Master's thesis

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

Markus Andresen
Simen Nordby Johansen

Deep Convolutional Encoder-Decoder Networks for Digital Rock Porosity Segmentation

Master's thesis in Informatics
Supervisor: Frank Lindseth
July 2019

NTNU
Norwegian University of
Science and Technology

Markus Andresen
Simen Nordby Johansen

Deep Convolutional Encoder-Decoder Networks for Digital Rock Porosity Segmentation

Master's thesis in Informatics
Supervisor: Frank Lindseth
July 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

Digital rock physics (DRP) is a modern approach for characterizing physical rock properties. By modeling grain, multi-phase and pore space in different rock types, institutions can be assisted in locating suitable locations for hydrocarbon storage. Furthermore, producing accurate models of rock structures can economically assist the oil and gas industry in selecting potential reservoir extraction sites. DRP is captured using non-destructive imaging technology, micro computed tomography (micro-CT), producing 2D or 3D images.

Deep learning has in recent years become very successful at accurately and efficiently performing tasks that previously could only be accomplished by humans. Semantic image segmentation is a computer vision task with the goal of assigning class labels to each pixel in an image. This thesis explores the application of multiple deep convolutional encoder-decoder networks to perform semantic image segmentation of gray-scale rock images. The provided datasets consist of three different types of sandstone, Bentheimer, Berea and Carbonate. We present experiments of combining data augmentation techniques to further increase the richness of the dataset.

Results of the different architectures are compared to each other, and the best performing model achieves an intersection over union (IoU) score of 88.6% (with a categorical accuracy of nearly 98%). The best performing network is further extended to include multiple adjacent images as channel inputs to provide additional information. We investigate if a network trained on one or two rock types can generalize and perform accurate predictions on one not included in training.

Sammendrag

Digital rock physics (DRP) er en moderne metode for å karakterisere de fysiske egenskapene til ulike typer stein. Ved å modellere korn-, flerfaset- og porevolum i forksjellige bergarter, kan en assistere institusjoner med å finne egnede lokasjoner for lagring av hydrokarboner. Utover dette kan nøyaktige modelleringer av strukturen i bergarter hjelpe olje- og gassindustrien med å velge potensielle lokasjoner for utbygging av oljefelt. DRP blir fanget opp ved å benytte micro-CT for å produsere 2D eller 3D bilder.

I nyere tid har dype læringsmetoder blitt effektive på å oppnå gode resultater på oppgaver som tidligere kun kunne løses av mennesker. Semantisk bildesegmentering er en datasyns oppgave med mål om å assosiere hver pixel-verdi i et bilde med en klasse. I denne avhandlingen utforsker vi å anvende *deep convolutional encoder-decoder*-nettverk til å utføre automatisk semantisk bildesegmentering av gråtonebilder av bergarter. Dataen vi har jobbet med består av tre forskjellige bergarter, Bentheimer, Berea og Carbonate. Vi presenterer eksperimenter der vi kombinerer dataaugmenterings-teknikker for ytterligere å utvide tilgjengelig data.

Resultatene fra de forskjellige arkitekturene blir sammenlignet med hverandre, og modellen som gir best resultat oppnår 88.6% *intersection over union* (IoU). Den beste arkitekturen utvides til å inkludere flere bilder som ligger ved siden av hverandre, for å gi nettverket mer informasjon. Vi undersøker også om et nettverk trent på en eller to bergarter kan generaliseres og oppnå gode resultater på en bergart som ikke var inkludert i treningsprosessen.

Preface

This thesis was written in Trondheim at the Norwegian University of Science and Technology (NTNU), Faculty of Information Technology and Electrical Engineering, Department of Computer Science. The thesis was accomplished in cooperation with Leonardo Ruspini at Petricore Norway AS, which provided us with both data and computing power.

Supervisor: Frank Lindseth

Shift in Research Focus

The main focus of this thesis was originally to explore the use of computer vision and deep learning techniques for automatic interpretation of seismic data. Therefore, the first half of this project went into understanding seismic, and researching relevant literature that applies deep convolutional neural networks to this domain.

The base of the research was to do automatic segmentation of faults and facies below the seabed. Braathen et. al defines a fault as: *"Faults are considered strained volumes of rock, defining a three-dimensional fault envelope in which host-rock structures and petrophysical properties are altered by tectonic deformation*." This meaning a clear deformation or offset in the rock structure. Facies is a geological term somewhat diffuse in its definition. The concept of facies has traditionally been applied to descriptions of sedimentary and metamorphic (change in form) rocks [1]. The end goal for this project was to develop a method to accurately predict and identify horizons (separators between different sediment layers) and faults below the seabed, in order to potentially locate oil and gas depositories.

Due to a delay in the dataset to be provided, initial experimentation was focused on a challenge posted on Kaggle. Kaggle is a website encouraging professionals and amateurs to progress the field of Data Science and Machine Learning. The challenge, named TGS Salt Identification Challenge, asked competitors to develop a model for segmentation of salt bodies beneath the Earth's surface, which is also a problem in the seismic domain. Many of the large accumulations of oil and gas also have huge deposits of salt below the surface.

However, in February it was decided to shift focus from seismic interpretation to digital rocks and rock property segmentation. This decision was made because of dataset problems that would not be feasible to resolve. Appendix A describes in detail the problems encountered with the seismic datasets that led to this shift, and documents some of the efforts that went into researching the seismic domain.

Table of Contents

Abstract i

Sammendrag ii

Preface iii

Shift in Research Focus iv

Table of Contents vi

List of Tables vii

List of Figures ix

Abbreviations x

1 Introduction 1
- 1.1 Background and Motivation . . . 1
- 1.2 Goals and Research Questions . . . 2
- 1.3 Research Method . . . 2
- 1.4 Contribution . . . 2
- 1.5 Thesis Structure . . . 3

2 Background 4
- 2.1 Digital Rocks . . . 4
- 2.2 Computer Vision . . . 5
  - 2.2.1 Object Recognition Tasks . . . 5
  - 2.2.2 Applications . . . 6
- 2.3 Deep Learning . . . 6
  - 2.3.1 Motivation . . . 7
  - 2.3.2 Machine Learning . . . 7
  - 2.3.3 Artificial Neural Networks . . . 12
  - 2.3.4 Convolutional Neural Networks . . . 16
- 2.4 Related Work . . . 22
  - 2.4.1 Structured Literature Review Protocol . . . 22

2.4.2 Historical Context of Semantic Segmentation . . . 25
2.4.3 Segmentation of Digital Rock Images Using Deep Convolutional Autoencoder Networks . . . 25
2.4.4 Fully Convolutional Networks for Semantic Segmentation . . . 26
2.4.5 U-Net: Convolutional Networks for Biomedical Image Segmentation . 27
2.4.6 Deep Residual Learning for Image Recognition . . . 27
2.4.7 Segmentation of Coronary Arteries from CT-scans of the Heart Using Deep Learning . . . 29

3 Methodology **31**
3.1 Digital Rocks Dataset . . . 31
3.1.1 Generation of Ground Truth . . . 34
3.2 Developing a Digital Rock Segmentation Network . . . 34
3.2.1 Digital Rock Network (DRNet) . . . 35
3.2.2 Pyramid Digital Rock Network (PDRNet) . . . 36
3.2.3 Hybrid Digital Rock Network (HDRNet) . . . 38
3.3 Evaluation Metrics . . . 39

4 Experiments and Results **41**
4.1 Experimental Plan . . . 41
4.2 Experimental Setup . . . 41
4.2.1 Hardware, Software and Environment . . . 41
4.2.2 Data Preprocessing and Augmentation . . . 42
4.2.3 Hyperparameters . . . 45
4.3 Experimental Results . . . 45
4.3.1 DRNet vs PDRNet . . . 45
4.3.2 HDRNet . . . 47
4.3.3 Results on Different Types of Sandstone . . . 47

5 Discussion and Conclusion **51**
5.1 Discussion . . . 51
5.2 Conclusion . . . 53
5.3 Future Work . . . 53

Bibliography **54**

Appendices **58**

A Seismic and Salt Identification Challenge **58**

B 2017 Kjerland, Digital Rocks Results **70**

C Bibliography URLs **79**

List of Tables

2.1 Search Words 23
2.2 Inclusion and quality criteria 24
2.3 The final selection of included articles. 24

3.1 Intersection over Union between each user and the segmented average, and shows us that Carbonate gives users the most different results (a data parsing issues resulted in some missing values for user 4). 34

4.1 Weak Augmentation 43
4.2 Medium Augmentation 43
4.3 Strong Augmentation 44
4.4 DRNet, test set results. 45
4.5 PDRNet, test set results. 46
4.6 HDRNet, test set results. 47
4.7 Hybrid network trained only on Bentheimer, tested on all three rock types separately. 49
4.8 Hybrid network trained only on Berea, tested on all three rock types separately. 49
4.9 Hybrid network trained only on Carbonate, tested on all three rock types separately. 49
4.10 Hybrid network trained on Bentheimer and Berea, tested on all three rock types separately. 50
4.11 Hybrid network trained on Bentheimer and Carbonate, tested on all three rock types separately. 50
4.12 Hybrid network trained on Berea and Carbonate, tested on all three rock types separately. 50
4.13 Hybrid network trained on Bentheimer, Berea and Carbonate, tested on all three rock types separately. 50

