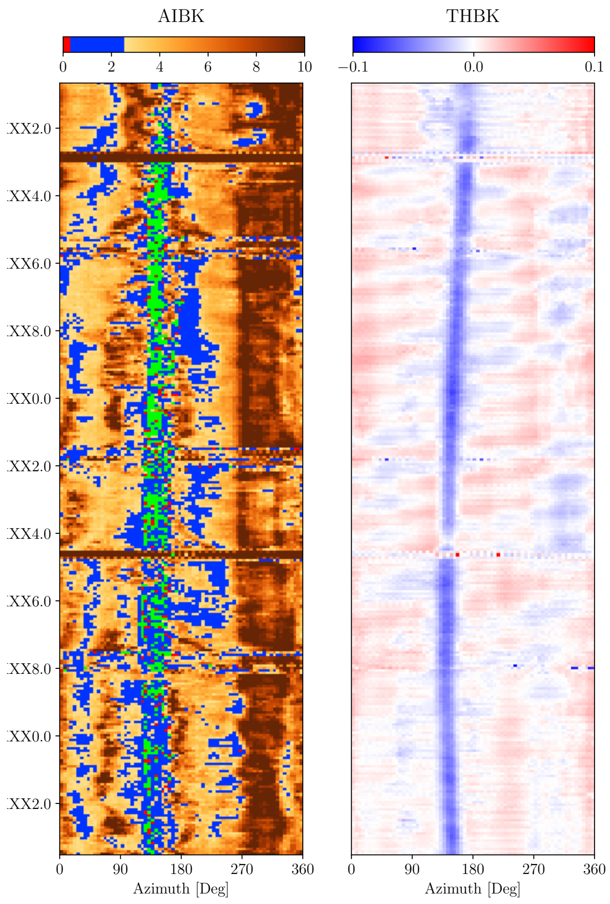

Figure 2.4: An example of a casing groove. Note how the acoustic image shows what appears to be a fluid channel in the same location as the casing groove.

Chapter 3

Machine learning

Machine learning describes the process of teaching a machine to recognize patterns. This chapter will provide a brief introduction to the theoretical side of machine learning, and discuss concepts important to the work in this thesis.

Machine learning is a field in artificial intelligence which attempts to allow computer systems to learn from experience. Machine learning is divided into three branches; supervised learning, unsupervised learning, and reinforcement learning. Supervised learning is performed by supplying labelled examples, where the goal is to learn the relationship between the data and the corresponding labels. For unsupervised learning, no labels are supplied, meaning that the goal is to find patterns in the data, without the model being able to understand what these patterns mean. Reinforcement learning is inspired by human learning. Here, a decision problem or data is supplied. The model then makes a decision based on the information supplied. At some later stage, either immediately or after several more decisions, information is supplied about how successful the decision was. This chapter will focus mostly on supervised learning, as this is the most relevant for this thesis.

3.1 The supervised learning problem

The supervised learning problem may be stated as follows: We seek to learn an unknown target function f that maps data points from the input space $\boldsymbol{\mathcal{X}}$ onto the output space $\boldsymbol{\mathcal{Y}}$. A set of data points x are collected from $\boldsymbol{\mathcal{X}}$. Additionally each data point in x is assigned a label, which is added to y. The goal is to learn the relationship between x and y, that is, approximate $f : \boldsymbol{\mathcal{X}} \rightarrow \boldsymbol{\mathcal{Y}}$. To achieve this, a set of hypotheses for approximating f is proposed. Normally the hypothesis set is the parameter space of a parametric model. For example, a linear model may be chosen as the hypothesis set. In that case, the hypothesis set is $\mathbb{R}^{n+1}$ where n is equal to the dimension of $\boldsymbol{\mathcal{X}}$. In order to pick a final hypothesis from the hypothesis set, a learning algorithm is used. The purpose of this algorithm is to pick the hypothesis h from the hypothesis set that provides the best match to f. Because f is unknown, quantifying this "match" is data driven, measuring the similarity between $h(x)$ and the assigned labels y. This similarity measure will normally be domain-specific.

3.2 Semi-supervised learning

For many applications, labelling of data (assigning labels y to data points x) is very time consuming. This presents challenges for supervised learning, where data availability heavily affects the ability to train a well-performing model. Unsupervised learning does not suffer from this problem, as the data does not require manual assessment. This means that as long as data is available, it can be used for learning. However, since unsupervised learning only looks for general patterns, these will not necessarily contain the information of interest. Further, manual intervention is normally required to analyze the meaning of the classifications produced by an unsupervised learning model. It is clear that supervised learning is necessary when the objective is to detect specific features.

Semi-supervised learning is a technique based on using unlabelled training data to improve the out-of-sample performance of a given supervised learning model. A simple option is to use pseudo labels [17]. This is performed as follows: After training the model to a point where classifications are reasonably precise, the training is halted. The model is then used to classify a number of unlabelled data points. These classifications are set as the labels for the data points, and the data points are added to the training set. Thereafter the model is trained for a number of iterations before the process is repeated.

As explained by Arazo et al. in [18], pseudo labels introduce confirmation bias to the model, because the model will be trained with some wrong predictions. An option to deal with this problem is to give more weight to labelled samples than unlabelled samples in the learning process. [18] uses convex combinations of labelled and unlabelled data in order to alleviate this problem, however this will not necessarily work on more complex data than simple points in a space, because creating superpositions of complex data is often not possible.

3.3 Learning algorithms

Learning algorithms vary widely depending on the model type. For linear regression the least-squares solution may be obtained by simple matrix multiplication [19]. However, for many modern machine learning models, the learning problem is solved by mathematical optimization, a process often referred to as “training”. Many such methods are stochastic, meaning that there is an element of chance in how the algorithm searches for the optimum. This often helps the algorithms avoid local optima, and speed up learning in regions with a small gradient. However, it can also mean that the algorithm is not able to properly converge to the optimum. A solution to this is to use checkpoints, saving the best solution so far as the training is performed. This way, if the learning fails to converge properly, or perhaps even diverges at some point, the best solution is still retained.

3.4 Model testing

For any learning problem, the goal is to maximize out-of-sample performance. This means that contrary to what the training setup indicates, the goal is not to maximize performance on the training set, so-called in-sample performance, but rather to maximize performance when the model is exposed to new data.

Because the training data are only a sample from the distribution of input data, and often influenced by noise, a perfect fit to the training data does not necessarily translate into good real world performance. A model of complexity higher than the complexity of the “true” model f, will tend to fit not only to the general patterns, but also to noise and other errors in the data.

To verify that the model obtained through training is indeed a good fit, it is desirable to test it. This is done by splitting the data set before training into a “training set” and “test set”. Here, the training set is used for training the model, after which it is tested using the test set. Because the model has not been exposed to the test data before testing, this can be used as an estimate of out of sample performance.

3.5 Validation data

In many machine learning problems, the model capacity is higher than that of the true model f. This is often necessary when the model used for approximating f is not of the same kind as f. For example approximating a sinusoidal function with a polynom will, depending on the domain of interest, require many extra parameters to provide a good fit. With insufficient data, this can lead to the model fitting to artefacts and noise in its training examples, which does not translate to out of sample performance. This is commonly referred to as overfitting. A common solution to avoid overfitting, is splitting the dataset used for training into a training set and a validation set. The validation set is used for testing the model after each training step. For each training step, the model is tested on the validation data, which

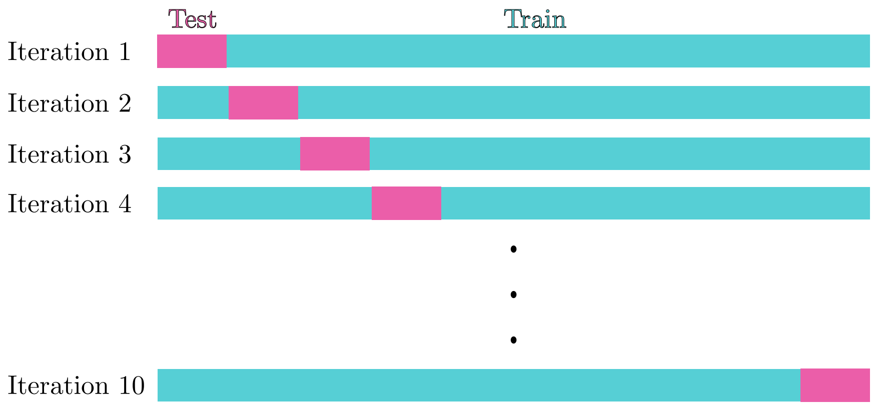

Figure 3.1: An illustration of 10 fold crossvalidation. The data are randomly split into 10 folds. For each iteration one of the folds serves as a test set, whereas the remaining data are used as the training set.

can be seen as an estimate of the out of sample error, much like the test data in the previous section. By employing checkpoints based on the model performance on the validation set, one can retain the last solution before the model started overfitting. This is because overfitting is characterized by the performance increasing when measured on training data, and decreasing when measured on validation data (or other data that are not part of the training).

3.6 Resampling methods

Resampling methods are used to obtain better information about a model's performance. Such techniques are particularly useful if the volume of available data is limited, as it gives a measure of how the model reacts to variations in the input data. This section will describe cross-validation and bootstrapping, which are two of the most common resampling methods.

3.6.1 Cross-validation

Cross-validation involves splitting the data set into n equally sized "folds" or subsets. The model is then trained on the complement of the fold, and then tested on the fold, successively over all folds. The performance of the model may then be estimated by averaging the test statistic over all folds. Common forms of cross-validation are 5-fold cross-validation, in which the data are split into 5 folds, 10-fold cross-validation, and leave-one-out cross-validation (LOOCV), in which each fold is just a single data point. An illustration of crossvalidation can be seen in Figure 3.1

3.6.2 Bootstrapping

Another option for resampling data is bootstrapping. For each bootstrap sample, a number of data points are picked from the data set with replacement. The data that have not been sampled constitute the test set for the given bootstrap sample. Typically one will use several hundred or more bootstrap samples for estimating model parameters or properties such as performance.

Chapter 4

Deep learning

This chapter will introduce the area of machine learning called deep learning, describing common artificial neural network architectures, as well as how these may be used for feature detecting in images.

In recent years, advances in the the field of deep learning has caused a paradigm shift in the field of image analysis. Deep learning based methods have shown a great ability to solve widely different tasks in image classification, image segmentation, and object detection. This chapter will introduce artificial neural networks, and describe the development from simple, fully connected neural networks, up to fully convolutional neural networks for image segmentation.

4.1 The artificial neuron

The basic building block of artificial neural networks is the artificial neuron. The artificial neuron works as shown in Figure 4.1. The artificial neuron receives N inputs $x_1, ..., x_N$. It then computes a weighted sum of these $z = \sum w_i x_i$, where w_i denotes the weights. Further, a bias term b is added to the weighted sum. In order to make computations simpler, a reorganization is sometimes referred to as the "bias trick", is applied. Rather than adding a bias term, the input vector $\mathbf{x}$ gets an extra term $x_0 = 1$. This means that the bias is now the element of the vector of weights. This means that the input vector is now $\mathbf{x} = [1, x_1, ..., x_N]^T$, and the weight vector is $\mathbf{w} = [w_0, ..., w_N]^T$. Finally, an activation function $f(\cdot)$ is applied to the result of the weighted sum, which yields the output y of the neuron.

Traditionally the sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-cz}}$$

has been used as the activation function in neural networks. Here, the c is some constant that is picked before training. For simplicity it may be left

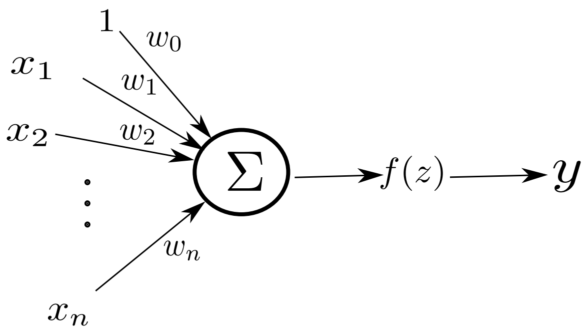

Figure 4.1: The layout of an artificial neuron. A weighted sum is applied to the input with a bias term. Then the activation function is applied, and the returned value is output from the artificial neuron.

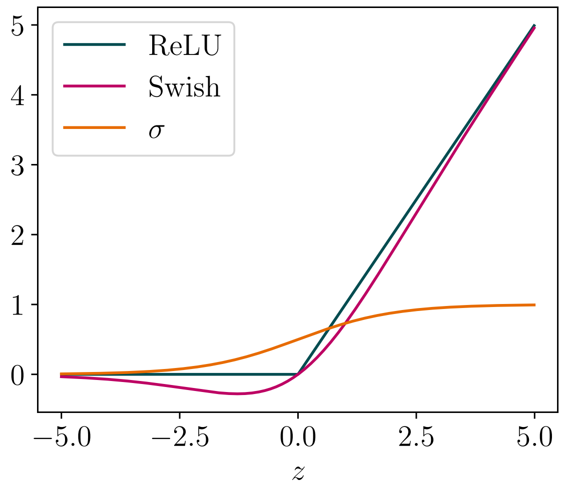

Figure 4.2: Comparison of the sigmoid activation function, the ReLU, Swish.

as 1. The function gets its name from s-like curve it produces, as seen in Figure 4.2. The sigmoid function does however suffer from problems, most commonly the ”vanishing gradient problem”. As is clear from Figure 4.2, if z is either small or large, the gradient of $\sigma(z)$ is small, which means that optimization of the neuron will be slow, as will be explained further later.

In recent years a new activation function called the rectified linear unit (ReLU) has grown popular. The ReLU is defined as

$$\mathrm{ReLU}(z) = \max(0, z).$$

The ReLU does however still suffer from the vanishing gradient problem. This has given rise to a number of fixes. The first is the ”leaky” ReLU

$$\mathrm{ReLU}_{\mathrm{leaky}}(z) = \max(az, z),\ 0 < a < 1.$$

In the leaky ReLU the constant a is a hyperparameter, meaning that it is not trainable, but rather picked ahead of time. Alternatively, the *parametric* ReLU leaves a as a trainable parameter.

Both the ordinary ReLU and the leaky ReLU suffer from the problem that for $z = 0$, its gradient is undefined. This is solved by the swish activation function [20], defined by

$$\text{Swish}(z) = \sigma(z)z.$$

This function can be seen as a smoothed version of the ReLU, as is shown in Figure 4.2.

4.2 Artificial neural networks

By combining multiple artificial neurons, a neural network is created. In the following a form of neural networks referred to as "feed-forward" neural networks will be presented. These are characterized by a topological organization where there are no circular connections. The network will receive its input in one end, propagate the signals through, and provide an output in the other end.

4.2.1 Fully connected neural networks

The conventional fully connected network architecture is shown in Figure 4.3. For this architecture neurons are organized into layers. There is one input layer that receives the input vector to the network. Then each input layer neuron outputs its output value to all neurons in the next layer. The final layer of the network is referred to as the output layer. Normally, the output layer will have the same number of neurons as the number of classes in the classification problem. Here, each neuron represents a certain class, so that high activation of given neuron corresponds to the presence of its corresponding class. This is commonly referred to as "one-hot" encoding. At the output, each class is given a value between 0, and 1. In a single–class problem, this value is thresholded to retrieve the classification. This threshold is often set at 0.5, however it can be augmented to serve specific needs. For instance, in safety critical contexts, where detecting anomalies is important, one may move the threshold to minimize the risk of false negative findings, accepting that this will increase the number of false positive findings.

Fully connected neural networks have a high flexibility for modelling complex features. However, as the number of neurons per layer, or the number of layers grows, the number of parameters in the network grows quickly. This means that there is a limit to how complex networks with this architecture can be before they become difficult or impossible to train. This

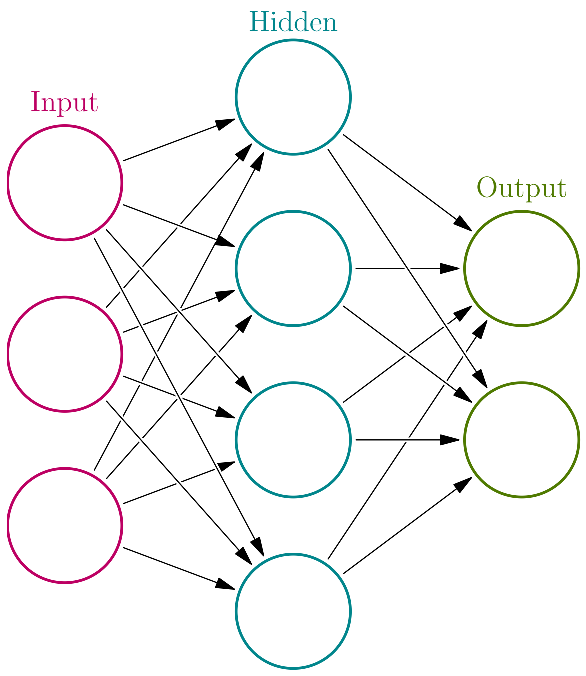

Figure 4.3: Example of a fully connected neural network with 1 hidden layer. Obtained from [21].

is problematic for classifying images because a typical image will be on the order of 10000 elements, meaning that a large number of neurons will be necessary for analyzing the image.