5.1 Table showing the dice scores of Kjerland and our thesis, when trained with different combinations of rock types. BT = Bentheimer, BR = Berea, CA = Carbonate. Kjerland dice scores are computed by averaging the scores for each experiment listed in the Appendix, Section B, Kjerland page 51. 52

List of Figures

2.1 Visual explanation of the different object recognition tasks covered in this section, figure adapted from [2]. 6
2.2 Flowchart that shows the components in different types of machine learning pipelines, grayed out boxes is components that learns from data. Figure adapted from the Deep Learning textbook [3]. 8
2.3 Three different models fitted to the same data points. Model to the left underfits and is neither able to fit to the training data well, nor generalizing to new data points. The model in the middle perfectly fits to the training data and has captured the underlying structure present in the data and will most likely generalize well to new data points. Model to the right overfits and is perfectly fit to all data points but has not captured the underlying structure present in the data and will probably not do well on unseen data. Figure adapted from the Deep Learning textbook [3]. 9
2.4 Optimal capacity. Figure from Deep learning textbook [3]. 11
2.5 A neuron/perceptron. Figure adapted from [4]. 13
2.6 Basic feed forward network with one hidden layer. Figure adapted from [5]. . . 13
2.7 Illustration of gradient descent, from the Deep Learning textbook [3]. 15
2.8 Batch Normalization transform applied to activation x over a mini-batch. Equation from the original paper [6]. 15
2.9 Sobel edge detection kernels of size 3x3. 17
2.10 Shows the first step in a convolution process. Figure adapted from [7]. 17
2.11 Shows resulting features after applying horizontal and vertical sobel filter on the input image, images from [8]. 18
2.12 The top image shows a convolution on input $x3$ with a kernel width of 3. We can see that only 3 of the outputs are affected. In contrast, the bottom image shows that every output is affected when using matrix multiplication (ANN). Figure adapted from [3]. 19
2.13 Illustration of pooling. Figures adapted from [9]. 20
2.14 Transposed Convolution; blue is input, green is output, figure adapted from [7]. 21
2.15 Visualization of how an 3d kernel is applied to an 3d input array, Figure from Kaggle notebook [10]. 22
2.16 The Fully Convolutional Network (FCN) architecture. Figure from paper [11]. . 27
2.17 U-Net architecture, figure from original paper [12]. 28
2.18 Single Residual Block, a building block. Figure adapted from [13]. 28

3.1 Shows the class distribution for the different sandstone types, Bentheimer and Berea has a very similar distribution, Carbonate differs slightly. 32
3.2 Core samples of sandstone, Berea (left), Bentheimer (bottom) and Carbonate (right). 32
3.3 Four slices from each sandstone volume. Left image shows the Micro-CT, to the right the corresponding segmented ground truth. In the ground truth slices black corresponds to pore, grey to multi-phase and white to grain. 33
3.4 DRNet architecture, number of residual down-sampling blocks varies. 35
3.5 Lateral connection for PDRNet, one for each bottom-up and top-down block. . 36
3.6 PDRNet high level architecture, details are left out to highlight differences with DRNet. 37
3.7 Visualization of input x and ground truth y used to train the hybrid model, two depth slices are added on each side of the current slice to segment (in red), as additional channels. 38
3.8 A 5-channeled input convolved with a 1x1x3 filter resulting in a 3-channeled output. Applying a non linearity (ReLU) to this output allows the network to learn a mapping that can utilizes the depth information available in the input. . 39
3.9 Intersection over Union. 40

4.1 Validation IoU and training time for HDRNet with increasing amount of depth channels. Results show that performance increases with additional channels before it saturates, training time also increases quite a lot. We argue that HDRNet with 4 depth channels is a good compromise between performance and training time for this task. HDRNet with zero depth channels is equal to DRNet and provided as a baseline. Data extracted from Tensorboard and plotted manually. 47
4.2 Plotted predicted segmentation maps for a slice of Carbonate during different stages of training and ground truth (grain in yellow, multi-phase in green and pore in purple). These plots indicate that the network only uses half a epoch before it is able to differentiate between pore and grain, and that most of the training time is spent learning how to properly segment multi-phase. 48
4.3 Confusion matrix shows that the model seldom confuses grain and pore, this only happens 1% of the time. Most confusion occurs between pore and multi-phase, together with multi-phase and grain. Results also shows that grain is the easiest class to predict thereafter pore and multi-phase which is the most challenging. Matrix was generated with Matplotlib using the best performing model on the test set. 49

A.1 Seismic survey by vessel (illustration from RagnarockGeo AS). 58
A.2 Illustration of a slice in a 3D seismic volume 59
A.3 TGS Salt Data. The left image shows the original image, while the right is the mask where blue pixels belong to sediment and red salt 60

Abbreviations

ANN	=	Artificial Neural Network
CNN	=	Convolutional Neural Network
CPU	=	Central Processing Unit
CT	=	Computed Tomography
DRNet	=	Digital Rocks Network
DRP	=	Digital Rock Physics
FCN	=	Fully Convolutional Network
FPN	=	Feature Pyramid Network
GPU	=	Graphical Processing Unit
HDRNet	=	Hybrid Digital Rocks Network
IoU	=	Intersection over Union
ML	=	Machine Learning
PDRNet	=	Pyramid Digital Rocks Network
ReLU	=	Rectified Linear Unit
RGB	=	Red Green Blue
RQ	=	Research Question
SVM	=	Support Vector Machine
SLR	=	Structured Literature Review
TPU	=	Tensor Processing Unit

Chapter 1

Introduction

1.1 Background and Motivation

The future of our planet depends on our capacity to stop emitting CO2 to the atmosphere. However, we also need to keep growing in order to reduce poverty and improve the quality of life in developing countries. To achieve this, Carbon dioxide Capture and Storage (CCS) and more efficient oil and gas production will play an important role. The derivation of rock properties from high-resolution images (Digital Rock) is a disruptive technology in that it can fundamentally change the way rocks are characterized. Digital rocks is a term commonly used to refer to 3D rock images obtained through micro-CT imaging. Micro-CT has emerged as a technology that can provide valuable insights in understanding the properties of porous media such as rocks. The industry standard for analyzing and segmenting rock samples today are operator dependant and to a large extent done manually by domain experts. This is however time-consuming and includes the risk of human error. Rock-analysis can help the oil and gas industry to better determine extraction strategies for reservoirs as well as locating potential rock beds suited for storing CO2. This thesis explores the possibility for automatically segmenting micro-CT images of different rock types into three properties: grain, micro-phase and pore.

Semantic image segmentation and object detection have become more and more feasible in recent years, due to advancements in convolutional neural networks (CNN) and improved hardware support. Deep convolutional neural networks (DCNN) has further improved the ability to accurately segment and classify images. Image segmentation is the process of assigning labels to each pixel in an image. Image segmentation plays an important part in the success of many domains such as medical imaging, face recognition and self-driving cars.

Ronneberger et. al. proposed in 2015 a new method for segmenting neuronal structures in electron microscopic stacks[12]. The network was a breakthrough in the segmentation domain as it performs much faster than previous models, while maintaining spatial awareness. In addition it performs well on small datasets due to data augmentations with elastic deformations. The segmentation problem presented in that paper is similar to that of digital rock segmentation. This thesis will focus mainly on automatic segmentation of high resolution 3D rock images using deep learning.

This thesis continues research of a previous method used for automatic image segmentation of medical datasets, which additionally was tested on rock data, based on a 3D convolutional

network called DeepMedic. Because of the computational requirements of 3D convolutions on large volumes, we address the task of segmenting the rock structures using a 2D based approach, to see if it is able to perform as good or better. Furthermore, we extend this 2D architecture to also include adjacent depth images as additional channels to supply more information to the network.

1.2 Goals and Research Questions

The overall goal of this thesis is to explore the use of deep learning and convolutional neural networks for sandstone pore segmentation on digital rock volumes. We also investigate and compare the performance of 2D and 3D convolutional architectures on this task.

Research question 1 *How does 2D-convolutional architectures compare to a 2.5D hybrid architecture that includes depth as input channels?*

Sub-goal Develop the best possible 2D-convolutional segmentation network.

Sub-goal Extend this network to include an arbitrary number of depth channels, and see if it improves performance.

Research question 2 *Can deep learning models trained to segment porosity of one rock type generalize and accurately segment other types of rock?*

Sub-goal Compare and evaluate models trained on various subsets of rock types.

1.3 Research Method

This thesis conforms to a *design science* methodology, defined as science which attempts to create something that serves human purposes, as opposed to natural science which tries to understand reality [14]. It follows a technology oriented approach in that the acquired results are assessed against criteria of value/utility: does it work or is it an improvement? Instead of producing theoretical knowledge, this thesis presents a way to apply knowledge of tasks (Deep Learning) or domains (Digital Rocks and seismic) to create effective artifacts (something that occurs as a result of a preparative or investigative procedure). The working framework consists mainly of two basic activities, build and evaluate. Building is the task of developing models and implementing deep learning methods, while the evaluation activity is the process of determining how well an implementation performs. Although this thesis is experiment driven, there is a need for some theoretical research in order to understand and explain why and how the resulting models and implementations work.