4.2.2 Convolutional neural networks

Convolutional neural networks are based on the observation that when classifying an image, a certain indicative feature may appear at various locations in the image. In a fully connected network, there must be multiple neurons to detect such a feature, because each neuron can only ”see” a certain location in the image (as well as its neighborhood) in order for the location of that feature to be known. Alternatively, the neuron can detect the presence of such a feature anywhere in the image. However, this means that the feature detected by the neuron cannot be localized, which means that downstream neurons cannot use information from this neuron for detecting compositions of features.

The convolutional neural network solves this problem by replacing the fully connected layers with sets of convolutional kernels. A convolutional kernel works as shown in Figures 4.4 and 4.5.

The kernel will see only a small portion of the image at a time, and apply a weighted sum to the values of the pixels, in the same fashion as the fully connected layers in the previous section. However, this kernel is moved across the image, where the weights of the convolutional kernel are the same every time, generating a new “output image”. A 3x3 kernel will only have 10 free parameters, i.e. 1 parameter for each pixel covered by the kernel, as well as the bias term. This means that one may typically use quite a few convolutional kernels and not come close to the number of parameters required for a single fully connected layer on a normal size image.

In addition to convolutional kernels, convolutional neural networks often use “pooling” layers in order to reduce the image size. This is useful to capture features on a larger scale. The pooling layers use a sliding window, much like the convolutional kernel, however pooling layers normally don’t use any parameters, but are rather based on simple mathematical operations. Some of the most common forms of pooling are average-, and max pooling. Average pooling outputs the average of its input values, whereas max pooling outputs the maximum of its input values.

While convolutional layers most often slide the kernel one pixel at a time for each pixel in the output image, it is common for pooling layers to slide the same number of pixels as the dimension of the pooling window. That is, if using a 2x2 max pooling layer, the step for each output (stride) will be 2 in the x-direction, and 2 in the y-direction. A one dimensional example of this is shown in Figure 4.6. For more details, see [22].

Each layer in a convolutional neural network will normally have multiple convolutional kernels, more commonly called filters. This allows each filter

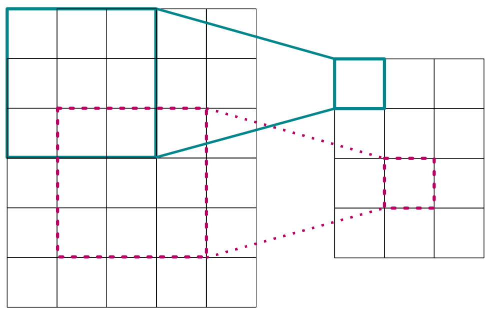

Figure 4.4: Example of how the input and output from a convolutional layer relate.

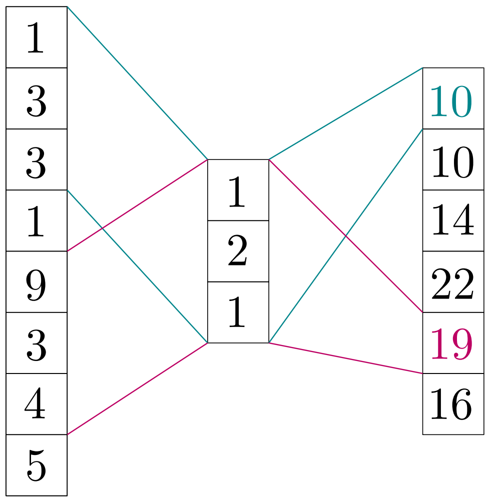

Figure 4.5: Example of a convolutional kernel.

to become specialized at detecting certain features (for instance edges or curves). Similar to how the human eye works, the early layers of a convolutional neural network will tend to detect "basic" features such as edges or ridges, whereas layers closer to the output will tend to focus on higher level, more complex features [23, 24]. In networks made for image classification, this means that the architecture will typically comprise a number of convolutional layers, often with some pooling layers in between, followed by a few fully connected layers to form the output of the network [25]. As will be discussed more later, an alternative to this is the fully convolutional network. Fully convolutional networks consist only of convolutional and pooling layers, which means that the output from the network will be another image. This is useful for tasks like image segmentation.

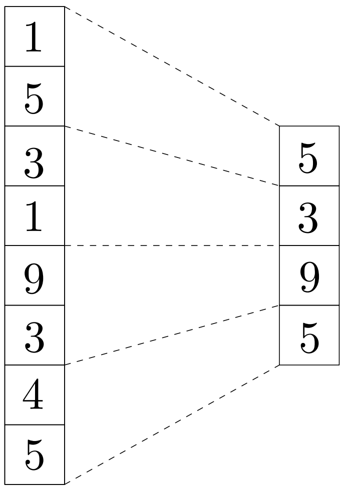

Figure 4.6: Example of a max-pooling operation.

4.3 Training neural networks

The learning process of a neural network is performed through what is commonly referred to as training, as explained in Section 3.3. Training simply entails performing a mathematical optimization of the neural network's parameters with respect to some objective. In a supervised learning context, this objective is a loss function, that is, a function quantifying the penalty for discrepancies between the target output and the output of the neural network. Loss functions are specific to the application. For ordinary classification problems, a simple option is to use the mean squared error of the classification, or the slightly better crossentropy.

The gradient of the loss function may be found using a technique called backpropagation. The idea behind this is that the gradient corresponding to the parameters of each layer in the network only depends on its current weights, current activation (the data that have been propagated forward through the network), the gradient of the activation function used, and the gradient of the layer one step closer to the output. By using this, one may start at the output of the network, and then compute the gradient corresponding to each layer by propagating the gradients back to the previous layers successively. For a more in-depth explanation of backpropagation, please see [22] or [26].

Utilizing backpropagation, one may feed data through the network, compute the loss at the output, and then perform backpropagation in order to find the gradient of the loss function given the data, which can be used to minimize the loss using gradient descent. However, feeding large volumes of data through the network for computing the gradient at every training step is computationally expensive. To circumvent this problem, only a small, randomly selected portion of the data is fed through the network at every training step. This small portion of data is often referred to as a batch. This approach to training neural networks is called stochastic gradient descent. There exist many more advanced methods based on these principles, most notably step size-adjusting algorithms such as RMSprop and Adam. For a description of these, see [25].

4.3.1 Transfer learning

As mentioned earlier, convolutional neural networks tend to contain filters that are more general in the first few layers, detecting basic features. Conversely, layers close to the output will tend to focus more on higher level features, more specific to the domain of application. Because of this, a network trained for any arbitrary task will tend to have filters well suited for any application in the layers close to the input, whereas the layers close to the output will be more specialized. As pointed out by [27], gradients become increasingly small when backpropagating through the network, meaning that the layers close to the input are far slower to train than layers close to the output. These moments are leveraged by a technique called transfer learning. Here, one first trains the network on a dataset separate to the training data acquired for the specific problem. This will serve as a feasible initialization for the final training on the domain-specific data.

4.3.2 Constraining the parameter space

Normally, a rule of thumb in any learning problem is that the number of data points should be significantly higher than the number of learnable parameters. However, in deep learning, this is often not the case, as neural networks often have several hundred thousand learnable parameters. This means that neural networks are prone to overfitting. As mentioned, one of the ways of overcoming this problem is to use a validation set for retaining the best performing set of parameters.

However, another way of constraining the parameter space is regularization. Regularization involves adding an extra term to the loss function, which is dependent on the magnitude of the parameters. This way, parameters that do not directly contribute to reducing the error when training will be driven to zero to reduce the loss.

Another method that seeks to solve the problem of overfitting with large parameter spaces is dropout. Dropout works by randomly disabling a percentage of neurons of the network during each epoch. If a neuron has grown highly specialized to detecting a specific feature in only one data point, then its downstream neurons will also rely on this representation. If that neuron is disabled, all neurons that relied heavily on it, will be forced to rely on a wider selection of neurons, which will tend to make the network more general, which often leads to higher out of sample performance.

4.3.3 Kernel initialization

Care must be taken when initializing the weights of a neural network. If not, the gradient of the network may end up being either very small, often referred to as a vanishing gradient, or very large, often referred to as an exploding gradient. This is solved by initializing the network with random weights. This randomization must be performed carefully to control the initial gradient of the network. For this thesis, the initializer developed by He et al. is used [27].

4.4 Image segmentation

Image segmentation is the task of classifying every pixel in an image. This means that it can be used to precisely describe the composition of an image, i.e. which features or objects are present in the image, and which pixels "construct" each feature.

Traditionally, image segmentation has been based on different clustering methods such as K-means, histogram methods, or thresholding. Another approach has been using edge detection to differentiate between clusters. However, such methods have not been able to detect complex patterns in the data beyond finding pixels of similar value. Deep learning has changed this by introducing its ability to detect complex structures in images. Ciresan et al. [28] proposed a deep learning approach to segmenting images by using a sliding window, classifying each pixel. The drawback of this method is that there is a lot of redundant computation since all computational steps are repeated for every pixel, ignoring the fact that the windows overlap. Further, as was pointed out by Ronneberger et al. [29], there is a tradeoff between localization accuracy and context. Using a large window provides a lot of context, but the necessary max-pooling will reduce the localization accuracy since the part of the input dominating the resulting prediction may not necessarily be in the location of the pixel of interest.

This problem was solved by Ronneberger et al. [29] proposing the U-Net architecture. This architecture can be seen as a two stage process: First, the image is downsampled through a series of convolution and max pooling layers. This produces a coarse feature map with a high number of channels. After this the image goes through the same process in reverse, but with the max-pooling layers replaced by upconvolution layers. Upconvolution works by taking a single input pixel and returning a filter multiplied with that pixel value, as shown in Figure 4.7. After each up-convolution, the image is combined with the final image from the downsampling process at the same resolution before the two convolutions. This finally produces the output segmentation map of classifications. It is worth noting that for each level

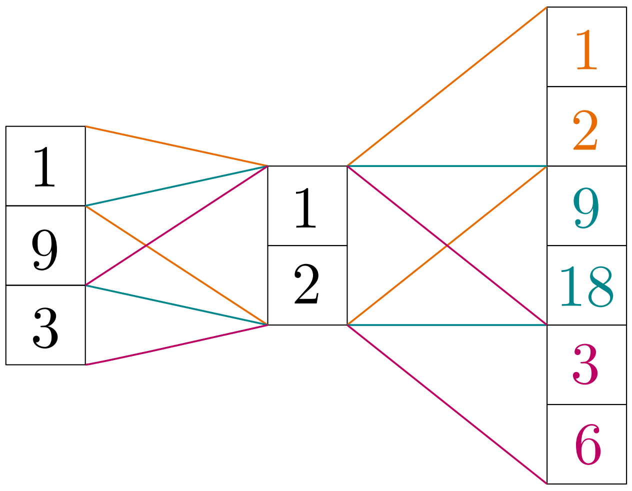

Figure 4.7: Example of upconvolution in 1 dimension.

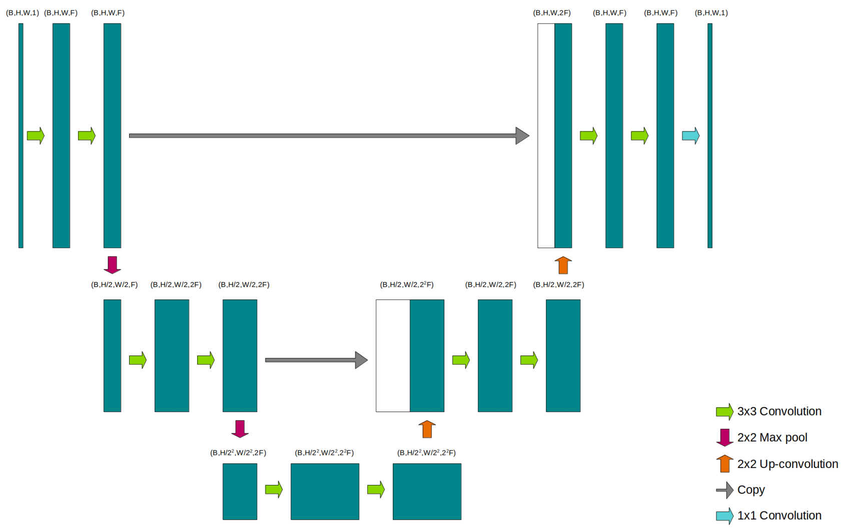

Figure 4.8: The Unet architecture used for this thesis. The tensor shape is shown for each layer in the model. B is the batch size, H and W are the image height and width, respectively, and F is the number of filters used in the convolutional layers.

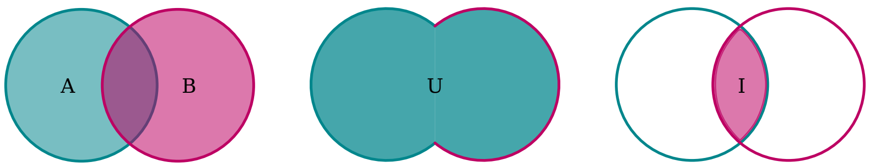

Figure 4.9: Illustration of the intersection $I = A \cap B$, and union $U = A \cup B$ between sets A and B.

of the network, the image dimensions are reduced by factor 2, as shown in Figure 4.8.

4.4.1 Metrics for image segmentation

A key part of machine learning is evaluating the performance of a model, both under training and after it is trained. Usually one makes the distinction between metrics, which one will generally seek to maximize, and losses (loss functions) which one will seek to minimize. The most common scheme for this is the mean squared error between the ground truth and the prediction, as well as accuracy, which is often defined as the fraction of correct classifications.

However, this is not ideal for segmentation. One reason for this is that often the image is highly unbalanced in terms of the number of pixels with each label. For the well logs, it is typical that less than 10 % of the pixels are labelled as a channel, that is, they have value 1 in the label image. This means that the network would achieve a very high accuracy simply by classifying every pixel as not a channel (i.e. output 0 for every pixel. If we consider an image consisting of only 5% labels, this raises the question if a network labelling every pixel as not a channel should be considered to be 95% right (accuracy), or 50% right (50% of classes identified correctly). In response to this, a metric which takes the distribution of the number of different labels into account is preferable.

For this purpose, a good option is the Intersection over Union metric,

$$\text{IoU} = \frac{|I|}{|U|} = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|},$$

where I is the intersection, U is the union, A and B are the sets evaluated, and $|\cdot|$ denotes the cardinality of a set, that is the number of elements in a set. A geometric representation of intersection and union are provided in Figure 4.9. A comparison of the IoU metric and Accuracy is provided in Figure 4.10.

Another option is the Dice coefficient

$$\text{Dice} = \frac{2|A \cap B|}{|A| + |B|}.$$

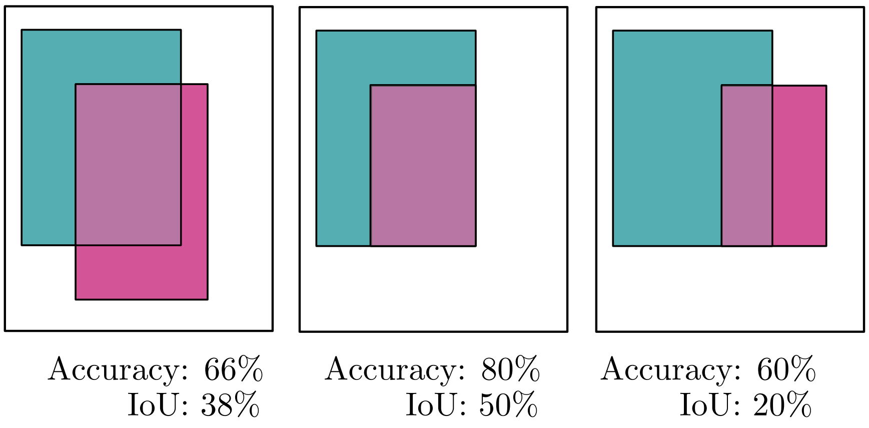

Figure 4.10: Demonstration of IoU, and how it differs from accuracy.

These are quite similar metrics, and will give the same results in extreme cases (every pixel classified as negative or every pixel classified as positive). The IoU metric punishes worst-case performance more than the Dice coefficient, much in the same way as L^2 norms punish large deviations relatively more than L^1 norms. With a single class classification problem, this distinction will not necessarily make any difference for the classification results, however it will tend to draw the output pixel values closer to 0.5. This means that if the output values are treated as fuzzy, the uncertainty of the classification may be quantified. The loss function used for this project is the Jaccard distance, which is one minus the Intersection over Union metric. However, the Jaccard distance is not smooth. This is problematic because training the networks requires the gradient of the loss function. If the loss function is not smooth, the gradient will not be defined for all input values to the loss function. To solve this problem a smoothing coefficient is added. This gives

$$\text{Jaccard}_{\text{smooth}} = 1 - \frac{|I| + s}{|U| + s},$$

where s is the smoothing coefficient.

Chapter 5

Log data and imaging

This chapter will describe how the log data are handled, as well as present tools for analyzing log images at at higher resolution than the original logging resolution.

5.1 Data handling

Most well logging data are provided in files following the DLIS standard formally known as the API RP66 standard [30]. DLIS-files can be read by the dlisio python library [31]. However, due to the difficulty of manually inspecting the contents of these files, we transfer the raw data into the HDF5 file format [32]. This enables faster access to the data in the file, as well as letting the user open the file in an appropriate reader to inspect the contents visually. Furthermore, the dlisio package is still in alpha, meaning that converting the data to HDF5 provides better reliability because the implementation will not be dependent on every new version of dlisio working the same. In order to save time, the conversion software written for this project only transfers the data necessary for this project to the HDF5 format. A library for reading DLIS data stored in a specific HDF5 layout called dlish5 was obtained from Erlend Viggen [33]. In order to access the desired data as easily as possible, a new class was implemented for accessing the data needed for this project easily, utilizing some of the functionality already implemented in the dlish5 library.

DLIS files contain data channels with different vertical resolutions. These data channels will be organized in DLIS frames, where each frame is a collection of data channels with the same vertical resolution, and measurements collected at the same depths. Common vertical resolutions are 0.5-, 1-, 2-, 3-, and 6 inch spacings, provided in the corresponding 5B, 10B, 20B, 30B, and 60B data frames.

To ensure consistency, as well as to provide a storage site for image annotations, a new frame is created in the HDF5 file. All data relevant to

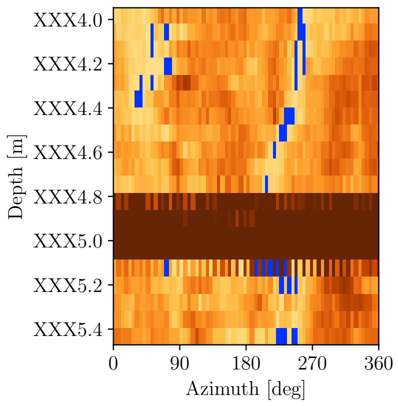

Figure 5.1: Image of a casing collar. Notice the alternating high impedance measurements along the edges of the casing collar, indicating that the measurements have not been taken sequentially from one side to the other. This log segment is from the dataset provided by Equinor ASA.

the annotation are corrected, and then added to this frame. This ensures that one will always have access to the data on which the annotations are based, and that retrieving data is as fast as possible, without requiring corrections to be applied every time.

Before evaluating, the image logs are corrected for rotations of the tool. This is done by finding a data channel which contains the rotation of the "base direction" of the tool. Further, the image is rotated at each depth according to the given rotation to retrieve the corrected image. For the rotation, the UCAZ (ultrasonic azimuth) channel is preferred, and if not present, the RB (relative bearing) channel is used. The UCAZ channel is preferred as it produces images that better match the rotation corrected logs presened in official log plots.

5.2 Measurement locations

When evaluating log images at the scale where pixels are clearly distinguishable, the exact reference location of each measurement becomes an important factor in evaluating material properties behind the casing. Figure 5.1 shows an image of a casing collar, where the alternating pattern on the edges of the high impedance region suggest that the measurements are not collected in perfect rows. This means that the measurement locations used for constructing the image must be considered.

For the remainder of this chapter, we will analyze a short log segment from the dataset provided by Equinor ASA, shown in Figure 5.2. The UTIM channel contains the arrival times of ultrasonic pulse echoes for each mea-

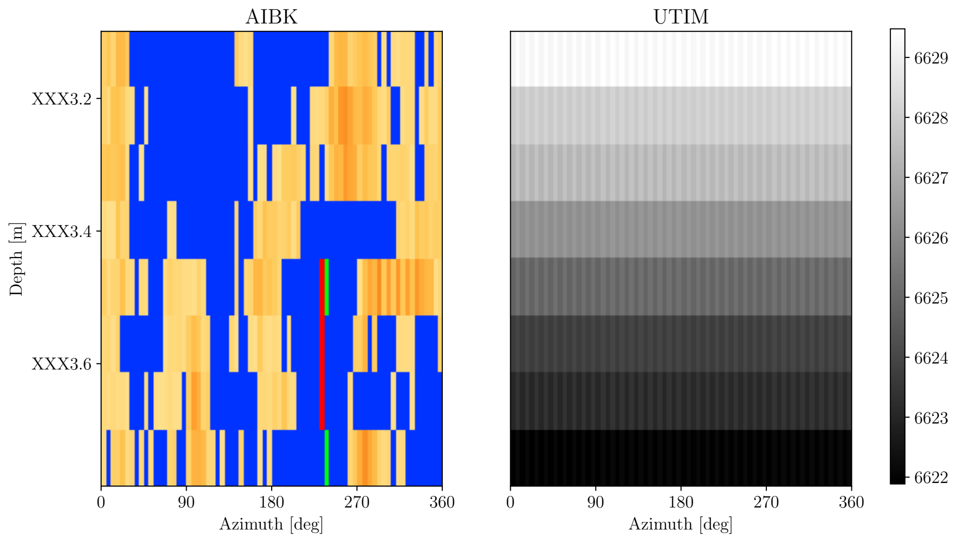

Figure 5.2: Example of how UTIM data are stored. The left plot shows a raw AIBK image, and the right plot shows an image of the UTIM channel using a grayscale colormap. As can be seen, there is one UTIM value for every AIBK value, meaning that the measurement time for each individual measurement is stored.

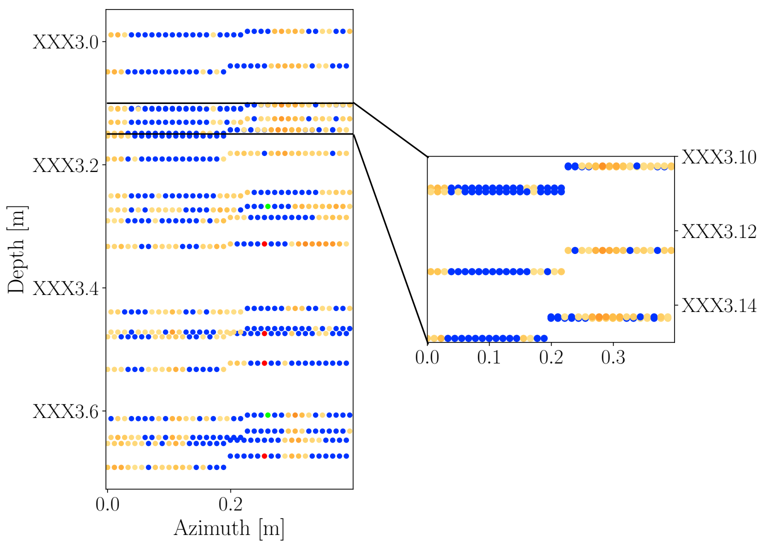

Figure 5.3: Depth corrected measurements from speed corrected depth using UTIM for each measurement row. The zoomed plot shows that measurements from different rows mix, meaning that there is disagreement between SCD and the combination of UTIM and cable speed (CS).

surement [34]. Figure 5.2 shows an example of the contents in the UTIM channel. As can be seen, there is a time measurement for each individual ultrasonic measurement (such as acoustic impedance). This means that the channel can be used to obtain additional information about where each measurement is collected compared to the conventional depth channels. The depth of each measurement is corrected by using the cable speed as an estimate of the tool speed in the measured depth direction (speed along the well path), as there are no accelerometer data available in this log. Figure 5.3 shows measurements based on the speed corrected depth (SCD) channel, where all measurements on each row are vertically corrected based on the time difference from the first time recorded on the row, to the time recorded for the given measurement, using the cable speed (CS) as an estimate of the tool speed. The correction is given by

$$z_{\text{corrected}} = z_{\text{begin}} - \text{CS}(z) * (t - t_{\text{begin}}), \tag{5.1}$$

where z_{begin} is the reference starting depth (in this case speed corrected depth) for each row in the image, CS is the cable speed, t is the UTIM value for given measurement, and t_{begin} is the starting time (minimum) for each row in the image.

It is clear that there is disagreement between the depths given by the logs, and the combination of ultrasonic wave arrival times and cable speed. For this work, this problem is solved by correcting all measurements using

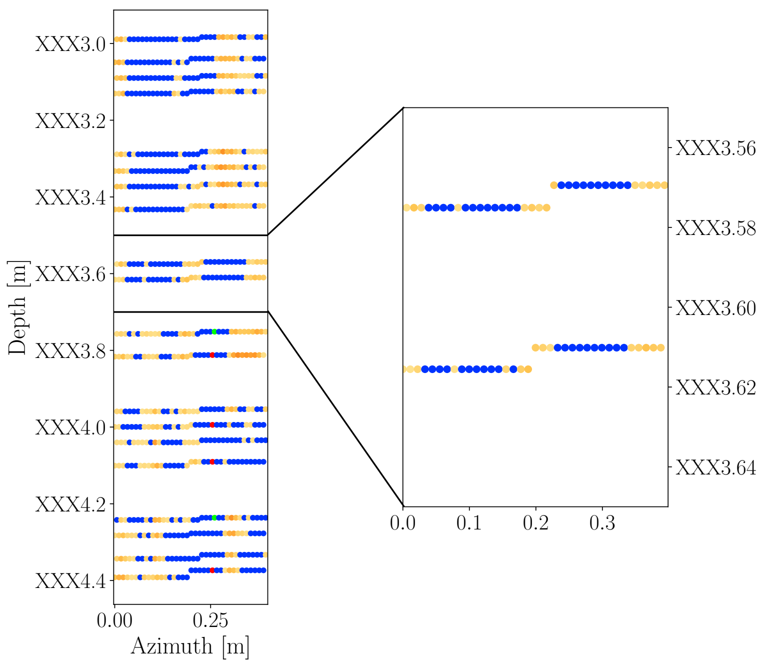

Figure 5.4: Depth corrected measurements using only UTIM and CS. Notice that compared to Figure 5.3, the measurement rows are all separated.

UTIM and a reference starting depth. The correction is given by (5.1), but here, the reference point z_{begin} and t_{begin} is given at the bottom of the logging interval, meaning that all measurements are referenced in the same point.

This gives the measurement locations shown in Figure 5.4. Note that the depth axis is stretched compared to the previous example. By examining this plot carefully, we observe that the measurements still distribute on horizontal lines. From the zoomed-in segment in Figure 5.4, it is clear that the measurements distribute on 4 horizontal lines. We hypothesize that this is caused by infrequent updates of the clock used for filling the UTIM channel. Further, we observe that the measurements from the single row are collected over 2 tool rotations.

When looking at Figure 5.4, it is clear that a depth correction based on UTIM alone is not representative for how the measurements are collected. It is well known that the tool is pulled up continuously, which means that the measurements should form a spiral. Based on this, we try linearly interpolating the recorded times, so that for each set of measurements with the same given pulse arrival time, our corrected arrival times are evenly distributed between their uncorrected value, and the next distinct time value of the row (if it is the last value, the "next" value is found by extrapolating the distinct time values of the row). Because this is only a segment from

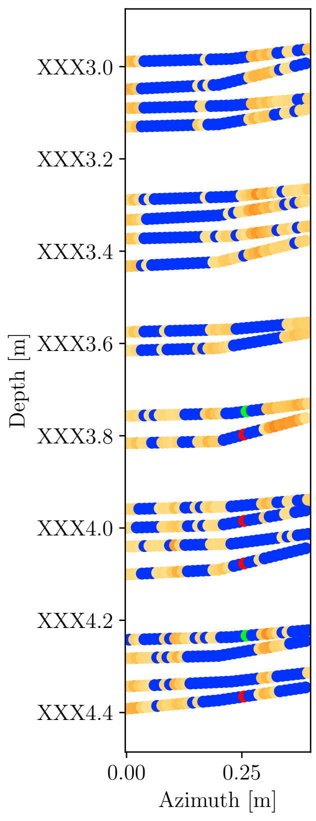

Figure 5.5: Vertically corrected measurement locations using linear interpolation to vertically distribute measurements with the same UTIM value.

the interior of the log, the next time value is available even if it is outside the log segment analyzed here. This is shown in Figure 5.5. As is clear from this figure, the linear interpolation does not always work properly, as it is unlikely that the tool moves faster over one half of the rotation every time it records measurements.

Due to this problem, we test an approach where we only rely on the first UTIM measurement from each row to set the start depth of that row. The remaining measurements are assumed to be collected over 2 rotations. In this case, the arrival times are estimated by using the cable speed as an estimate of vertical tool speed, and the RSAV channel as the ultrasonic probe rotation speed. Further, we rely on the UTIM channel for determining which measurements are collected on the first rotation and which measurements are collected on the second rotation. This correction is shown in Figure 5.6. For the log in question it is clear that this correction is probably the closest to replicating the actual locations of each measurement for this particular log. The preceding examples clearly show that for each measurement row

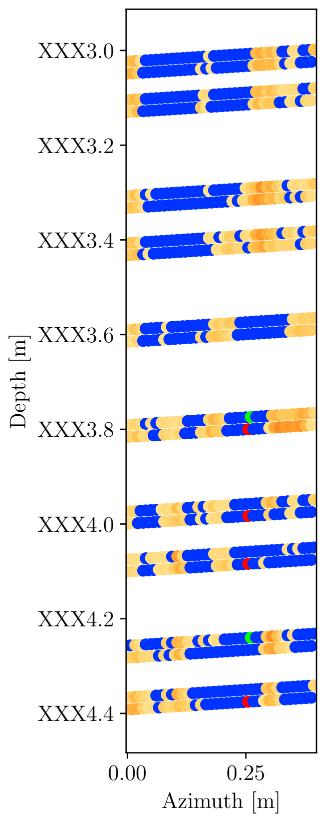

Figure 5.6: Vertically corrected measurement locations using RSAV and CS to vertically distribute measurements from a reference point decided by the first measurement at each depth in the raw data.

the image is collected over 2 tool rotations. Further, it is easy to see that based on this, the conventional way of displaying images is perhaps not the best representation of the physical reality. It is clear that the vertical distance between measurements is far greater than the azimuthal distance between measurements. This means that displaying the measurements as if they were all collected at the same depth is not necessarily suitable.

5.3 Nearest Neighbor interpolation

Image logs are normally displayed as shown in Figure 5.2. This is based on making a grid, and then filling each square of the grid with a color corresponding to the measurement made inside the given square. This is based on an assumption that the measurements are collected on horizontal lines. If this assumption is broken, this approach is no longer a good representation of the physical reality, as the grid should be augmented to better represent where the measurements are made. Based on this, the sampling of the image must be considered.

The original image display is based on a nearest neighbor interpolation, where all measurements are assumed to be made at the same depth. If this assumption is augmented to assume that the measurements are made at the corrected depths, we may construct an upsampled nearest-neighbor interpolated image by

$$\mathrm{NN}(\vec{r}) = \mathrm{IM}(\mathrm{argmin}_{\vec{r_0}}(d(\vec{r}, \vec{r_0})),$$

where NN is the nearest neighbor interpolated image, IM is the set of measurements, $\vec{r_0}$ is picked from the set of measurement locations, and $\vec{r}$ is the interpolation location. The result of this is shown in Figure 5.7.