1.4 Contribution

This thesis contributes to the continued research into digital rocks in conjunction with deep learning, specifically the use of convolutional networks for segmentation of rock properties. The main contribution is a network architecture that efficiently performs automatic segmentation of porosity in Bentheimer, Berea and Carbonate sandstone samples. We anticipate that this applies

to other types of stone as well. In digital rocks physics this can replace conventional multi-thresholding algorithms, which are difficult to automate since they are based on image color contrasts, often requiring parameter tuning to fit each sample. This thesis also shows that the performance of this network can be improved, by providing some adjacent slices as additional input channels. This idea is not exclusive to rock property segmentation. It can be applied to any network that segments volumes by taking predictions one slice at a time.

1.5 Thesis Structure

This section presents the structure of the thesis and gives a brief overview of the contents of each chapter.

Introduction: Presents the background, motivation, research questions and the main contributions of the work presented in this thesis.

Background: Gives a brief overview of the fields of computer vision, deep learning and digital rocks in order to provide the reader with the understanding necessary for this thesis. Finally, we present related work and describe our structured literature protocol.

Methodology: This chapter presents the Digital Rocks dataset and its creation. Then it details three different architectures for pore segmentation on this dataset. Finally, it discusses evaluation metrics used to compare our models.

Experiments and Results: This chapter starts by presenting the experimental plan. Then we talk about the experimental setup, including hardware, tools data pre-processing, augmentations and hyperparameters. Finally we list the results obtained from running the experiments.

Discussion and Conclusion: This chapter concludes the work presented in this thesis. It starts by discussing the findings presented in Chapter 4, before presenting the conclusion. Finally, we discuss future work that can potentially improve our findings and results.

Chapter 2

Background

This chapter provides the reader with the theory and background necessary for this thesis. Section 2.1 explains what Digital Rocks is, how it is collected and its applications. Section 2.2 presents the field of computer vision and object recognition tasks. Section 2.3 gives a brief overview of deep learning and convolutional neural networks, which in recent years has been dominating in visual recognition problems. Section 2.4 presents related work and describes our structured literature protocol.

2.1 Digital Rocks

Digital Rock Physics (DRP) is an innovative approach for computing and analyzing the properties of rocks. DRP aims to provide a better understanding of rocks such as flow transport units and geometrical properties. Some rock properties can be very hard and expensive to measure in the laboratory, which is one of the driving factors for investments in the conjunction of Machine Learning and DRP. The combination of laboratory measurements and deep learning will compliment well(oil) log analysis and other insights. Improved understanding of rock properties can open new possibilities for economic exploration and extraction for hydrocarbons. Furthermore it will give valuable insights in the ability to efficiently store CO2 in various rock types.

The paradigm of DRP is image-and-compute: the rock sample is imaged to obtain a 2D or 3D representation of the mineral phase and pore space, and this representation is then used to simulate physical processes in the sample. DRP normally consists of three steps:

1. Digital imaging to create a digital representation in 2D/3D at large enough resolution to capture rock features as pores, organics and grains.

2. Digital image processing to categorize pixels

3. Digital analysis to model rock properties

The data used in this thesis is provided by Petricore, which is a company focusing on digital rocks and core analyses. Their services deliver 3D digital rock models that enables simulations of multiple petrophysical and dynamic rock properties. They use nondestructive imaging techniques to provide high-value images for visualization and core assessment. Computed tomography scanners (CT), similar to those used in medial applications, are used to capture 3D images of rock samples. We have worked with data captured by Micro-CT which provide 20-200 times

higher resolution compared to traditional CT-scans. This provide very detailed images of the rock sample.

2.2 Computer Vision

Computer vision is the task of designing computer systems that can gain high level understanding of digital images or videos and to automate activities that the human visual system is able to do. Computer vision is closely related to the field of digital image processing and it is hard to define a sharp boundary between the two fields. A common distinction is that in digital image processing both input and output is an image, while computer vision often in addition to producing an output image, also tries to understand the image in a higher level than a matrix of pixel values which makes it possible to answer questions like; What is the image of? Which person is in the image? Where in the image is this persons face? The biggest difference between the two fields is in goals, not methods.

There are many sub-domains of computer vision. The one relevant to this thesis is object recognition, which contains several well defined tasks that aims to recognize objects in images with different levels of granularity. Common for all these tasks is that they consume digital images or video in order to produce numerical information that serves as more suitable descriptions of the world, that can be communicated and used in decision making.

2.2.1 Object Recognition Tasks

This section explains different object recognition tasks from the coarse to fine in information detail. See Figure 2.1 for a visual explanation.

Image Classification is the task of classifying which object is present in an image. This is trivial for humans and require almost no effort, but was for a long time considered notoriously hard for computers. Before the break-trough of deep learning, state of the art image classification algorithms was nowhere near human performance.

Classification with Localization is an extension to image classification. In addition to classifying which object is present in a given image, classification with localization also specifies the spatial location of the object. This is commonly done by drawing a bounding box around the object.

Object Detection is classification with localization of multiple objects in an image. Object detection is performed using a technique called sliding window. For an input image, multiple windows of different sizes are slid on top of the image, and for each window a prediction is made. This process can be very computationally expensive when images are large and a high number of windows are used.

Semantic Segmentation is very similar to object detection but are more fine grained. Segmentation aims to infer labels for every pixel, so that each pixel is labeled with the class of its enclosing object ore region.

Instance Segmentation goes one step further, in addition to assigning a class to every pixel it separates between different instances of that class.

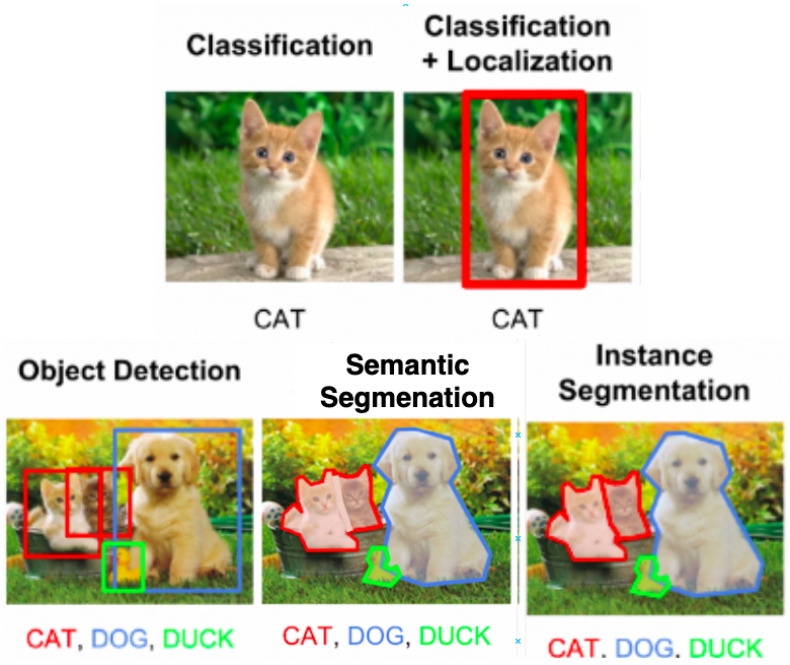


Figure 2.1: Visual explanation of the different object recognition tasks covered in this section, figure adapted from [2].

2.2.2 Applications

Computer vision is an important success-factor in many different applications today. It is used in self driving car technology, facial recognition, biomedical imaging and image search, to name a few. The applications are expanding increasingly along with improved performance of computer vision systems, much driven by the success of deep learning.

2.3 Deep Learning

Deep learning is an approach to artificial intelligence that allows computers to learn from experience, by learning multiple levels of representation that corresponds to some level of abstraction. Each level represents abstract features that are discovered from the features represented in the previous level. Hence, the level of abstraction increases with each level. We say that these levels forms a hierarchy of concepts where each concept is defined in relation to simpler concepts. If we draw a graph of this hierarchy, the graph is deep with many layers, which is the reason we call this approach Deep learning. Deep learning is not something new, and have existed since the 1940s under various names. However, it is not until recent years that deep learning has really gotten its momentum. This growth in both popularity and usefulness is in large parts due to increased amount of available data as a bi-product of digitization, increased computational power and new techniques that allows for training of even deeper models. Today deep learning algorithms are successfully applied to fields like computer vision, natural language processing, machine translation, recommender systems and speech recognition and in many cases works so

well that they have completely replaced traditional approaches all together [3]. Deep learning is a type of representation learning which again is a type of machine learning.

2.3.1 Motivation

In traditional machine learning, the important features of the input are manually designed and the system automatically learns to map the features to outputs. This is called feature engineering. Coming up with features requires a lot of domain knowledge and is often both difficult and time consuming. Even tough humans are able to classify objects in an image, we are not always able to describe the features in terms of pixel values that lead us to that classification. A better approach would be to skip this step, and instead design algorithms that inputs raw data and learns which features that are important in order to solve the task. This is what we call representation learning and deep learning is a kind of representation learning that builds concept hierarchies that consist of multiple levels of features, where each level is input to the next one. This allows us to design models that are trained end-to-end, without feature engineering, that solves problems which was not possible before. These models can generalize to different domains and sometimes even transfer knowledge learned from solving one task, to better solve another.