5.4 Gaussian processes

It is clear that the nearest neighbor is still not ideal for viewing the images, as it is not really a true upsampling, but rather a skewed version of the original image. The problem with upsampling these images is that most methods for interpolating 2D data, such as minimum curvature interpolation, rely on creating a smooth surface that intersects the data points. Creating a smooth interpolation would not be representative of the physical reality, as interfaces between fluids and solids, or between different solids, tend to be sharp. An alternative is to use Gaussian Processes, which in the geosciences is commonly known as kriging. This is a statistical approach to interpolation, in which all measurements are assumed to be drawn from a multivariate normal distribution. One may then estimate the interpolated image by computing the expected value for each point on an upsampled grid. This method is actually a regression method, but as Figure 5.8 shows, it has

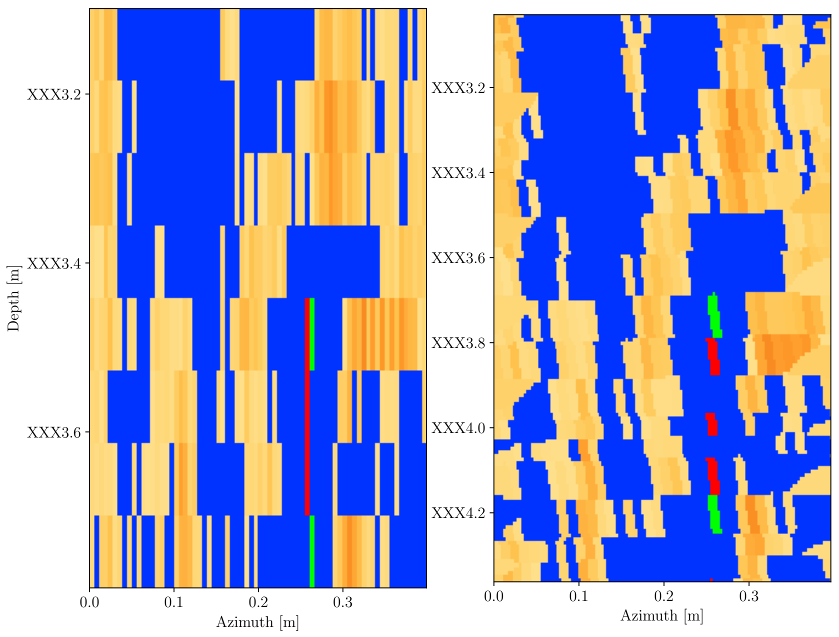

Figure 5.7: Original image compared to the nearest-neighbor interpolated image. Note that the nearest-neighbor interpolated image is stretched in the depth direction compared to the original image.

the flexibility to be used for interpolation. Further, due to the measurements being affected by some uncertainty, a perfect fit to the measurements is not necessarily the best representation of the physical conditions behind the casing.

Gaussian process regression is performed by considering all the points to be estimated as a multi-dimensional Gaussian random variable. By applying some covariance function, as well as the measurements acquired, one may compute the expectation and variance for each point. Figure 5.8 shows an example of how gaussian process regression works in the one dimensional case. In the following, a brief derivation of the process is introduced.

The measurements are collected in the points $\mathbf{x}_0$, and are assigned an a priori expectation $\mathrm{E}(\mathbf{x}_0) = \mu_0$ and covariance $\mathrm{Cov}(\mathbf{x}_0) = \mathbf{\Sigma}_0$. The covariance matrix is constructed using a correlation function and a variance. The variance is either assumed, known from previous data, or estimated.

Similarly, the outputs from the regression are given by $\mathbf{x_1}$, and are assigned an expectation $\mathrm{E}(\mathbf{x}_1) = \boldsymbol{\mu}_1$ and covariance $\mathrm{Cov}(\mathbf{x}_1) = \mathbf{\Sigma}_1$. The covariance matrix is constructed using a correlation function and a variance, similarly to $\mathbf{\Sigma}_0$.

Further,

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \end{bmatrix}$$

is assumed to be drawn from the multivariate normal distribution

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^k |\mathbf{\Sigma}|}} \exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})),$$

where k is the dimension of $\mathbf{x}$ Here, we have

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_0 \\ \boldsymbol{\mu}_1 \end{bmatrix}, \text{ and } \mathbf{\Sigma} = \begin{bmatrix} \mathbf{\Sigma}_0 & \mathbf{\Sigma}_{0,1} \\ \mathbf{\Sigma}_{1,0} & \mathbf{\Sigma}_1 \end{bmatrix},$$

where $\mathbf{\Sigma}_{0,1} = \mathbf{\Sigma}_{1,0}^T$ is the covariance between $\mathbf{x}_0$ and $\mathbf{x}_1$. Thus, the conditional mean and variance of $\mathbf{x}_1$ given $\mathbf{x}_0$ is given by

$$\mathrm{E}(\mathbf{x}_1 | \mathbf{x}_0) = \boldsymbol{\mu}_1 + \mathbf{\Sigma}_{1,0} \mathbf{\Sigma}_0^{-1} (\mathbf{x}_0 - \boldsymbol{\mu}_0)$$

$$\mathrm{Cov}(\mathbf{x}_1 | \mathbf{x}_0) = \mathbf{\Sigma}_1 - \mathbf{\Sigma}_{1,0} \mathbf{\Sigma}_0^{-1} \mathbf{\Sigma}_{0,1}.$$

For more details, please see [35].

The most important part of this process is selecting the covariance function C to be used. This is normally done by using $C(\vec{r_0}, \vec{r_1}) = \sigma^2 c(\vec{r_0}, \vec{r_1})$, where σ^2 is the variance, and c is a correlation function. Ideally this correlation function would be estimated from data, however, for this thesis it turned out to require more computation time than what was available to produce

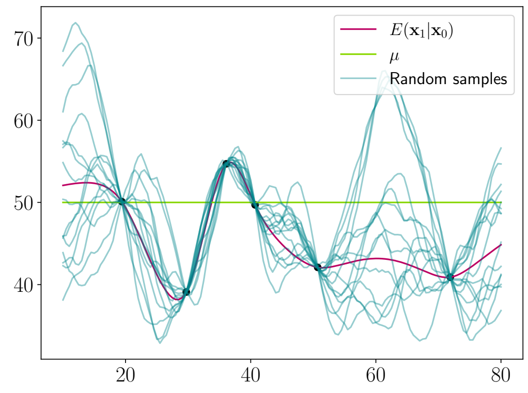

Figure 5.8: Example of a Gaussian process with a constant a priori expectation $\mu = 50$, $\sigma^2 = 4^2$, and a Matérn correlation function with $\nu = 1.5$. The measurements $\mathbf{x}_0$ are marked in black, the red line is the a posteriori expectation $\mathrm{E}(\mathbf{x}_1|\mathbf{x}_0)$, and the blue lines are 10 random realizations.

stable results. We therefore use qualitative methods to choose the correlation function. In this section we will use the nearest-neighbor interpolated image for comparison, as this image is somewhat similar to the raw images as shown in Figure 5.7, while correcting the depth of the measurements to the same depths the kriging uses.

First, the Matérn correlation function [35] is tested, which is given by

$$c_\nu(d) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\frac{d}{l}\right)^\nu K_\nu \left(\sqrt{2\nu}\frac{d}{l}\right) \tag{5.2}$$

Here l is a distance measure, and ν is a smoothness parameter. One can show that for all half integers $\nu = n + 0.5, n \in \mathbb{N}^+$, the Matérn correlation function c_ν is n times differentiable. When $\nu \to \inf$ we get the Gaussian [35]

$$c(d) = e^{-\frac{d^2}{2l^2}},$$

which is also referred to as the squared exponential (SE), or radial basis function (RBF).

When $\nu = 0.5$ the result is an exponential function [35]

$$c(d) = e^{-\frac{d}{l}}$$

When using kriging for interpolation, the expected value image is not sufficient for making decisions. This is because the expected value image will tend to be smoother than the ground truth, and does not capture the variability in the estimates. One could utilize the variance measurement

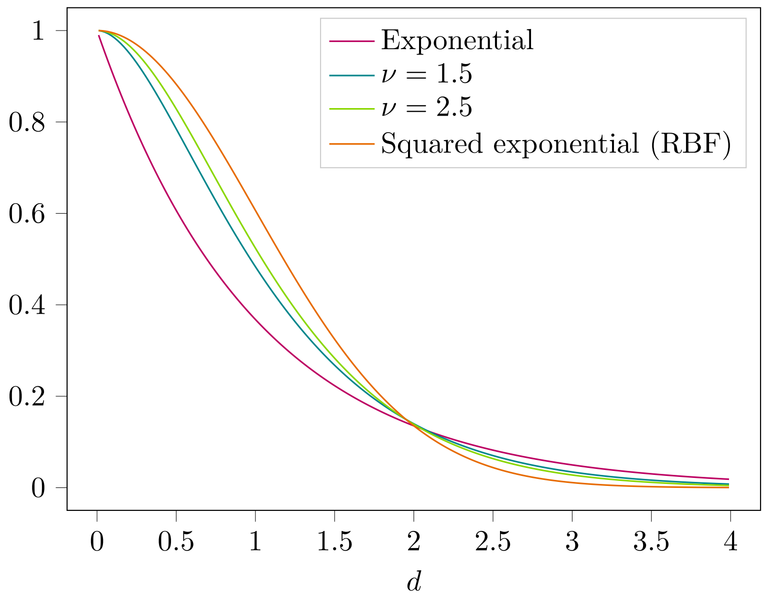

Figure 5.9: Comparison between different Matérn functions with $\nu = 0.5$ (exponential),$\nu = 1.5$, $\nu = 2.5$, and $\nu \to \inf$ (Gaussian).

in each point to create a lower bound to a confidence interval, however this would tend to be an overly pessimistic approach not representative of the ground truth. As discussed extensively in [36], a better approach is to combine the expected value image with multiple random realizations of the image. This way we paint a better picture of what the ground truth possibly looks like, based on the data available. As mentioned previously, this analysis is performed on a short log interval. This is due to the large memory requirements in order to hold the covariance matrix between all the estimated points. This means that in order to work on a larger interval, the resolution would have to be decreased, defeating the purpose of this analysis, which is to analyze the well log on a finer scale.

For all following experiments we use a constant expectation $\mu = 2.48\,\text{MRayl}$, and $\sigma^2 = (3\,\text{MRayl})^2$. These are based on the mean and variance of the measurements on the analyzed interval.

Additionally, a white kernel is added to all convolution kernels. This corresponds to what is referred to as “nugget” in the kriging literature. This stems from the origin of kriging in the gold mining industry, where this noise term would correspond to finding a gold nugget, which would look like a noise point. For the purpose of kriging well logs, this noise term represents the heterogeneity of material in the annulus.

Finally, the library used [37] does not support cylindrical coordinates. For producing the expectation image this is solved by padding the measurements as shown in Figure 5.10. However, the realizations do not obey

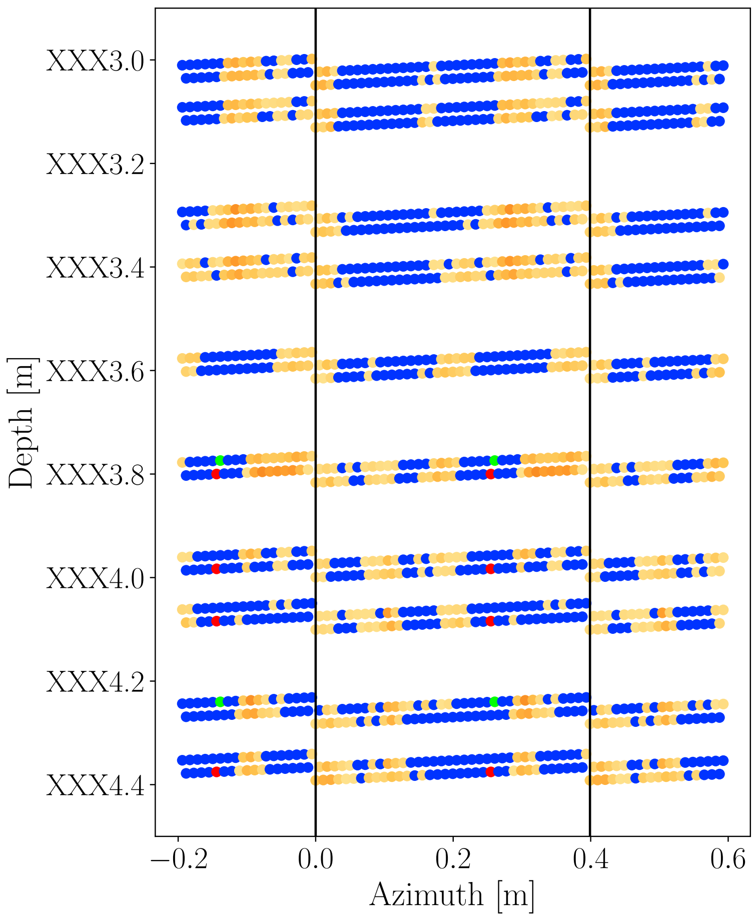

Figure 5.10: Example of measurement padding for making the krig expectation image and nearest neighbor image cylindrical. The black lines mark the edges of where the interpolated image is constructed.

continuity across the edge to the opposite edge. This does not affect the analysis significantly, but is worth noting.

First, we try the Matérn correlation function with $\nu = 1.5$, which gives the estimate of Figure 5.11 with $l = [5 \cdot 10^{-2}\,\text{m}, 2 \cdot 10^{-1}\,\text{m}]$. It is easy to see that this produces an overly smooth estimate, as the random realizations tend to follow the expectation perfectly, with some noise due to the white kernel added.

Next, the Matérn correlation function with $\nu = 1.5$ and $l = [7 \cdot 10^{-3}\,\text{m}, 2 \cdot 10^{-2}\,\text{m}]$ is tested, as shown in Figure 5.12. This image appears overly variable, and the expectation image shows clear signs of the estimates being "pulled" towards the a priori expectation of 2.48 MRayl, indicating that this correlation length is overly short.

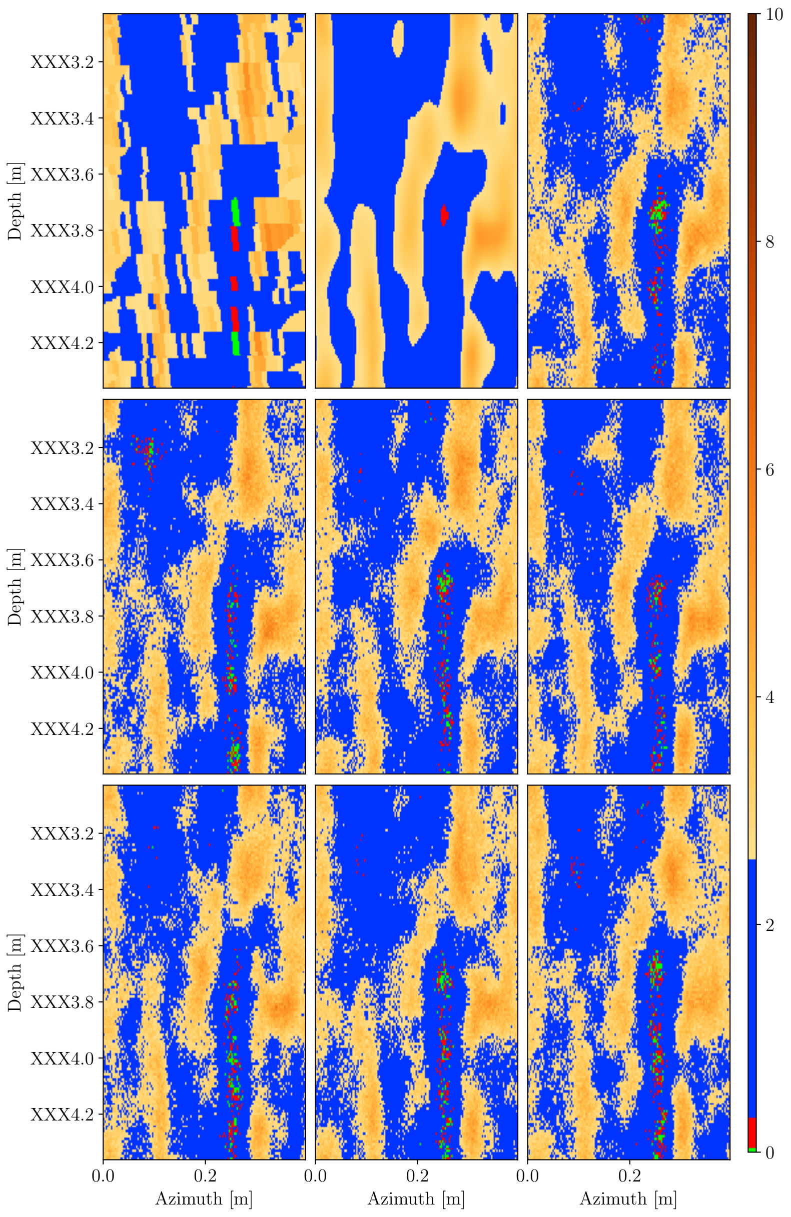

Figure 5.11: Example of kriging using a Matérn kernel with $\nu = 1.5$, and $l = [0.05, 0.2]$m. Top left: Nearest neighbor-interpolated image for comparison. Top middle: Krig expectation image. Remaining images: Random samples.

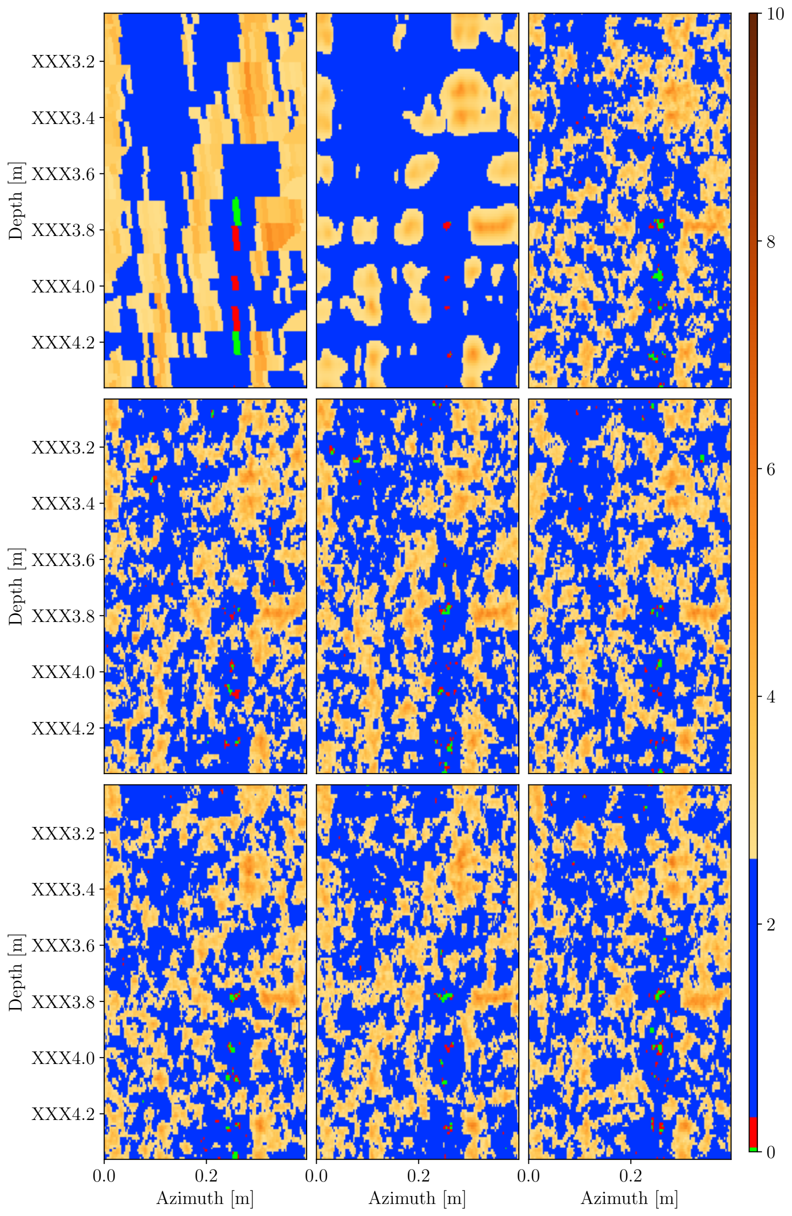

Figure 5.12: Example of kriging using a Matérn kernel with $\nu = 1.5$, and $l = [0.007, 0.02]$m. Top left: Nearest neighbor-interpolated image for comparison. Top middle: Krig expectation image. Remaining images: Random samples.

Further, the Matérn correlation function with $\nu = 1.5$ and $l = [1 \cdot 10^{-2}\,\mathrm{m}, 4 \cdot 10^{-2}\,\mathrm{m}]$ is tested, as shown in Figure 5.13. This image does not contain the artefacts visible in the earlier examples, making it a feasible alternative. To further check the quality of the estimates, we check the depth-wise average acoustic impedance as shown in Figure 5.14. The depth-wise averages are smoothed by a moving average filter of length 0.23m in order to remove edge artefacts from the nearest neighbor image. It is clear by looking at the average impedances that the match is sufficiently good that this estimate is a plausible representation.

The Gaussian correlation function is tested as shown in Figure 5.15. It is clear that the Gaussian correlation results in an overly smooth estimate, as even the random appear smoother than what is probable.

The absolute exponential function, corresponding to a Matérn function with $\nu = 0.5$, is discouraged in [35], as it typically leads to highly variable estimates. To verify this, it is tested and shown in Figure 5.16. While the expectation image looks fairly realistic, with a suitable compromise between smoothness and following the data, it is clear from the random realizations that it still allows for more variability than what is plausible.

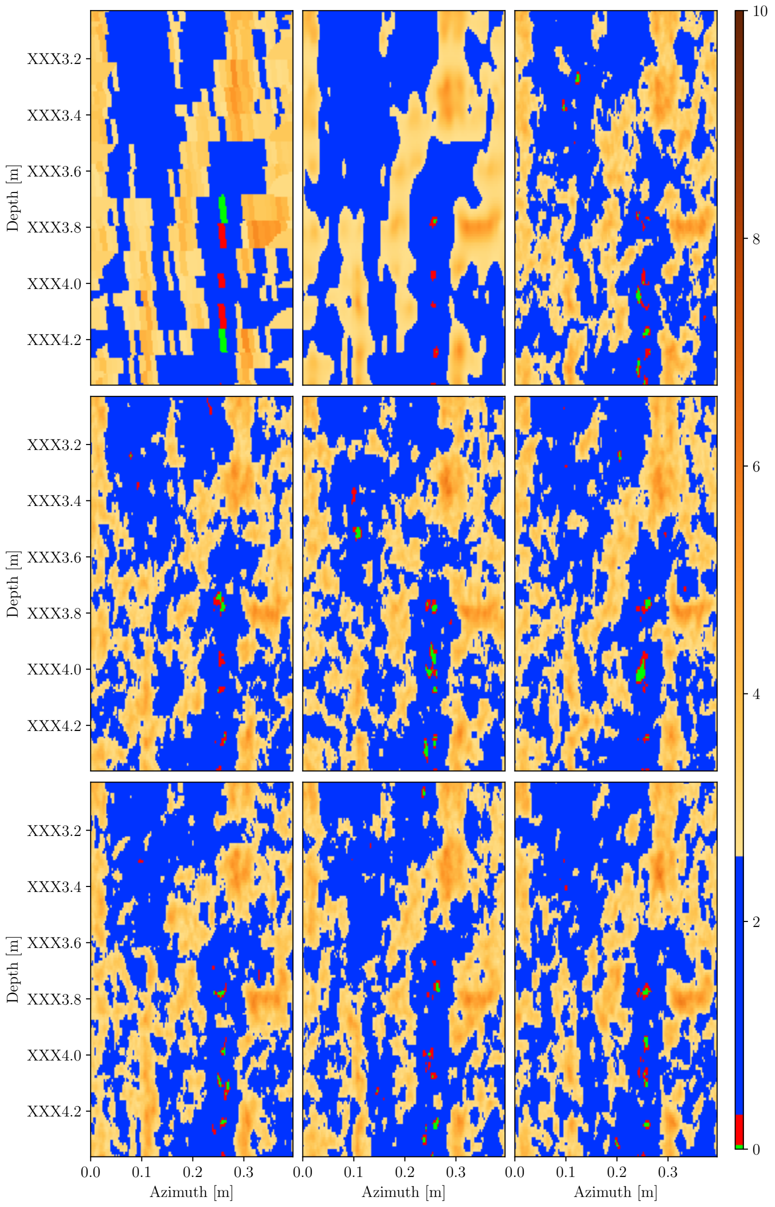

Figure 5.13: Example of kriging using a Matérn kernel with $\nu = 1.5$, and $l = [0.01, 0.04]$m. Top left: Nearest neighbor-interpolated image for comparison. Top middle: Krig expectation image. Remaining images: Random samples.

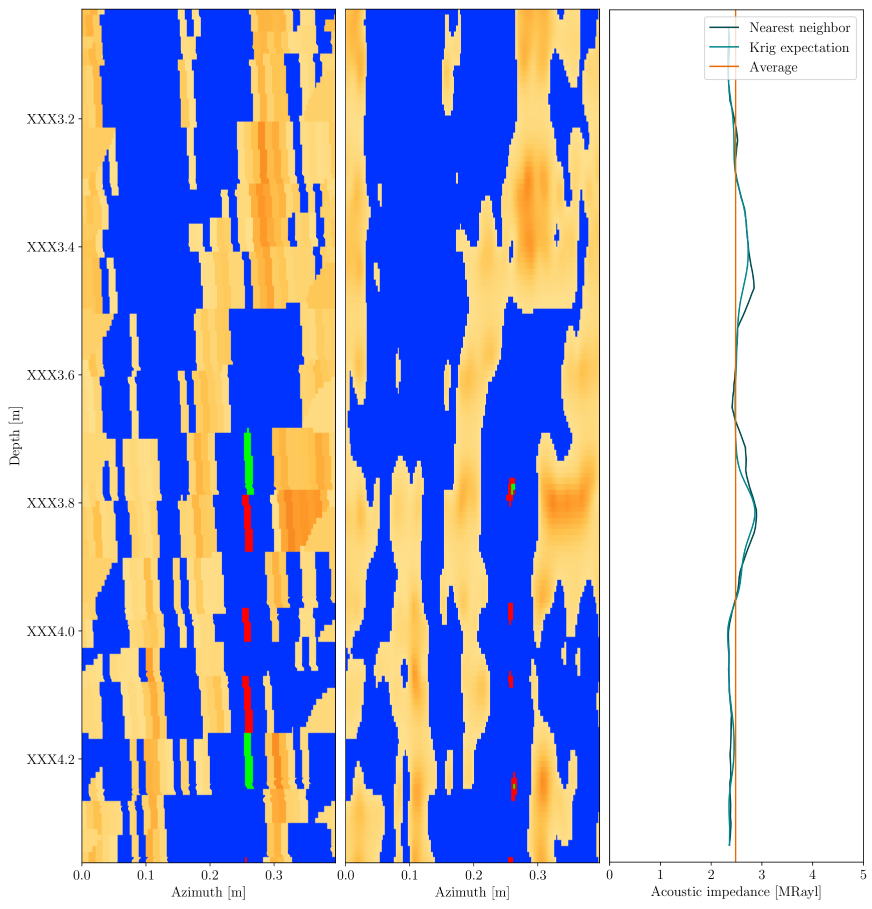

Figure 5.14: Quality control of the krig expectation for Matérn correlation with $\nu = 1.5$ and $l = [0.01, 0.04]$m. The left image shows the nearest neighbor interpolated image, the middle image is the krig expectation, and the right plot shows the depth-wise average of the nearest neighbor image and krig expectation image, as well as the a priori mean μ. The depth-wise averages are smoothed by a moving average filter to remove artefacts

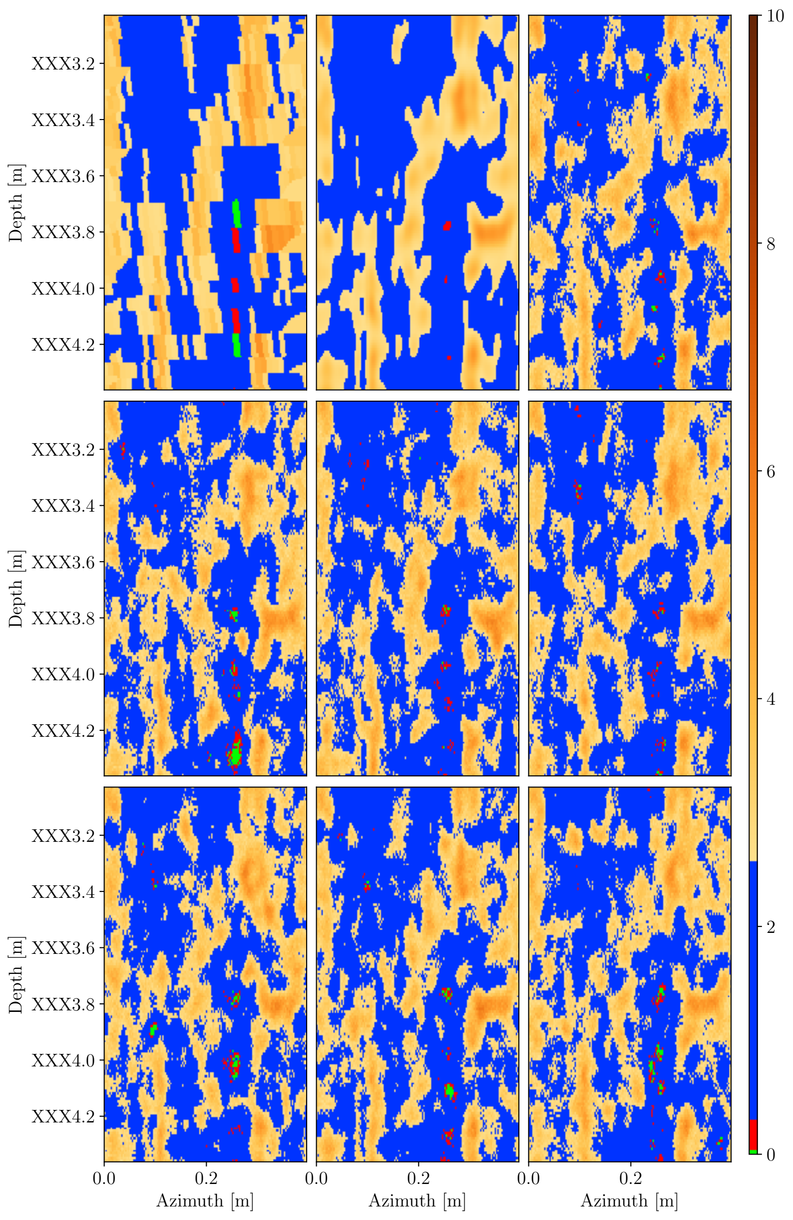

Figure 5.15: Example of kriging using a Gaussian kernel with $l = [0.01, 0.04]$m. Top left: Nearest neighbor-interpolated image for comparison. Top middle: Krig expectation image. Remaining images: Random samples.

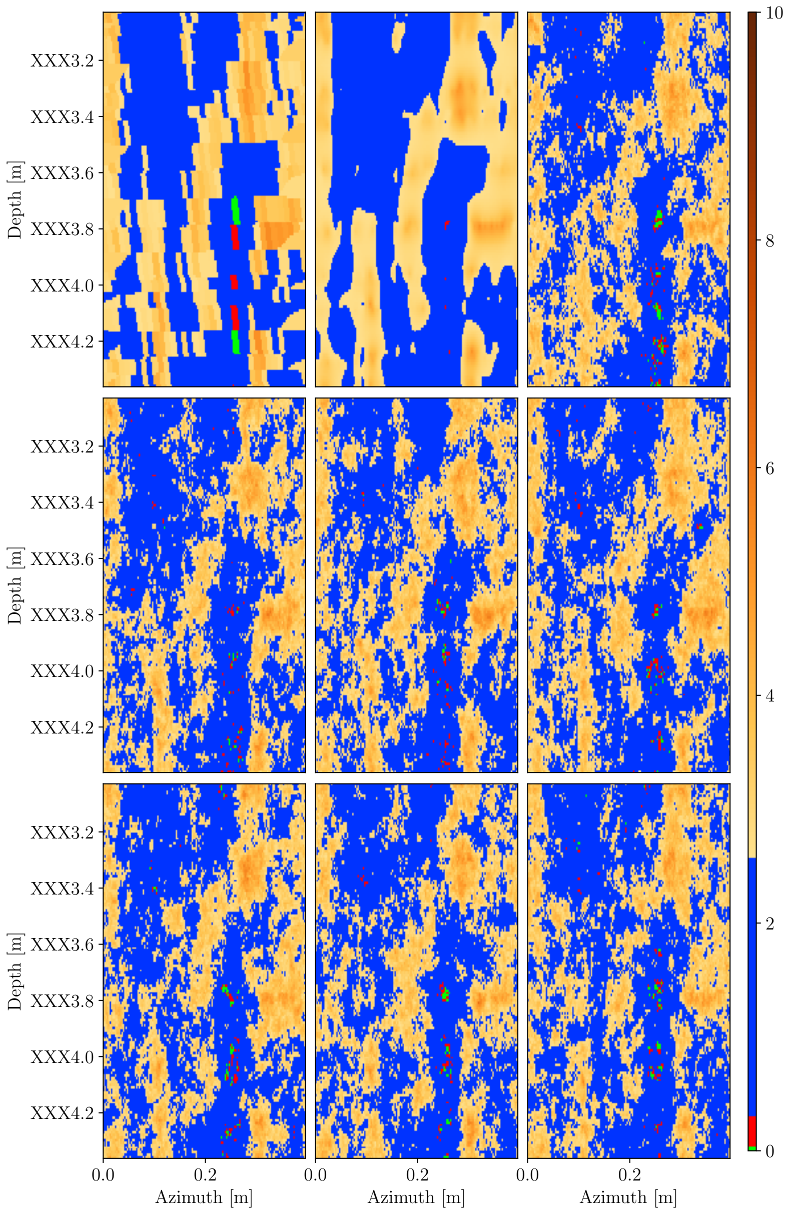

Figure 5.16: Example of kriging using an exponential kernel with $l = [0.01, 0.04]$m. Top left: Nearest neighbor-interpolated image for comparison. Top middle: Krig expectation image. Remaining images: Random samples.