We will in the next sections introduce some basic machine learning concepts before we talk about the quintessential deep learning model, the deep feed-forward neural network. Section 2.3.4 discusses a class of neural networks called convolutional networks, inspired by the visual cortex in humans and often applied to visual imagery. This is the model that we in this thesis further explore for use in conjunction with digital rock physics (DRP).

2.3.2 Machine Learning

Machine Learning is an approach to artificial intelligence that uses statistical models that learn from data, in order to perform a specific task without any task specific instructions. This allows us to solve tasks that are too difficult to solve with programs of logical rules written by human programmers. Learning algorithms builds experience from datasets containing examples where each example is a collection of features recorded from some object or event. Examples are represented as a vector $x \in \mathbb{R}^n$ where each entry x_i corresponds to a feature. Learning algorithms differ in approach, the type of data they input and output, and the type of task or problem that they are intended to solve. Most algorithms can be classified as either being supervised or unsupervised.

Supervised learning algorithms builds experience from a dataset that contains examples of features where each example X is associated with a label Y. The goal of supervised algorithms is usually to learn a function that maps new unseen examples to a label $F : X \to Y$. Supervised learning are commonly used to solve tasks like regression and classification.

Unsupervised learning algorithms builds experience from a dataset containing examples without any explicit labels. The goal of these algorithms are to discover the underlying structure of a dataset which again can be used to build useful models. An example of unsupervised learning is clustering, which tries to divide the supplied dataset into clusters of similar examples.

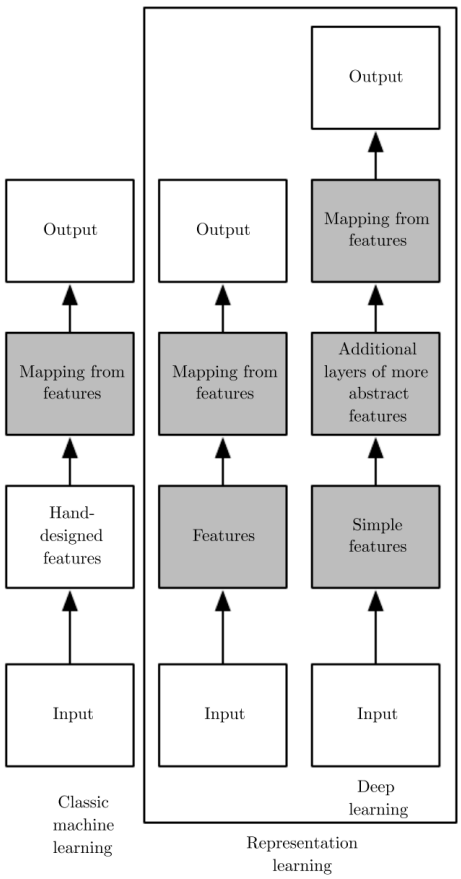


Figure 2.2: Flowchart that shows the components in different types of machine learning pipelines, grayed out boxes is components that learns from data. Figure adapted from the Deep Learning textbook [3].

2.3.2.1 Overfitting and Underfitting

Machine learning algorithms are useless if they only perform well on the data used to train them. The ability to perform well on new/unseen data is what truly matters. If an algorithm does this we say that it is able to generalize well. When training models we use a training set and compute an error measure which we call the training error. The training error is reduced during training, but what we really want is to reduce the test error. This is accomplished by computing an error measure on a test set that contains unseen data. The two factors that determine how well machine learning algorithms perform is the ability to make the training error small and make the gap between training and test error small. These two factors corresponds to whats called underfitting and overfitting. We say that a model is underfitting when it is not able to reduce the training error to a sufficient level, while overfitting occurs when the gap between training error and test error is to large. The goal of not only minimizing error, but generalization, is what separates machine learning from mathematical optimization. One way to control whether a model is likely to underfit or overfit is by altering its capacity (controlling the set of functions that the model can select as possible solutions). Low capacity limits a model from fitting complex patterns which can cause underfitting. High capacity gives a model the ability to perfectly fit all possible patterns, thus increasing the possibility of overfitting. The challenge is to select a capacity that is proportional to the complexity of the dataset and the task that one is trying to solve.

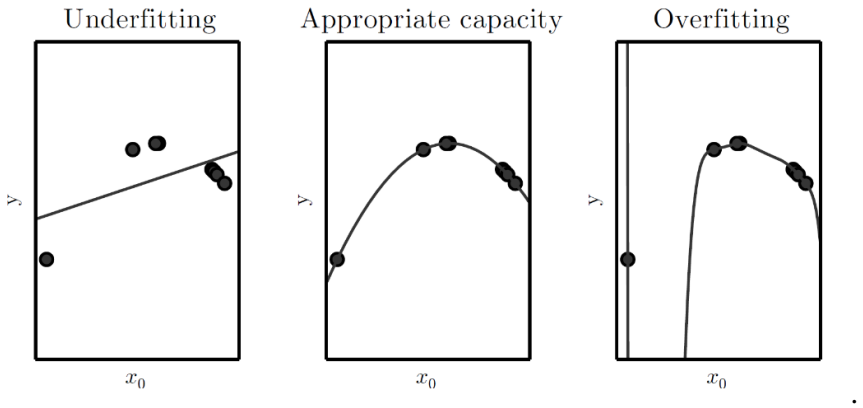


Figure 2.3: Three different models fitted to the same data points. Model to the left underfits and is neither able to fit to the training data well, nor generalizing to new data points. The model in the middle perfectly fits to the training data and has captured the underlying structure present in the data and will most likely generalize well to new data points. Model to the right overfits and is perfectly fit to all data points but has not captured the underlying structure present in the data and will probably not do well on unseen data. Figure adapted from the Deep Learning textbook [3].

2.3.2.2 Hyperparameters

A hyperparameter is a parameter which is set manually before initiating the learning process, in contrast to regular parameters which values are derived during training. Hyperparameters affect learning algorithms ability to learn and thus also their performance. This is challenging because there is no set of values that always works. Appropriate values are dependent on both task and dataset. Finding appropriate values are called hyperparameter tuning. There exists search algorithms that can find these values, but this involves training a model multiple times with

different set of hyperparameter values. This is not always feasible when applying deep learning because training can be very time consuming. For this reason deep learning hyperparameter tuning is commonly done using heuristics and trial and error.

2.3.2.3 Optimization

The majority of supervised learning algorithms involve some sort of optimization. Optimization is the task of either minimizing or maximizing a function $f(x)$. The function is referred to as either a cost or error function. The cost function is a function that is used to measure inconsistency between the predicted value ($\hat{y}$) and the actual label (y). The function outputs a non-negative value where a smaller value indicates an increase in the robustness of the network. Optimization is finding the set of parameters θ that greatly reduces the cost function $J(\theta)$.

2.3.2.4 Dataset Split

When doing supervised learning, to quantify the performance of a model and detect if we are underfitting or overfitting, we need some previously unseen data. This is because performing really well on the training data gives no indication of true generalization. A common approach to this problem is to divide all available data into three non-overlapping partitions called the *training set*, the *validation set* and the *test set*. A common way to split is to put 70% of data in the training set, and 15% both in the validation and test set, but many other splitting schemes are also reasonable. It is important that the splitting is done in way such that data in the training set is representative to that of the validation and test set.

Training Set
This is the data available to the learning algorithm during training and used to update model parameters to minimize loss, thus also expected loss on the validation and test set. This set should by far be the biggest partition of the three.

Validation Set
Used to validate and tune hyperparameters during the learning phase. How well the model perform on the validation set should not be used in any way to update parameters. If performance on the training set continues to increase while it is decreasing on the validation set, the model is likely overfitting.

Test Set
Used when the model is completely finished with training and tuned against the validation set. This important to prevent *peeking*, which is to use the test set performance to both choose a hypothesis and evaluate it [15, p. 709]. Test set performance is the closest we can get to that of a model deployed in the real world, given that the data is drawn from the same probability distribution.

2.3.2.5 Regularization

A common problem with learning algorithms is that they perform well on the training data, but has problems with new input, i.e. they overfit. Regularization are strategies designed to reduce test error possibly at the expense of higher training error. Regularization has a broad definition and is defined as any method or technique that are designed to reduce a models generalization

error but not the training error. This is important because we want our algorithms to perform well not only on training data but also on unseen inputs. Regularization is about adjusting estimators by trading increased bias for reduced variance. A good regularizer is one that greatly reduce variance but does not increase the bias too much. Some regularization techniques are universal and can be applied to almost any learning algorithm while others are more model specific. Below we cover some of the well known and used techniques.

Parameter Norm Penalties
Regularization technique that adds a penalty to the objective function. By adding a hyperparameter, $\alpha \in [0, \infty)$, to our objective function, we can limit the capacity of our model. Setting α to 0 gives us no regularization, while a large number gives more. The most common parameter norm penalty is L2, which is also known as weight decay.