Chapter 6

Deep learning methods for fluid channel detection

This chapter outlines the methodology used for the deep learning part of this thesis. Because this is built on the framework of [3], parts of the methodology is the same. Specifically, this includes how training data are generated, and how the data are augmented for training.

6.1 Image annotation

Using a software package developed by the author in the earlier project, the dataset of ultrasonic well logs provided by Equinor ASA has been annotated by the author to be used for training and testing of deep learning models. While 10 images were annotated for an earlier project [3], all annotations have been subjected to a more thorough quality check for this thesis, in order to ensure consistent evaluations. The images were not resampled using the methods from the previous chapter because of computational complexity. However, the methods developed were used in the quality control stage. One of the reasons for performing quality control on the annotations is that there is a lot of bias in the evaluations of the logs, as pointed out by Viggen et al. [7]. Further, it is important to ensure that this bias does not change over time, which can easily happen when spending weeks performing the same task. Due to time constraints, the size of this dataset is limited to 29 images.

In the previous project, all logs were resampled to a common resolution to make sure that features of the same size would have the same appearance. This was done by nearest neighbor interpolation. However, nearest neighbor interpolation produces overly pixelated images when upsampling. There is a possibility that this may alter features in the log, so that the convolutional kernels' ability to detect features on such images is inhibited. It is well known that convolutional neural networks are able to pick up features at multiple scales, hence, resampling should be unnecessary. Based on this, we performed a test early in the work on this thesis, which showed that the performance of the network was comparable when using resampled images to that when using original resolution images. Because of this, we proceeded with images at their original resolution for the remainder of this thesis.

6.2 Models

Like in [3], an implementation of U-net is used for segmenting the image logs. Parts of this section are therefore similar to the corresponding section in [3]. The implementation of U-net is based on an implementation by Karol Żak [38]. The model is implemented in Keras [39], with the TensorFlow backend [40]. This implementation differs from the version of U-net described in [29] in multiple ways. First, it allows varying the number of levels of the network, so that one can tune the model behavior more freely. Using fewer levels will significantly reduce the number of parameters in the network, with the disadvantage that the classification of any one pixel will receive context from a smaller area. Further, this implementation uses padding before convolution layers that ensures that the classified image is the same size as the input image. The reason for this change from the original architecture is that it ensures that the output image always has the same size, independent of the specific architecture chosen.

In the previous project, zero-padding was used for this. This led to the network struggling to detect channels around the edges of the image. There are multiple ways of solving this problem. One option introduced by [41] is to horizontally stack multiple copies of the image as shown in Figure 6.1, which gives a better impression of the periodic azimuthal direction. However, this is problematic, as it will at least double the memory usage for every image (depending on the number of stacked images). Another option is to make the network periodic in the azimuthal direction. This is done by applying cylindrical padding, meaning that the top and bottom are zero-padded, whereas the sides are padded by data from the opposite side of the image, as shown in Figure 6.2. This is a far more efficient approach, as the memory overhead is minimized. Therefore, this will be the approach used in the U-Net model.

6.3 Model training

To train the models, the previously annotated dataset of ultrasonic well log data provided by Equinor ASA is used. Although the dataset contains a large number of logs, only 29 log images were labelled for this project due to time constraints.

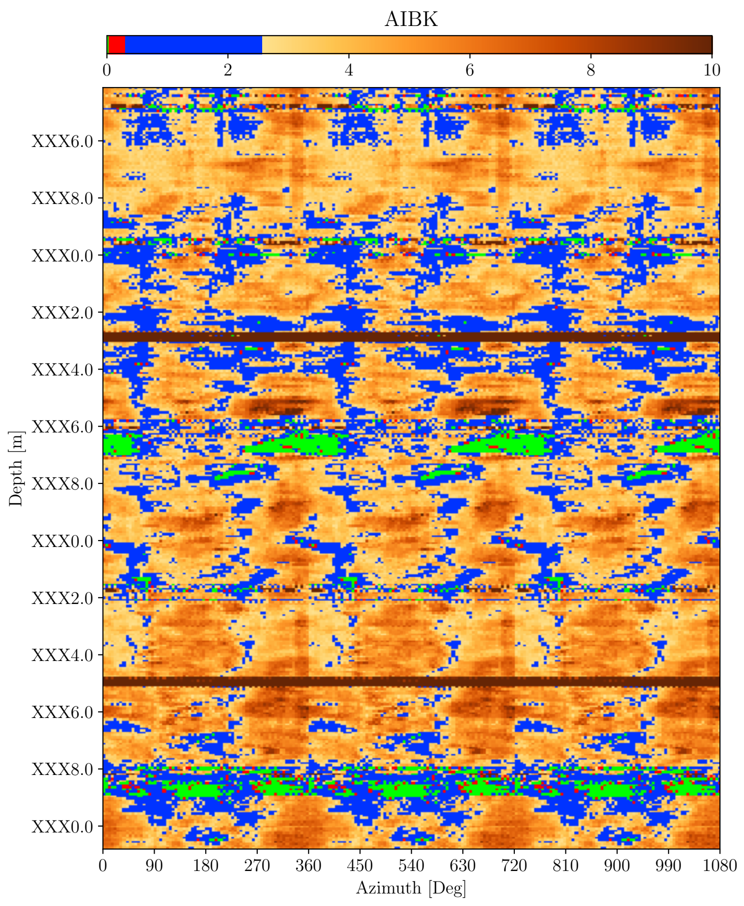

Figure 6.1: An example of how log images may be horizontally stacked to improve the perception of the periodic azimuthal axis. This log segment is the same as in Figure 2.2

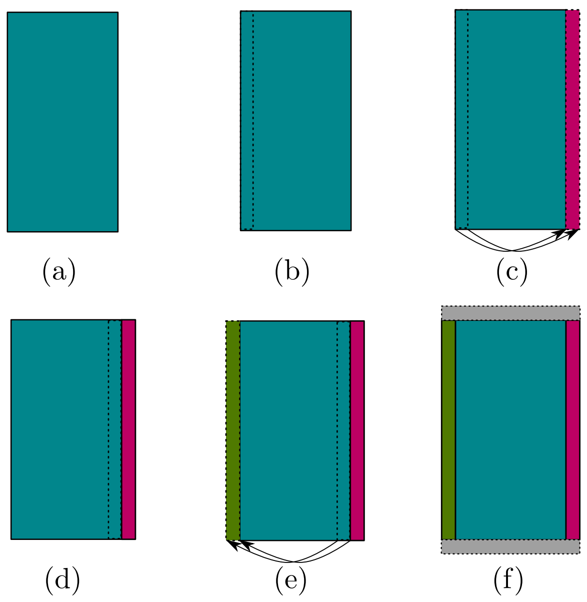

Figure 6.2: An overview of the cylindrical padding process. (a): The original image. (b) and (c): A stripe of the same width as the padding is picked from the left side of the image, flipped, and appended to the right side of the image. (d) and (e): The same process is repeated, taking a segment from what was the right side of the image, flipping, and appending to the left side. (f): Finally, zero padding is applied to the top and bottom of the image.

Because of the limited size of the dataset, transfer learning is tested as it should help teach the network to recognize basic image features. The CAMUS dataset [42] is used for this purpose. This is a dataset of segmented cardiac ultrasonic images. At first glance, these images seem to have very little in common with the image well logs, they were chosen because they only have 1 channel (i.e. they are grayscale images), and because the features in the images are fairly simple, just like the channels in the ultrasonic well logs.

The small number of labelled log images means that the training process is more vulnerable to overfitting, which is the phenomenon of the neural network memorizing training data rather than learning general patterns. This is a common problem in deep learning since the number of parameters in the network is normally much larger than the number of data points. To overcome this problem, semi-supervised learning with pseudo labels is used, as described in section 3.2. This allows using the unlabelled parts of the dataset as training data, by generating so-called ”pseudo-labels”, where the model is used to predict labels for the unlabelled data after each epoch. This allows exposing the network to a wider array of features, and should help improve performance. There are different options for how to use these, ranging from using pseudo-labelled images from the beginning of training, to using it as a fine-tuning method towards the end of training. For this thesis, the latter is chosen. This is due to the difficulties with confirmation bias introduced when using pseudo labels. This is less likely to cause big problems if the model has already been trained to a point where it is performing well. This means that the model is first trained using only labelled data. At a later stage, when the model is starting to get close to peak performance, the training is switched to using a combination of labelled and pseudo-labelled data.

A well known technique to avoid overfitting, is to use data augmentation. Data augmentation is the process of changing the data in ways that don’t affect the label of the data. Common ways of doing this is shearing, rotating and flipping images. For our purposes, we only use flipping, as well as a rotation of the azimuthal reference, that is, we shift where the cylindrical coordinate system is “opened” to retrieve a 2D image. Traditional rotations of these images would not work for obvious reasons (the image is very high and narrow, and a rotation would destroy the periodicity of the image).

Further, because of hardware optimizations, all images in a batch must be the same size. This is achieved by randomly drawing the images for the batch as normal. Then the image with the shortest depth interval is found. All other images in the batch are cut down to this height by drawing a depth interval from the log by a uniform distribution.

Drawing depth intervals from the logs from a uniform distribution introduces a problem into the learning setup: It will have a tendency to create unbalanced training data. Typically, the longest logs will contain long free pipe intervals containing little interesting information, as there will typically be no cement behind the casing, which means that the log simply shows a lot of fluid, with some patches that are typically caused by contact with the formation. It is desirable to create a system that makes it more probable to use data with a higher density of non-zero labels for training the network. Because this classification task is set up with a boolean classification, this is fairly simple. First, compute a per-depth label density by summing each row of data. This produces a label density function of depth, as follows:

$$g(z) = \sum_{\varphi} L(z, \varphi) \tag{6.1}$$

Here, g is the label density per depth index, z is the depth, L is the matrix of labels, and φ is the azimuth. Second, we want the density of labels inside the slice that will be used for training. This is found by summing $g(z)$ over the appropriate z. This is obtained by a convolution between g and a rect function, i.e. a function that is 1 over a given interval, and 0 elsewhere:

$$\mathrm{rect}_{\mathcal{I}}(x) = \begin{cases} 1 & x \in \mathcal{I} \\ 0 & else \end{cases} \tag{6.2}$$

We then obtain the desired probability distribution for each interval starting depth as

$$f(d_{\text{begin}}) = (g * rect_{[-\Delta z, 0]})(z_{\text{begin}}), \tag{6.3}$$

where f is the desired (unscaled) probability distribution, Δz is the length of the depth interval to be sampled, and z_{begin} is the starting depth of the depth interval. Finally, the probability distribution is found by normalizing f

$$p(z_{\text{begin}}) = \frac{f(z_{\text{begin}})}{\sum_x f(x)} \tag{6.4}$$

This allows sampling depth intervals with probability density f, which will ensure that there is a significant presence of features in the data that are provided for training.

This solution solves the problem of sampling large intervals with no labels. However, this introduces the problem that such intervals will never be sampled, which is not ideal. To solve this, one may add a small value to every element in the unscaled probability density function, to ensure that every depth interval has some small probability of being sampled, like so:

$$f_{\text{aug}}(z_{\text{begin}}) = f(z_{\text{begin}}) + b,$$

where b is the baseline value. To obtain the probability distribution, (6.4) is applied to f_{aug}.

Towards the end of the training, semi-supervised learning with pseudo-labels [17] is used to improve performance further. This is performed as follows: For each epoch, the model is used to predict an unlabelled dataset. The predictions are then rounded and used as pseudo-labels for the unlabelled data. Each data batch fed to the training algorithm is set up as a mixture of labelled data and pseudo-labelled data.

6.4 Model selection

Given the general U-net architecture in Figure 4.8, there is a significant opportunity for modification of the architecture. Specifically, we vary the number of filters (feature maps). In a previous work [3], the number of levels of the U-net was varied. However, for this thesis, all models use two levels. There are two reasons for this:

1. In [3], all images were captured over 72 azimuths. This meant that the triple pooling used in such an architecture was possible. For this thesis however, the images are kept at their original resolution, which means that the network needs to be able to accept images with 36 azimuths. This is because, as explained in Section 4.4, 2 levels will reduce the images azimuth dimension to 9, which is not divisible by 2.

2. The findings of [3] indicated that the number of filters had a much bigger impact on the performance of the network than the number of levels.

In order to compare the models, their performance must be estimated. In [3], bootstrapping was used for this purpose. However, this turned out to give a lot of samples with very poor performance, meaning that the final estimates of model performance were very low compared to the expected real world performance. To circumvent this problem, the bootstrapping is replaced by crossvalidation. Here, the data for each sample are selected as shown in Figure 3.1.

The training data is used to train the model, whereas the test set is used to compute an evaluation metric on the trained model. This procedure is repeated a number of times, and the performance metric of the model architecture is estimated as the average of the performance metric value over all bootstrap samples. This allows for comparing the performance of different model architectures.

Further, to avoid overfitting, validation data checkpoints are employed when training the model. This means that a few data points are removed from the training set in order to constitute a validation set. For each training epoch, a performance metric is computed on the validation data. We store model weights that resulted in the best validation performance metric rather than the weights from the final training step. This is done due to the fact that when the model starts overfitting, the validation metrics will begin to decline. The checkpoints will therefore ensure that one retains model parameters that provide good performance on data that are not present in the training set.

For final testing of the chosen model from the model selection step, the model is trained using all the data from the crossvalidation. Further, it is tested on the test set, and the model is evaluated using the IoU and Dice coefficients, as well as an ROC curve. An ROC curve is constructed by varying the classification threshold for a classifier between 0 and 1, and recording the true positive and false positive rates. A good classifier will be able to achieve both a high true positive rate and a low false positive rate, and so, the curve will tend to first go up steeply almost to $TPR = 1$, and then slowly approach $TPR = 1$ while the false positive rate increases. In order to easily quantify the performance of a classifier based on an ROC curve, the area under curve metric (AUC) is often used. A higher AUC will mean that the classifier tends to have a high true positive rate and low true negative rate simultaneously. A commonly used baseline for ROC curves is the "chance" curve, i.e. a straight line. However in this case, a better baseline is to use a simple classifier based on raw AIBK data. In this case, we clip the AIBK values to the interval $[0, 10]$MRayl, and then normalize to the interval $[0, 1]$ by dividing all values by 10. We denote this by $\text{AIBK}_{\text{norm}}$. Since lower impedance values indicate fluid behind the casing, a basic approach is to let the channel classifier be one minus the normalized acoustic impedance value,

$$S = \mathbf{1} - \text{AIBK}_{\text{norm}}, \tag{6.5}$$

where $\mathbf{1}$ is a matrix of 1's of the same dimensions as AIBK. Classifications are obtained by applying a threshold to S.

Chapter 7

Channel detection results

This chapter will summarize the results from the deep learning part of this thesis.

7.1 Testing training methods

First, we discuss the different training methods, namely transfer learning and and semi-supervised learning. The IoU obtained with each method is estimated by 10-fold cross-validation. The models are first trained with a supervised setup, either initialized from a transfer learning model, or from a random initialization [27]. After the supervised training stage the models are tested, before a semi-supervised training stage is run. Finally, the models are tested again. This yields 4 distinct training setups. The model used for this test is the model that achieved the best results in [3], namely a U-net with 24 filters. However, as mentioned earlier, the current model is limited to 2 layers, and uses the cylindrical padding developed for this thesis. The performance distribution over the folds is shown in Figure 7.1, while the results of this are summarized in Table 7.1. Figure 7.2 shows the performance on every fold.