Early Stopping
Universal regularization technique that can by applied to any supervised learning algorithm. Works by halting training when it detects that overfitting is beginning to occur. Overfitting is detected by comparing training error with validation error, if training error keeps decreasing while validation error plateaus then increases, we are overfitting. Early Stopping is often implemented using a patience parameter that specifies how many epochs (complete passes over training set) of non decreasing validation error to tolerate, before halting training. The red line in Figure 2.4 shows the optimal capacity of a model and the goal of early stopping is to get as close to this line as possible.

Data Augmentation
A collection of techniques that aims to increase the diversity of training data available, thus minimizing the risk of overfitting. Further discussed in Section 2.3.3.5.

Dropout
Regularization technique specific to artificial neural networks. In each training step an individual node is either dropped out of the network with a probability $1 - p$ or kept with a probability p. It is important to note that dropout should only be active during training, when using the network for inference all nodes in the network should be considered.

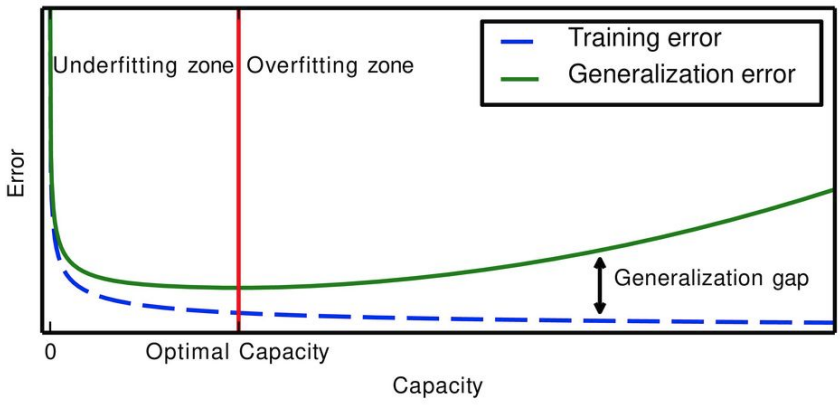


Figure 2.4: Optimal capacity. Figure from Deep learning textbook [3].

2.3.3 Artificial Neural Networks

Artificial Neural Networks (ANN) are networks loosely inspired by human biology and how neurons in the human brain functions. The original goal of the ANN was to solve problems like humans. However, over time the attention moved to solving specific tasks like computer vision, speech recognition, machine translation and medical diagnosis. This has led to a deviation from biology. The goal of a neural network is to approximate some function f. The neural network itself is not an algorithm but a tool used in different machine learning algorithms to process complex data into a space that computers understand. Neural networks generally do not need to understand and be programmed with specific rules that specifies what to expect from the input. The ANN instead learns from many examples with the correct answer labeled and using this labeling to extract what characteristics from the input are needed to calculate correct output.

2.3.3.1 Neurons and Layers

The ANN consists of a collection of nodes called artificial neurons which loosely mimics the neurons of a brain. Each connection between two nodes (like the synapse in a biological brain) can transmit a signal from one to the other. A node receives and process one signal and then signal additional nodes connected to it. The signal in an ANN is usually a real number, and the output of each node is computed by a non-linear function of the sum of its inputs. Artificial neurons typically have a weight that adjusts as the learning progress. The weight increases or decreases the strength of a signal. It is also common that the neurons have a threshold in such that the signal is only sent if the aggregate signal cross that threshold.

A neuron or node consists of several components as illustrated in Figure 2.5. The output of some nodes may be input to others. Each node take multiple weighted inputs (a weight vector) and applies an activation function to the sum of inputs. An activation function takes the input of the node and maps it to an output signal. The linear output of a node n_j can be calculated by:

$$Output(n_j) = b_j + \sum_{i=1}^{n} x_i w_i$$

where x_i is the output from the preceding node and w_i is the weight. A bias (b_j) is also commonly used in each neuron to be able to change when the node activates.

Multiple neurons in a connected acyclic graph generate what is called a feed forward neural network (as seen in Figure 2.6). Such networks may include one or more hidden layers, and when multiple hidden layers are present the network is usually referred to as a deep neural network.

2.3.3.2 Activation Functions

The use of non-linear activation functions allows neural networks to approximate any function, not only linear ones. Below some of the most commonly used activation functions are presented.

Sigmoid Closely resembles how we think neurons fire in the brain but suffers from many drawbacks which makes it rarely used today, except in the output layer when doing regression. Some of the drawbacks include slow convergence and the vanishing gradient problem (which prevents the weight from changing its value). Another problem with Sigmoid is that it is not zero cen-

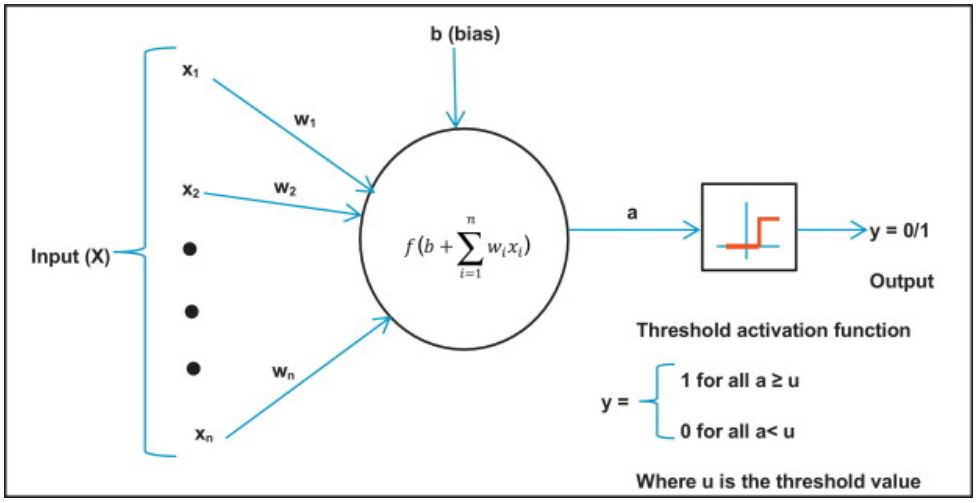


Figure 2.5: A neuron/perceptron. Figure adapted from [4].

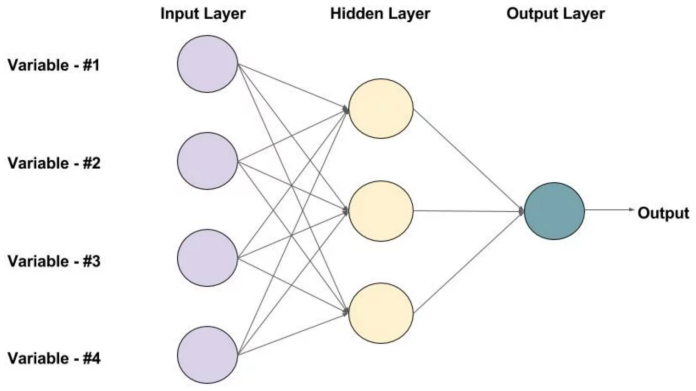


Figure 2.6: Basic feed forward network with one hidden layer. Figure adapted from [5].

tered, i.e the output is between 0 and 1. This cause problems because it makes the gradients on the weight either negative or positive.

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

Hyperbolic Tangent (TanH) Solves the zero centered problem, the output is between -1 and 1. However, it still suffers from vanishing gradient.

$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.2}$$

Rectified Linear Unit (ReLU) Suffers less of the vanishing gradient problem since it only saturates in one direction. It is by far the most common activation function in use today.

$$f(x) = relu(x) = max(0, x) \tag{2.3}$$

2.3.3.3 Gradient Descent

Gradient descent is an optimization algorithm used to minimize some cost function. In an ANN the algorithm is used to adjust the weights in the graph. Gradient descent works by iteratively moving in the direction of steepest descent until it reaches a local minimum (see Figure2.7). How big steps are taken in each iteration is determined by the learning rate. With a high learning rate, the algorithm completes faster but introduces a risk of overstepping the local minimum. A lower learning rate gives more precise results, but at the cost of computational time.

Gradient descent suffers when training sets become large which makes it very computationally expensive ($O(m)$ where $m = trainingsetsize$). A solution to this problem is an extension of the algorithm called stochastic gradient descent (SGD). The idea of SGD is that the gradient is an expectation and that we can get a stochastic approximation of the gradient using only samples of training data. Following this idea SGD use a sample batch, called minibatch, from the training set at each step of the algorithm. The minibatch size typically includes a couple to a few hundred samples. This makes it possible to fit billions of examples using only a few hundred. A technique to further improve the computational speed of SGD is the introduction of momentum. Momentum is particularly useful for gradients that are of high curvature, small and consistent or noisy. The algorithm accumulates an exponentially decaying average of past gradients and continues to move in their direction.

Backpropagation
Backpropagation is an algorithm used to compute the gradients for each weight in the network, flowing backwards from the cost function throughout the network. Generating an expression for the gradient is simple, but evaluating it can be expensive. This is what backpropagation does efficiently, by recursively applying the chain rule. It is important to note that backpropagation by itself is not a learning algorithm, it is used only to compute gradients, whereas an optimizer algorithm like stochastic gradient decent uses these gradients to update weights.

2.3.3.4 Batch Normalization

Batch Normalization [6] is a technique that improves learning in neural networks in addition to having a regularization effect. Batch normalization normalizes the output of a previous acti-

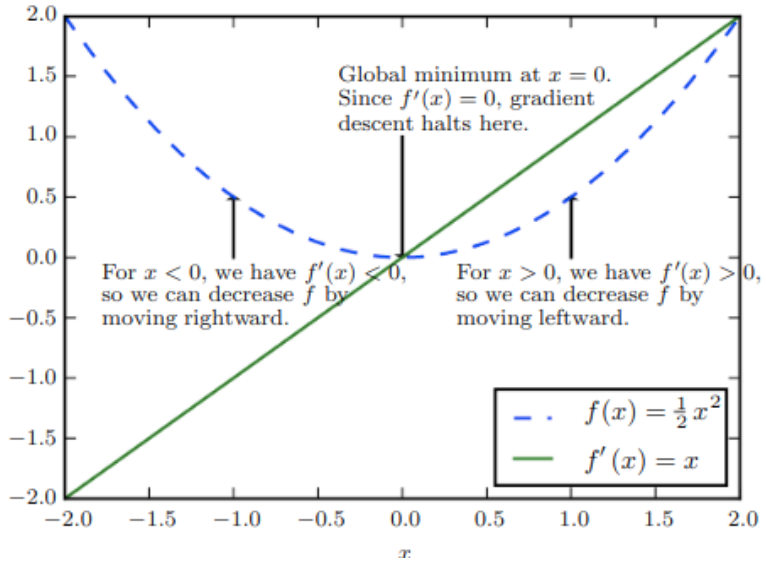


Figure 2.7: Illustration of gradient descent, from the Deep Learning textbook [3].

vation layer by subtracting the batch mean and dividing by the batch standard deviation. Batch normalization can be applied to any layer in the network, making it very effective and allows each layer of a network to learn by itself a little bit more independently of other layers. When using batch normalization no activation becomes very low or high. This is great because it allows for training with higher learning rate yielding faster convergence, in addition to having a regularizing effect because of the noise it adds to each of the hidden units activation.

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \texttt{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \texttt{// mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \texttt{// normalize}$$

$$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \texttt{// scale and shift}$$

Figure 2.8: Batch Normalization transform applied to activation x over a mini-batch. Equation from the original paper [6].

2.3.3.5 Data Augmentation

Data augmentation is the process of enriching a collection of data. This is particularly important when working with small datasets in order to have enough training data. A recurring problem when working with limited datasets is that models trained on them do not generalize well. By having more training data the network can reduce overfitting and learn its domain on a more general level. Even when data of poorer quality is introduced, an algorithm can perform better as long as useful data is possible to be extracted [16]. Common and generic methods for

augmenting image data is to run algorithms that augments color and alters geometric shapes. This includes reflecting, flipping, cropping and translating the image. Below are descriptions of the most commonly used data augmentation strategies when working with machine learning on images.

Traditional Transformations - for each input image a duplicate is produced where it is either shifted, scaled, rotated, flipped or distorted. The augmented image can also be a combination of these. Such simple distortions can be generated by applying affine displacement fields to images[17]. Both the original and the copy is sent through the network. So if a network of is of size N, a dataset of $2N$ is generated.

Elastic Deformation is a technique where a deformation is applied to the input image. It works by creating creating a grid of displacement fields that has random directions, so it becomes like a grid of transformed squares.

Color Augmentations: brightness, contrast, saturation, hue

Blur Augmentations - There are different ways to blur an image. This can be don linearly or non-linearly. Gaussian blur is a linear function which can lose edges in the image. Median is non-linear and replace pixel values with the median color value in the neighbouring pixels.

Emboss and Sharpen are techniques that translates the image. Emboss add a stamp-like effect on an image, while sharpening highlights edges.

2.3.4 Convolutional Neural Networks

Convolutional Neural Networks (CNN) is a category of neural networks that has proven to perform extremely well in computer vision tasks. As the name implies, the network employs a mathematical operation called convolution. A CNN is simply a neural network where the convolution operation is used in at least one of the layers, instead of regular matrix multiplication. A convolution network can be applied to many types of data, but this section will focus mainly on image data.

2.3.4.1 The Convolution Operation

The convolution operation, in its simplest form, is an operation on two functions $f(x)$ and $g(x)$ that produces a third function, $y(x)$, that shows how the shape of one is modified by the other. A convolution operation can then be described as:

$$y(x) = \int f(x)g(x)dx \tag{2.4}$$

In literature, the convolution operation is often denoted with an asterisk:

$$y(x) = (x * y) \tag{2.5}$$

The first argument in the convolution is normally treated as the input, while the second is referred to as the kernel. The output is often called a feature map.

The activation map A is produced by convolving the kernel K over the input I and computing the dot product between K and the current location of K in I.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.6}$$

When working with images we often apply a set of kernels (or filters) in order to extract features from the image. So in terms of the operation explained above the input is an image and the kernel is a set of weights that when applied can extract features like for example vertical or horizontal edges.

-1	0	1
-2	0	2
-1	0	1

(a) Vertical

-1	-2	-1
0	0	0
1	2	1

(b) Horizontal

Figure 2.9: Sobel edge detection kernels of size 3x3.

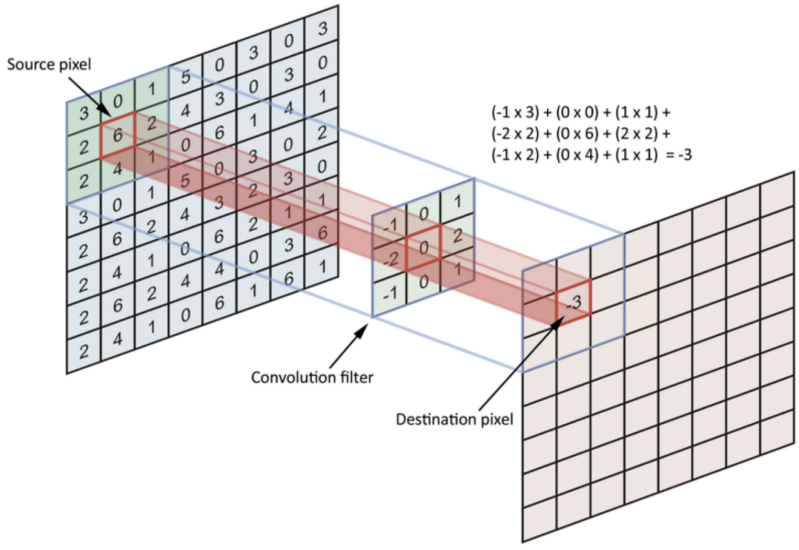


Figure 2.10: Shows the first step in a convolution process. Figure adapted from [7].

Figure2.9 shows kernels that extract horizontal and vertical edges. Figure2.10 shows the first step in a convolution operation using the vertical kernel. The filter is applied iteratively to every location in the image. Because the resulting convolution is placed in the center of the kernel, the resulting image resolution is reduced. When convolving there are two parameters that can be used to control the output of each layer. These parameters are called stride and padding. The stride controls how many units the kernel convolves around the input image, that is to say how many units does it move in each step. By increasing the stride the output volume shrinks. Padding, or zero-padding, is a process where we simply append a border of zeroes

around the image, making the resulting output maintain its original size. When convolutions are applied to inputs with multiple channels such as RGB-images, each of the channels are convolved with its own kernel producing its own output, which is then merged to produce the final output array. Figure 2.11 shows an example of resulting images after applying two sobel filters on a input image.

(a) Original image

(b) Vertical sobel applied

(c) Horizontal sobel applied

Figure 2.11: Shows resulting features after applying horizontal and vertical sobel filter on the input image, images from [8].

2.3.4.2 Motivation for CNN

One of the main problems with regular neural networks is that they scale poorly when larger images are introduced. When working with small images a normal neural network can perform well. Take the MNIST dataset as an example. The MNIST dataset consists of 70 000 examples of handwritten digits of 28x28 pixels with one color channel. This means that a single fully-connected neuron in a first hidden layer would have $28 * 28 = 784$ weights. Now consider a problem that require processing of an image of 256*256 pixels with 3 color channels. A single neuron in this network would require $256 * 256 * 3 = 196,608$ weights. Furthermore such a network would require several neurons which quickly becomes unmanageable.

Sparse connectivity, parameter sharing, equivariant representations, enables us to work with images of variable size.

Sparse connectivity - Traditional ANN layers use matrix multiplication where a matrix of parameters and a separate parameter describing the interaction between each input and

output unit is used. This means that every output needs to interact with every input. A CNN have what is called sparse connections. Having a kernel smaller than the input makes this possible. An input image could potentially have millions of pixels. By using a small kernel that only occupies i.e ten pixels it is possible to detect meaningful horizontal or vertical edges (among many other properties). This approach enables us to have a network with fewer parameters, which in turn reduces memory requirements for the model. Sparse connectivity can greatly improve the efficiency of the network. Lets say we have *m* inputs and *n* outputs, an ANN would need $m * n$ parameters. For a CNN, where the number of connections each output is limited to *k*, the network requires only $k * n$ parameters. See Figure 2.12.

Parameter sharing simply means that the network is able to use the same parameter for multiple functions. In a ANN, each member of the weight matrix is used only once when calculating the output of a layer. For the CNN, each member of the kernel is used at every position of the input.