It is clear from Table 7.1 alone that semi-supervised learning provides a significant benefit. To confirm this, a one-sided Wilcoxon test is performed. With H_0: Original setup $>$ Semi-supervised, $p = 0.048$, indicating that H_0 should be rejected, so the semi-supervised setup is indeed significantly better.

Table 7.1: Crossvalidation performance for the different training setups tested.

	No transfer learning	Transfer learning
Supervised learning	0.23 ± 0.09	0.30 ± 0.15
Semi-supervised learning	0.31 ± 0.14	0.29 ± 0.13

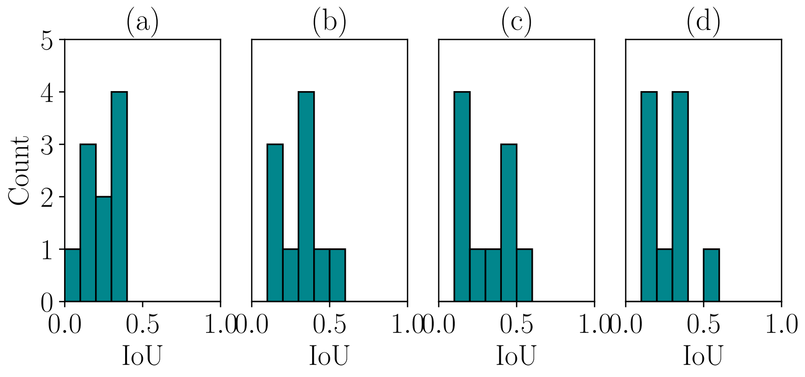

Figure 7.1: Histogram of IoUs from the training testing. (a) Original setup. (b) Semi-supervised learning. (c) Transfer learning. (d) Transfer learning and semi-supervised learning.

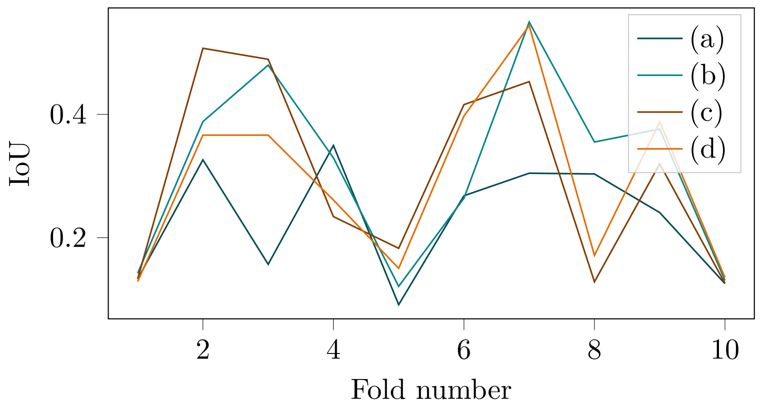

Figure 7.2: Performance per crossvalidation fold for each of the tested training methods. (a), (b), (c), and (d) represent the same training methods as in Figure 7.1.

Table 7.2: Crossvalidation performance for each number of filters tested for the U-Net model.

Filters	IoU
1	0.08 ± 0.07
2	0.09 ± 0.06
4	0.17 ± 0.08
8	0.24 ± 0.11
12	0.29 ± 0.15
18	0.32 ± 0.16
24	0.34 ± 0.16
30	0.32 ± 0.15

7.2 Model selection

In order to assess whether it may improve performance, the Swish activation function is tested using crossvalidation. This test is performed using 6 models of 18, 24, and 30 filters, with ReLU and Swish. Testing the significance of the ReLU's performance relative to that of Swish (H_0: $IoU_{\text{ReLU}} < IoU_{\text{Swish}}$), gives p=0.0001, meaning that H_0 is rejected, i.e. that Swish does not provide any improvement over the ReLU.

Based on the tests of the training methods, 8 models are tested using the combination of supervised learning first, and then semi-supervised learning. All models use 2 levels, and ReLU activation. Thus, the only variable is the number of filters. The models and their scores are presented in Table 7.2

Further, a graph of the model performance is shown in Figure 7.4.

Performing Wilcoxon-tests over all pairs yields the matrix of p-values

$$P = \begin{array}{c|cccccccc} i\backslash j & 1 & 2 & 4 & 8 & 12 & 18 & 24 & 30 \\ \hline 1 & & 0.86 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \\ 2 & 0.16 & & 0.99 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \\ 4 & 9 \cdot 10^{-4} & 0.01 & & 0.99 & 1.0 & 1.0 & 1.0 & 1.0 \\ 8 & 2 \cdot 10^{-3} & 6 \cdot 10^{-3} & 0.01 & & 0.95 & 0.99 & 1.0 & 1.0 \\ 12 & 2 \cdot 10^{-3} & 2 \cdot 10^{-3} & 3 \cdot 10^{-3} & 0.07 & & 0.92 & 1.0 & 0.90 \\ 18 & 9 \cdot 10^{-4} & 9 \cdot 10^{-4} & 9 \cdot 10^{-4} & 0.01 & 0.10 & & 0.75 & 0.28 \\ 24 & 2 \cdot 10^{-3} & 9 \cdot 10^{-4} & 2 \cdot 10^{-3} & 9 \cdot 10^{-4} & 7 \cdot 10^{-3} & 0.28 & & 0.10 \\ 301 & 2 \cdot 10^{-3} & 2 \cdot 10^{-3} & 2 \cdot 10^{-3} & 3 \cdot 10^{-3} & 0.11 & 0.75 & 0.92 & \end{array},$$

Where $p_{i,j}$ is the p-value for the test with H_0 : Model with j filters $>$ Model with i filters(The first index corresponds to the row, and the second to the column). Here, a low p-value indicates that a model with i filters is as good, or better than a model with j filters. For example, $p_{24,1} = 2 \cdot 10^{-3}$, indicating that 24 filters is significantly better than 1 filter. Note

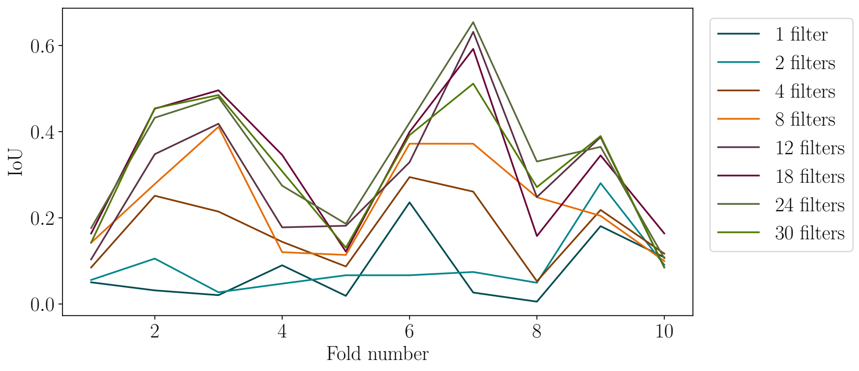

Figure 7.3: Performance per crossvalidation fold for each of the models tested, corresponding to Table 7.2.

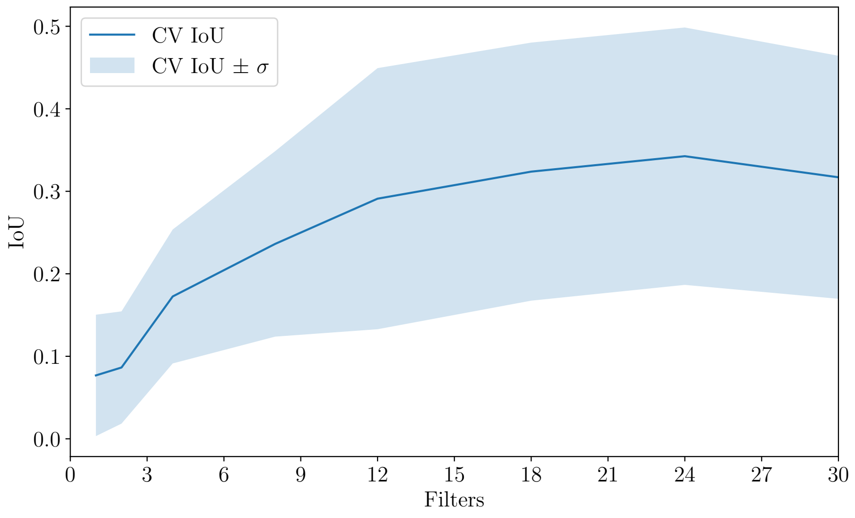

Figure 7.4: Graph of the performance as a function of the number of filters in the U-Net model, with uncertainty.

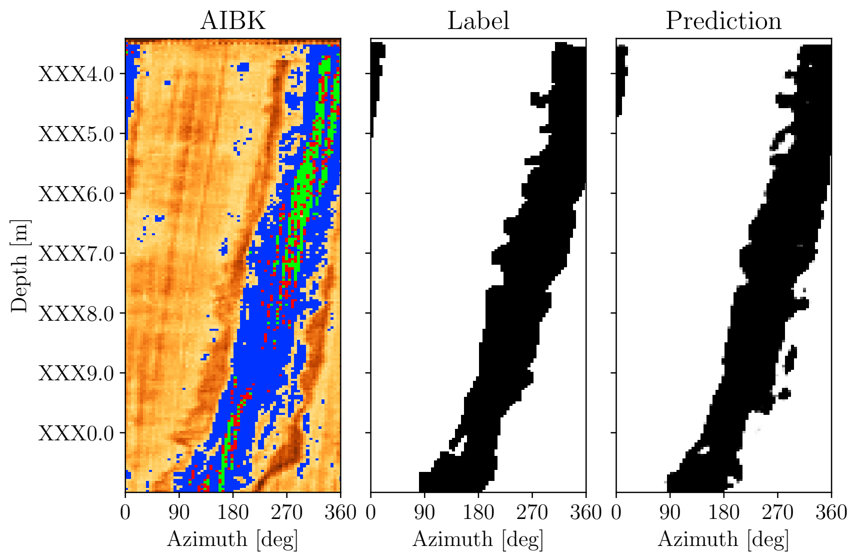

Figure 7.5: A test of the final U-Net model with 24 filters. This log is from the Volve data village [43].

Table 7.3: Performance overview for the final U-Net model.

	Value
True positive rate	0.63
False positive rate	0.05
False negative rate	0.37
True negative rate	0.95

that in many cases $p_{ij} \neq 1 - p_{ji}$. This is unexpected, but is likely due to a limited data table used in the Wilcoxon test function, which causes limited resolution in the p-values.

7.3 Final testing

Using the best performing model from the model selection, the model using 24 filters, a final model is trained for testing. Example results are shown in Figures 7.5 and 7.6. The performance of this model on the test set is summarized in Table 7.3.

Finally, by moving the threshold for the predictions, an ROC curve is produced, as shown in Figure 7.7

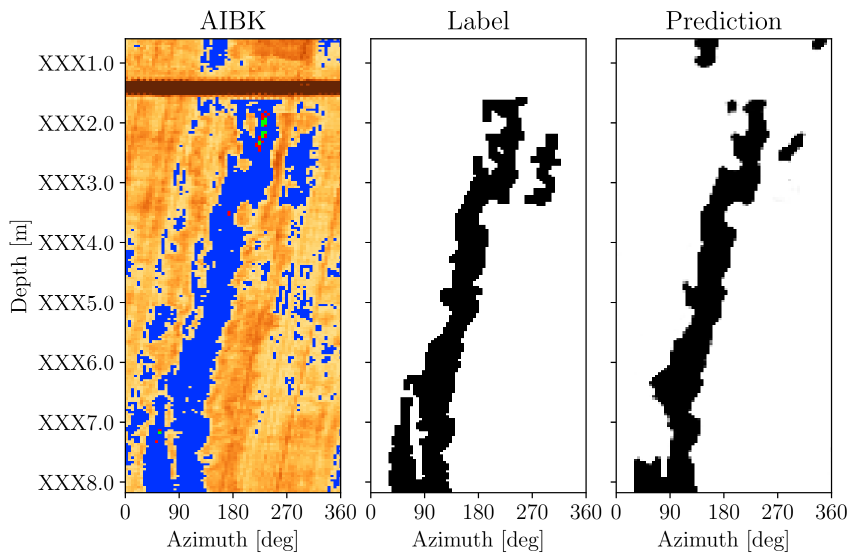

Figure 7.6: A test of the final U-Net model with 24 filters. This log is from the Volve data village [43].

Table 7.4: Metrics for the final U-Net model.

Metric	Value
IoU	0.44
Dice	0.61

Table 7.5: AUC metrics based on Figure 7.7.

Model	AUC
U-Net	0.882
$\mathbf{1} - \text{AIBK}_{\text{norm}}$	0.846

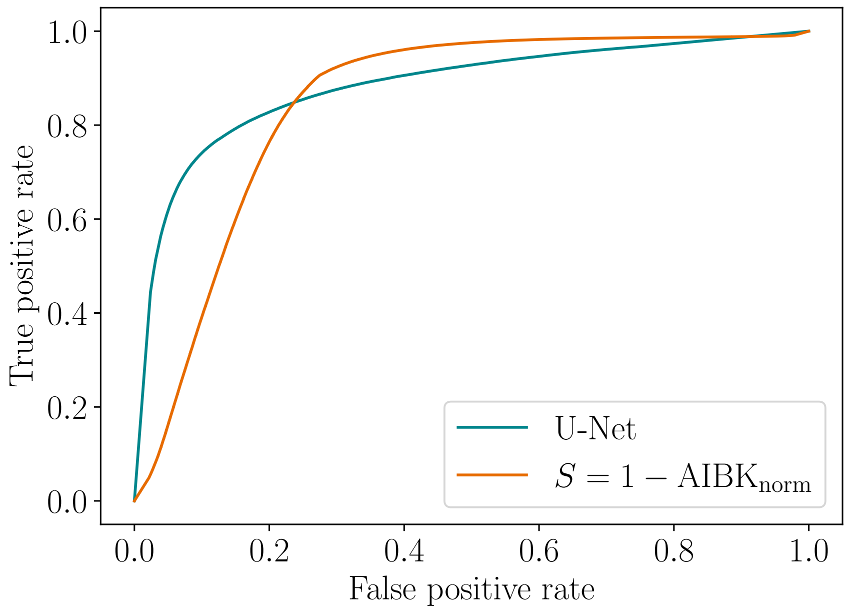

Figure 7.7: ROC curve for the final U-Net model, as well as a simple model used for comparison.

Chapter 8

Discussion

This chapter will discuss the findings in Chapters 5 and 7.

8.1 Well log imaging

As mentioned previously, the analysis of the measurement locations is inhibited by the lack of accelerometer data. However, it is still clear from the analysis used here, that analyzing the measurement locations explains artefacts in the images normally displayed in ultrasonic well logs.

The nearest neighbor image provides a simple way to paint a better picture of where the measurements are made. For this thesis, however, the main use of this image is as a comparison for the kriging images. This is because the nearest neigbor image retains the general appearance of the raw image display, while implementing the more accurate information about where the measurements are collected from.

The kriging produces realistic-looking estimates, although it is challenging to conclude on which estimate is the best, due to the fact that the ground truth is unknown. This also means that even if a set of good parameters for the covariance are found for one log, they will not necessarily be equally good for a different log, meaning that parameter testing will be necessary for every new log. It is however clear that the combination of the expectation image and the random realizations works as a valuable tool to better understand the possible variation given the measurements. Further, it enables far better analysis in cases where it is difficult to determine what is behind the casing from the raw impedance images alone.

The most prominent limitation of the approach is the computational expense. For this thesis, the longest logging interval used for kriging was around 1 meter. This means that it is completely infeasible to make evaluations directly on images produced by kriging. One of the reasons for this is that the kriging library used does not leverage the sparsity of the covariance matrix for this application. It is obvious that the covariance function around a given point will have compact support, that is, each point will only correlate with a limited number of points, which are the points in the immediate neighborhood. This will in turn lead to a sparse covariance matrix, i.e. a matrix with a large fraction of its entries being 0. A library with specific support for this would be able to save a lot of computational resources, which would enable kriging of far longer log intervals. This will further make the random sampling more efficient, as logging intervals far away from each other can be estimated and sampled independently of each other.

Another limitation of this analysis is that a constant mean is used. While this is not critical for such short intervals analyzed here, the mean should ideally be estimated locally, to avoid biasing the gaussian process as seen in Figure 5.14.

8.2 Deep learning

The high standard deviation of the estimates in Table 7.1 does at first look problematic for comparing both different training methods, and for comparing models later. However, as shown by Figure 7.2, the performance of the different training setups is correlated, meaning that most of the variation is caused by some folds being more challenging to learn than others. The Wilcoxon test used for comparing different approaches compares performance on a per-fold basis, meaning that the variation in difficulty across folds is eliminated in the statistical testing.

As seen from Table 7.1 training with either semi-supervised learning, transfer learning, or both, is significantly better than not using any of the techniques. However, there is not a significant difference between these 3 methods, with all p values from Wilcoxon tests being larger than 0.2.

At the start of this thesis, we hypothesised that the transfer learning would introduce a large increase in performance. The results found here are weaker than expected. This may be due to the fact that the medical ultrasound images show impedance contrasts. Because of this, detecting features in such images is based on detecting edges and ridges, and then creating regions using these edges and ridges. Conversely, the well log data have absolute impedance values, meaning that here, the model is looking for features with a certain appearance, and within a given absolute impedance range.

An alternative approach to transfer learning using an unrelated dataset could be to generate synthetic data. One could do this by using an algorithm to first define the output image. This image could be built with a combination of patches and channels, where the channels could be generated by a random height, a random width (which can vary randomly across different depths), and a wandering azimuthal position. From there, one could simply translate this to probable impedance values, add casing collar signatures, and perhaps some noise on top. As this is a fairly simple process, this seems more feasible than transfer learning on images from a different domain.

Further, the results indicate that semi-supervised learning works well. As has been discussed earlier, confirmation bias may present a problem in setups using pseudo-labelling. This is an interesting point for further investigation. One could also experiment with different thresholds for the pseudo-labels, to augment the models performance. This step was considered for this thesis, but was omitted due to time constraints.

Given that the performance with only transfer learning and only semi-supervised learning is similar, a choice must be made on which one to proceed with. As mentioned earlier, the dataset used for transfer learning may not be ideal for this application. By looking at Figure 7.2, we see that semi-supervised learning achieves the highest peak performance in addition to having the highest estimated performance by the crossvalidation shown in Table 7.1. Further, given the access to a large number of unlabelled logs, this is a simple option to improve performance. Based on this, semi-supervised learning has been chosen as the training method for the remaining tests.

This means that for model testing, the models are first trained using supervised learning, and then trained further with semi-supervised learning, like in the training method tests. The results are shown in Table 7.2. As with the training method testing, the standard deviations are fairly high. However, as Figure 7.3 shows, most of this variation is still related to how challenging the folds are to learn. As seen in Figure 7.4, the performance peaks around 24 filters. From examining P in (7.2), the models with 18, 24, and 30 filters are not significantly different. However the figure still indicates that 24 filters is the best choice for a final model.

Final testing produces the results shown in Table 7.3. It is clear that the model still has problems detecting all channels like in [3], however the true positive rate is higher (63% vs 45%). Further, by looking at Figure 7.5, we see that the addition of the cylindrical padding has been successful in the sense that the model is able to detect channels "wrapping" around the edge of the image. This solves the problem observed in the project [3], where the model would not detect channels that passed the edge of the image, appearing on the oppisite side. Table 7.4 shows that the final model has an IoU of 44%. This is higher than the estimated performance of Table 7.2, however it is worth noting first that this model is trained with all the data used for the crossvalidation, and that the peak performance of this model in the crossvalidation testing is higher than 44%, as indicated by Figure 7.3.

The ROC curve in Figure 7.7 shows far better than chance performance. The ROC curve provides more useful information about the performance of the classifier than metrics based on a 0.5 threshold. This is because an evaluator should be able to decide the threshold based on risk tolerance. In a risk-averse context, the user should be able to set a low threshold to minimize the number of false negative findings.

From Table 7.5, we see that the AUC for the U-Net model is slightly better than the basic model S given in (6.5). This small difference is likely caused by the basic model detecting free-pipe intervals as channels. This means that if this method was slightly more refined it might out-perform the U-net model. This refinement should probably include both some method for detecting free-pipe intervals, which can be done using a threshold on the average impedance per depth, as well as some clustering method, that ignores patches below a certain size. Such an approach would leave only a few trainable parameters, which should be far easier to learn than the U-Net's $\sim$260000 parameters.

As mentioned previously, well log evaluations are highly subjective. This means that the edges of features will tend to be highly variable depending on the evaluator, and sometimes the appearance of the feature. However, it is more critical that the feature is detected, than exactly where the edges are, within reason. This means that a performance metric weighting the interior of a feature more than the exterior could provide a performance benefit over the IoU and Dice coefficients used for this thesis.

The addition of cylindric padding means that the rotations used for data augmentation previously is now redundant. However, one may use an azimuthal rotation of the image that varies slowly in depth in cylindrical coordinates (shearing in the flat image). This would conserve continuity of any features, while distorting the image sufficiently to provide the machine learning with different features than those present in the raw data. For this, one could for instance generate a depth-wise rotation using a random walk (brownian motion). This has not been done in this project due to time constraints, but may be an interesting development for future work.

For this thesis, the deep learning model only received acoustic impedance images as input. As explained in Chapter 2, well log evaluations are based on multiple data channels. As such, it is likely that performance could be improved by including more data channels, such as thickness images.

8.3 Future work

A major problem with the kriging is that the ground truth is not known, making it difficult to assess which parameters produce the best estimate. Therefore, an interesting exercise would be to test kriging in a case where the ground truth is known. A partial solution to this would be to downsample a high-resolution log, and then use kriging to upsample the log to its original resolution. This could help shed more light on how to do kriging of AIBK images. It would also be interesting to test data-driven approaches to finding the correlation function, and comparing this to the correlation functions used in this thesis.

As discussed previously, it is still likely that the machine learning approach to detecting channels could be improved by improving the volume of labelled data even more. However, as indicated by Figure 7.7, this model only barely outperforms the naive model based on input data alone. This indicates that perhaps the machine learning is an overly complex approach to the problem, and that the problem should rather be approached with simpler, classical image analysis methods, as described previously. It is however worth noting that the annotations made for this thesis are still useful for training and testing a classical image analysis method, meaning that a large portion of the work done for this thesis is still relevant even if deep learning does not seem like a good tool to solve the feature detection problem posed in this thesis.

Chapter 9

Conclusion

This thesis utilized deep learning for detecting channels in the casing cement of oil and gas wells. Alongside this task, a detailed analysis of the ultrasonic measurements was made, focusing specifically on how to reconstruct images from the measurements using kriging. This approach yielded a deeper insight into how to evaluate the logs, and has potential for being a useful tool in future work on well integrity evaluations, as well as for further research on feature detection in such well logs.

This thesis implemented several improvements over previous work on this problem. While the performance was improved over previous efforts, the results also show that a far simpler approach to the problem yields competitive results. It is likely that an even larger volume of data would help the deep learning approach, however given the volume of data available now, it is not certain that a further increase in data volume will be enough to make the performance sufficient for operational use.

At this point, the deep learning approach to this problem seems like an overly complex solution given the results presented here. Therefore, further work on this problem will likely be more successful with an emphasis on classical methods.

Bibliography

[1] S. H. Bittleston, J. Ferguson, and I. A. Frigaard. “Mud removal and cement placement during primary cementing of an oil well”. In: *Journal of Engineering Mathematics* 43.2-4 (2002), pp. 229–253. ISSN: 00220833. DOI: 10.1023/A:1020370417367.

[2] Ragnhild Skorpa and Torbjørn Vrålstad. “Visualization of fluid flow through cracks and microannuli in cement sheaths”. In: *SPE Journal*. Vol. 23. 4. Society of Petroleum Engineers, 2018, pp. 1067–1074. DOI: 10.2118/180019-pa.

[3] Simon Andreas Hoff. “Feature detection in petroleum well logs”. 2020.

[4] M Van Ginkel et al. *Robust Curve Detection using a Radon Transform in Orientation Space Applied to Fracture Detection in Borehole Images*. Tech. rep. 2003, pp. 299–306. URL: https://d1rkab7tlqy5f1.cloudfront.net/TNW/Over%20faculteit/Decaan/Publications/2001/ASCI2001MGLVPV.pdf.

[5] K. Glossop et al. “An Implementation of the Hough Transformation for the Identification and Labelling of Fixed Period Sinusoidal Curves”. In: *Computer Vision and Image Understanding* (2002). ISSN: 10773142. DOI: 10.1006/cviu.1999.0747.

[6] Yinyu Wang et al. *Method of determining planar events from borehole or core images*. 2007. URL: https://patents.google.com/patent/US7236887B2/en.

[7] Erlend Magnus Viggen et al. “Automatic interpretation of cement evaluation logs from cased boreholes using supervised deep neural networks”. In: *Journal of Petroleum Science and Engineering* (2020). in press.

[8] E.B. Nelson and D. Guillot. *Well Cementing*. Developments in petroleum science. Schlumberger, 2006. ISBN: 9780978853006. URL: https://books.google.no/books?id=FblqPAAACAAJ.

[9] S. Ştefănescu in collaboration with C. and M. Schlumberger. “Sur la distribution électrique potentielle autour d’une prise de terre ponctuelle dans un terrain à couches horizontales, homogènes et isotropes”. In: *Le journal de physique et le radium, Série 7, Tome 1* (1930), pp. 132–140.

[10] M. Grosmangin, P.P. Kokesh, and P. Majani. “A Sonic Method for Analyzing the Quality of Cementation of Borehole Casings”. In: *Journal of Petroleum Technology* 13.02 (Feb. 1961), pp. 165–171. ISSN: 0149-2136. DOI: `10.2118/1512-g-pa`.

[11] Warren L. Anderson and Terry Walker. “Research Predicts Improved Cement Bond Evaluations With Acoustic Logs”. In: *Journal of Petroleum Technology* 13.11 (Nov. 1961), pp. 1093–1097. ISSN: 0149-2136. DOI: `10.2118/196-pa`.

[12] G.H. Pardue and R.L. Morris. “Cement Bond Log-A Study of Cement and Casing Variables”. In: *Journal of Petroleum Technology* 15.05 (May 1963), pp. 545–555. ISSN: 0149-2136. DOI: `10.2118/453-pa`.

[13] Ioan Alexandru Merciu. Personal communication. Equinor ASA. 2019.

[14] *NORSOK STANDARD D-010 Well integrity in drilling and well operations.* Tech. rep. URL: `www.standard.no/petroleum`.

[15] A.J. Hayman, R. Hutin, and P.V. Wright. “High-Resolution Cementation And Corrosion Imaging By Ultrasound”. In: *SPWLA 32nd Annual Logging Symposium* (1991).

[16] R. Van Kujik et al. “A novel ultrasonic cased-hole imagerfor enhanced cement evaluation”. In: *2005 International Petroleum Technology Conference Proceedings.* 2005, pp. 855–868. DOI: `10.2523/iptc-10546-ms`.

[17] Dong-Hyun Lee. *Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks.* Tech. rep.

[18] Eric Arazo et al. *Pseudo-Labeling and Confirmation Bias in Deep Semi-Supervised Learning.* Tech. rep. arXiv: `1908 . 02983v5`. URL: `https://git.io/fjQsC.`.

[19] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from data : a short course.* AMLBook.com, 2012. ISBN: 9781600490064.

[20] Prajit Ramachandran, Barret Zoph, and Quoc V Le Google Brain. *SEARCHING FOR ACTIVATION FUNCTIONS.* Tech. rep. arXiv: `1710.05941v2`.

[21] *File:Colored neural network.svg - Wikipedia.* URL: `https://en.wikipedia.org/wiki/File:Colored%7B%5C_%7Dneural%7B%5C_%7Dnetwork.svg` (visited on 01/06/2020).

[22] Michael A. Nielsen. *Neural Networks and Deep Learning*. 2015. URL: `http://neuralnetworksanddeeplearning.com`.

[23] Ilya Kuzovkin et al. "Activations of deep convolutional neural networks are aligned with gamma band activity of human visual cortex". In: *Communications Biology* 1.1 (Dec. 2018), pp. 1–12. ISSN: 23993642. DOI: `10.1038/s42003-018-0110-y`. URL: `https://www.nature.com/articles/s42003-018-0110-y`.

[24] Grace W Lindsay. *Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future*. Tech. rep.

[25] Francois Chollet. *Deep Learning with Python*. Vol. 80. Manning Publications Co., 2018, p. 453. ISBN: 9781617294433. eprint: `1-933988-16-9`.

[26] Gareth James et al. *An Introduction to Statistical Learning*. Vol. 103. Springer Texts in Statistics. New York, NY: Springer New York, 2013. ISBN: 978-1-4614-7137-0. DOI: `10.1007/978-1-4614-7138-7`. URL: `http://link.springer.com/10.1007/978-1-4614-7138-7`.

[27] Kaiming He et al. *Deep Residual Learning for Image Recognition*. Tech. rep. arXiv: `1512.03385v1`. URL: `http://image-net.org/challenges/LSVRC/2015/`.

[28] Dan C Ciresan et al. *Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images*. Tech. rep. 2012. URL: `http://www.idsia.ch/`.

[29] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9351. Springer Verlag, 2015, pp. 234–241. ISBN: 9783319245737. DOI: `10.1007/978-3-319-24574-4_28`. arXiv: `1505.04597`.

[30] *Recommended Digital Log Interchange Standard (DLIS), Version 1.00: API Recommended Practice 66 (RP66)*. Tech. rep. American Petroleum Insitute, 1991. URL: `http://www.posc.org/technical/data%7B%5C_%7Dexchange/RP66/V1/rp66v1.html`.

[31] *Welcome to dlisio's documentation! — dlisio 0.1 documentation*. URL: `https://dlisio.readthedocs.io/en/stable/` (visited on 08/30/2019).

[32] *What is HDF5?* URL: `https://support.hdfgroup.org/HDF5/whatishdf5.html` (visited on 01/08/2020).

[33] Erlend Magnus Viggen. Personal communication. NTNU/CIUS. 2019.

[34] Erlend Magnus Viggen, Erlend Hårstad, and Jørgen Kvalsvik. "Getting started with acoustic well log data using the dlisio Python library on the Volve Data Village dataset". In: *Proceedings of the 43rd Scandinavian Symposium on Physical Acoustics*. Ed. by Erlend Magnus Viggen and Lars Hoff. Geilo, Norway: Norwegian Physical Society, Apr. 15, 2020, p. 36. ISBN: 978-82-8123-020-0. URL: `https://www.researchgate.net/publication/340645995_Getting_started_with_acoustic_well_log_data_using_the_dlisio_Python_library_on_the_Volve_Data_Village_dataset,%20Full-text%20on%20ResearchGate%20https://github.com/equinor/dlisio-notebooks/blob/master/acoustic.ipynb,%20Companion%20Jupyter%20Notebook`. published.

[35] C E Rasmussen and C K I Williams. *Gaussian Processes for Machine Learning*. MIT press. ISBN: 9780262182539. URL: `www.GaussianProcess.org/gpml`.

[36] Albert Tarantola. *Inverse Problem Theory and Methods for Model Parameter Estimation*. Society for Industrial and Applied Mathematics, Jan. 2005. ISBN: 978-0-89871-572-9. DOI: `10.1137/1.9780898717921`. URL: `http://epubs.siam.org/doi/book/10.1137/1.9780898717921`.

[37] *Gaussian Process Regressor*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.gaussian%7B%5C_%7Dprocess.GaussianProcessRegressor.html` (visited on 03/15/2020).

[38] Karol Zak. *keras-unet: Helper package with multiple U-Net implementations in Keras as well as useful utility tools helpful when working with image semantic segmentation tasks. This library and underlying tools come from multiple projects I performed working.* URL: `https://github.com/karolzak/keras-unet` (visited on 12/03/2019).

[39] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[40] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. Tech. rep. arXiv: `1603.04467v2`. URL: `www.tensorflow.org.`.

[41] M Peter-Borie et al. "Borehole damaging under thermo-mechanical loading in the RN-15/IDDP-2 deep well: towards validation of numerical modeling using logging images". In: (). DOI: `10.1186/s40517-018-0102-7`. URL: `https://doi.org/10.1186/s40517-018-0102-7`.

[42] Sarah Leclerc et al. "Deep Learning for Segmentation Using an Open Large-Scale Dataset in 2D Echocardiography". In: *IEEE transactions on medical imaging* 38.9 (Sept. 2019), pp. 2198–2210. ISSN: 1558254X. DOI: `10.1109/TMI.2019.2900516`. arXiv: `1908.06948`.

[43] *Volve Data Village.* URL: `https://data.equinor.com/dataset/Volve` (visited on 01/03/2020).