Equivariance means that if some input change, the output change in the same way. A function is equivariant if $f(g(x)) = g(f(x))$.

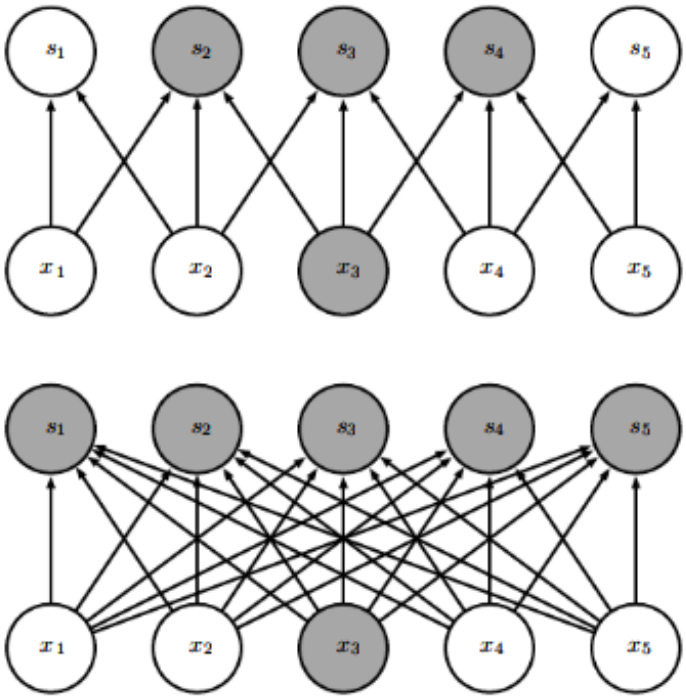
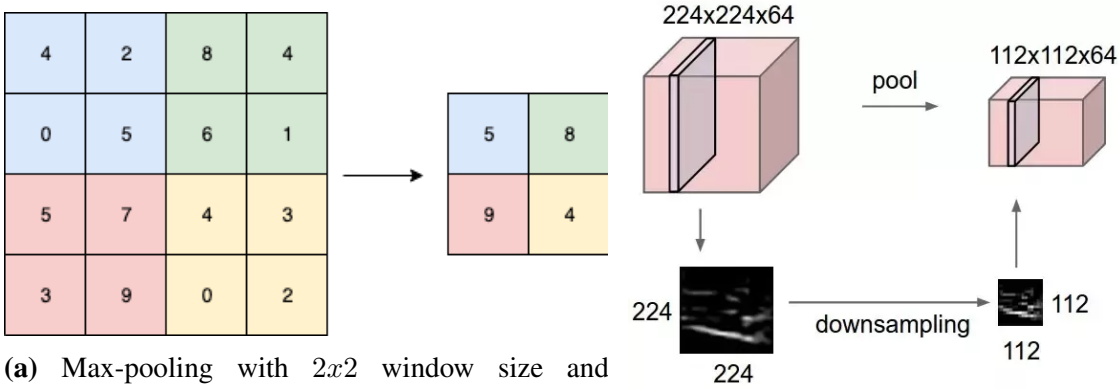


Figure 2.12: The top image shows a convolution on input $x3$ with a kernel width of 3. We can see that only 3 of the outputs are affected. In contrast, the bottom image shows that every output is affected when using matrix multiplication (ANN). Figure adapted from [3].

2.3.4.3 Pooling

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. Several statistic functions can be used, for example average-pooling

and min-pooling, but in deep learning architectures max-pooling is by far the most common. Max-pooling works by applying a max filter to the input array. In a non-overlapping way, the maximum value inside the region that the filter covers is used as an element in the output matrix, see Figure 2.13 for an example showing the details. Adding pooling gives convolutional networks *in-variance to translation*, which means that if we translate the input by a small amount the value of most pooled outputs do not change. Another benefit of pooling is that it improves computational efficiency because the layer after a pooling function has k times fewer inputs to process, making both network training and inference faster. Spatial information is lost in the pooling process but this is not a problem for tasks like classification where we care more about whether a feature is present or not, than exactly where it is. However, pooling can cause problems when solving tasks where localization of features is important such as the semantic segmentation task. Fortunately there are techniques to deal with this that we will discuss in Section 2.4.4.



(a) Max-pooling with $2x2$ window size and $stride = 2$.

(b) Pooling applied to a 64-channeled input array.

Figure 2.13: Illustration of pooling. Figures adapted from [9].

2.3.4.4 Variants of the Convolution Function

Transposed Convolution

A transposed convolution is like a regular convolution, but it reverts the spatial transformation such that the resulting matrix is of higher dimensions than the input. Transposed convolutions are used to up-sample images which is useful because many models involves up-sampling from a low resolution to high resolution. Up-sampling is traditionally done with bi-linear or nearest neighbour interpolation, but this is like manual feature engineering since there are no parameters for the network to learn. Using transposed convolutions instead makes the network able to learn how to up-sample directly from training data in one step, and avoids the need to choose a predefined interpolation method when constructing network architectures. A transposed convolution is in some literature called a deconvolution, this is somewhat unfortunate since the operation does not reverse the effect of a convolution, nor is it the opposite operation. A side effect of transposed convolutions is checkerboard artifacts in the up-sampled image, but this is mainly a problem for generative models [18].

1x1 Convolution

There is no difference between a regular convolution and a 1x1 convolution, and at first it might

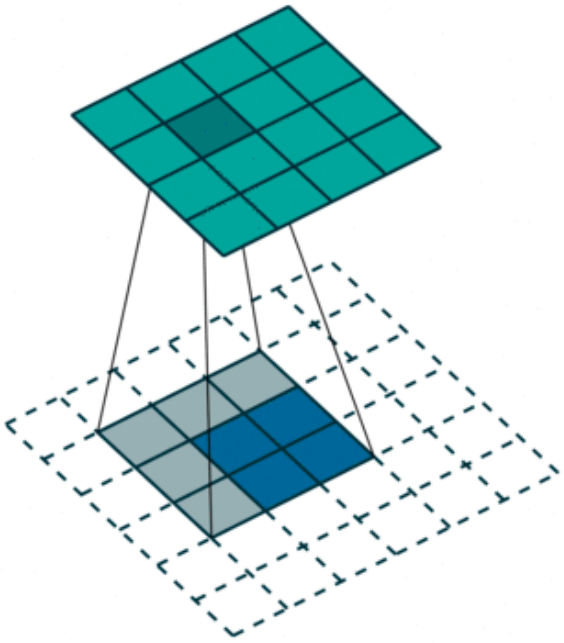


Figure 2.14: Transposed Convolution; blue is input, green is output, figure adapted from [7].

seem rather useless. All a 1x1 convolution is doing is multiplying the input array with a number before applying a non-linearity like ReLU, but this is only the case when the input array has one channel. When operating on input with multiple channels, a 1x1 convolution acts like a fully connected network within the convolution network over a patch. This idea of a network inside a network was first described in [19] and is highly used in the famous GoogleNet architecture. 1x1 convolutions allows us to down-sample feature maps in a way that summarizes them, which is useful because feature maps tend to increase with network depth. A 32x32x100 image convolved with a 1x1x20 filter will produce a 32x32x20 output, thus shrinking the amount of channels. Just like pooling reduces the height and width of an image, 1x1 convolutions can be used to reduce the depth which improves computational efficiency.

1D and 3D Generalizations
Convolutions are not restricted to only 2-dimensional data. The same concept can be applied in just 1-dimension, to find patterns in sequential data, and even to 3-dimensions to process 3d images which is common when analyzing medical images produced by CT-scans or MRI machines. When convolutions are applied to 3d images, a 3-dimensional kernel is slid across an input array with dimensions $width \times height \times depth$. This allows the kernel to extract patterns across all spatial dimensions, which in theory will produce better results than when applying 2d kernels which are slid across the width and height dimension for every depth slice. However, 3d convolutions is unfortunately in many cases not feasible to use because of high computational cost and huge memory requirements.

2.3.4.5 Transfer Learning

Transfer learning is a common and well documented approach to use when you are trying to solve new tasks with deep learning. Instead of training weights from random initializing you can make much faster progress by using transferring knowledge from one task to another. Some

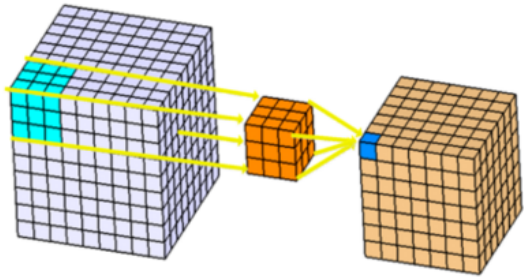


Figure 2.15: Visualization of how an 3d kernel is applied to an 3d input array, Figure from Kaggle notebook [10].

deep convolutional networks can take several weeks to train from scratch. Many datasets are also not of sufficient size to train independently and in order to train such datasets successfully you have to rely on transfer learning.

In all the different disciplines and applications of deep learning, computer vision and conditional networks, transfer learning should be considered unless you have access to a very large dataset.

When working with computer vision, there are numerous sets of pre-trained weights available. There is published a lot of work on the ImageNet dataset, which consists of over 1 million labeled images and has had yearly challenges [20]. Many different architectures have been implemented to try to outperform its competitors, making an abundance of available sets of pre-trained weights. Even when the task you are trying to solve is different from the source of pre-trained weights, early layers looking for low level features such as edges, contours and corners can still benefit from transfer learning.

2.4 Related Work

This section presents the literature review protocol used for searching and selecting relevant literature, gives some historical context to semantic segmentation and summarizes relevant previous work.

2.4.1 Structured Literature Review Protocol

The purpose of a structured literature review (SLR) is to assist researchers in searching and selecting relevant literature. The SLR provides a strict framework for how to identify, evaluate and interpret available research. It is important for researchers to follow the SLR strictly to reduce biases that might come from previous expectations. The SLR protocol consists of three main stages: planning, conducting and reporting [21].

2.4.1.1 Planning

The objective of the SLR is to map what state-of-the-art is in computer vision, specifically for semantic segmentation. It should address and find available literature that includes semantic segmentation or digital rock data.

The first step in conducting a SLR is identifying literature research questions(LRQ).

LRQ1 *What are state of the art deep learning algorithms for semantic segmentation?*

LRQ2 *What techniques exists for using deep learning for digital rock interpretation?*

LRQ3 *How does different CNN architectures compare for semantic segmentation?*

2.4.1.2 Conducting

The second step in SLR is defining the search strategy. The goal of conducting the SLR is to find primary studies that address the above literature research questions as much as possible. Conducting the literature search is an iterative process where search words are refined as new knowledge is acquired. Search words are created by breaking down the literature research questions into individual facets. A list can then be created of these facets that contain synonyms, abbreviations and alternative spellings. Table 2.1 shows the final selection of search words. When searching for primary studies at least one keyword from group 1 and 2 should be combined with a keyword from group 3 or 4 .

Search Strategy

After performing trial searches on a number of online scholarly websites it was decided that future literature searches should be performed on three search engines:

1. IEEE Xplore
2. Google Scholar
3. Springer Link

Additionally we employ the "*snowballing approach*", in which we follow references in papers to locate other relevant ones. This works well because one paper may only briefly introduce a concept taken from another publication.

	Group 1	**Group 2**	**Group 3**	**Group 4**
Term 1	Deep Learning	Computer Vision	3D - Data	Digital Rocks
Term 2	Machine Learning	Object Detection	CT - Scan	Rock properties
Term 3	CNN	Semantic Segmentation	Medical Imaging	
Term 4	ANN			

Table 2.1: Search Words

Selection and Quality Assessment

To help decide if a primary study is relevant and worth committing to, the material should comply with the inclusion and quality criteria. These criteria help build the final list of included literature. Table 2.2 contains the list of quality criteria (QC) and inclusion criteria (IC).

Through a three stage screening the criteria can be applied:

a abstract inclusion criteria screening

b full text inclusion criteria screening

c full text quality criteria screening

Criteria ID	Criteria
IC1	The study mainly address at least one of the search word groups
IC2	The study relates to one of the literature research questions
QC1	There are clearly relevant methods and results in the study
QC2	The algorithm design is justified and reproducible
QC3	The abstract should clearly state the goal of the paper
QC4	Findings should be clear in the conclusion
QC5	The study/paper has more than 10 citations
QC6	Results are clearly presented and supported

Table 2.2: Inclusion and quality criteria

2.4.1.3 Reporting

The final step of the SLR is producing the resulting literature list. Table 2.3 contains the final list of selected papers and articles.

ID	Title	Authors	Year
S001	U-net: Convolutional networks for biomedical image segmentation	O. Ronneberger, P. Fischer, T. Brox	2015
S002	Convolutional neural networks for automated seismic interpretation	A. Waldeland, A. Jensen, L. Gelius	2018
S003	Fully convolutional networks for semantic segmentation	J. Long, E. Shelhamer, T. Darrel	2015
S004	Deep Learning Convolutional Neural Networks to Predict Porous Media Properties	N. Alqahtani, R. Armstrong, T. Mostaghimi	2019
S005	Industrial applications of digital rock technology	C. Berg, O. Lopez, H. Berland	2017
S006	Segmentation of digital rock images using deep convolutional autoencoder networks	S. Karimpouli, P. Tahmasebi	2019
S007	The Effectiveness of Data Augmentation in Image Classification using Deep Learning	L. Perez, J. Wang	2017

Table 2.3: The final selection of included articles.

2.4.2 Historical Context of Semantic Segmentation

Most of the successful approaches to semantic segmentation before deep learning relied on hand crafted features, combined with traditional machine learning classification algorithms such as random forest and support vector machines. Performance suffered from limitations in the supplied features and was nowhere near that of today. However, in 2012 Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton entered a submission, a deep convolutional architecture named AlexNet to the Large Scale Visual Recognition Challenge (ILSVRC) [22]. Their submission reduced the existing top error rate in half and is considered to be the big deep learning breaktrough. AlexNet combined several ideas that would go on the become the backbone in modern deep learning architectures.

Convolutional Neural Network (CNN) algorithms are now dominating in visual recognition problems and this is also true in the case of semantic segmentation. Most algorithms can be categorized to belong in either one of two classes. The first class of algorithms builds upon the idea introduced by Ross Girshick et al. in the paper ”Rich feature hierarchies for accurate object detection and semantic segmentation” [23]. The main idea is to use region proposals from a decoupled segmentation algorithm as inputs to a deep convolutional network. The shape information provided by the proposals helps with the increasing loss of spatial information caused by the pooling operation in each consequent layer. This combination of classical tools from computer vision and deep learning outperformed earlier approaches. The only drawback is that the CNN can not recover from errors introduced by the extraction of proposals. The second class of algorithms are inspired by the fully convolutional network by Jonathan Long et al. [11], which showed that it is possible to create a CNN-architecture trained end-to-end that can produce state of the art segmentation. Fully convolutional networks enables the reuse of successful classification networks by replacing the the fully connected layers with convolutional layers. Then concatenating features extracted in earlier layers to provide spatial awareness to the classification process, without the need for any external segmentation algorithms to create proposals. In this thesis we will focus on and use algorithms that are based upon and/or inspired by fully convolutional networks to segment porosity in digital rocks.

2.4.3 Segmentation of Digital Rock Images Using Deep Convolutional Autoencoder Networks

In 2019 Karimpoulia and Tahmasebi [24] presented two approaches using a convolutional autoencoder network for segmenting digital rocks. Their dataset consist of only 20 pre-labeled images. In the first approach they generate more data by manually segmenting images and in the second they use a stochastic image generator (HYPPS). For the semi-automatic segmentation it is mentioned two issues, of which the first one is relevant to our task. That is, that grain boundaries are brighter than the surface, which makes the segmentation algorithm misclassify some boundaries. The second is that different minerals are classified as the same class, because of similarity in pixel-color values. To mitigate these problems they apply the watershed algorithm to enhance the segmentation process, but it did not resolve cases for complex structures. According to P. Acharjya the watershed algorithm suffers from some unwanted drawbacks. One is that images can become over segmented, meaning that some segments can arise inside other segments. The algorithm can also cause the image to be corrupted by Gussian noise [17].

In their second approach segmentation is performed using SegNet, which is similar to U-

Net, in that it based on a encoder-decoder architecture. The main difference is that SegNet uses fully connected layers which makes it computationally more expensive because it includes more parameters. The U-Net instead transfers the entire encoder which makes it more memory consuming. For this approach they use a *Hybrid Pattern-Pixel-based Simulation* (HYPPS), which is a data augmentation technique, to generate more training data. This method takes a dataset of labeled images and produces equiprobable realizations, which put more simply means they take regions from randomly chosen images in the set and stitch them together to produce new, unseen data. SegNet is able to achieve a categorical accuracy of 96%.

This paper was published in May of 2019, the research presented was prior to this unknown. It was later discovered, reviewed and added to our list of relevant literature given its significance.

2.4.4 Fully Convolutional Networks for Semantic Segmentation

The Fully Convolutional Network (FCN) [11] showed that it is possible to build an *end-to-end* deep learning architecture by extending successful classification networks, like VGG-16 and GoogLeNet to produce segmentation maps from arbitrary input sizes and achieve state of the art performance without any region proposals. The FCN architecture is like its name implies fully convolutional. This means it only contains locally connected and no dense layers, which has a fixed number of learned weights to work with such that varying inputs would require a varying number of weights. The absence of such layers in the FCN architecture enables training and inference on images of arbitrary input sizes. The architecture is divided into two main parts. Firstly, the **down-sampling** path which purpose is to extract features and then interpret the context (what is in the image) in order to make class predictions. Secondly, the **up-sampling** path that recovers spatial information lost in the down-sampling path in order to enable precise localization (where in the image) and produce a segmentation map with dimensions equal to that of the input image. Between the down-sampling and up-sampling paths there are several skip connections (connections in the network graph that bypasses at least one layer). The skip connections are used to concatenate the spatial information that exists in the earlier feature maps, gradually lost by the consecutive pooling operations, with features in the up-sampling path. This is done in order to improve the quality of the predicted segmentation map. The re-purposing of classification networks for semantic segmentation works because the down-sampling path can essentially be any classification network. This is demonstrated in the paper by using the famous GoogLeNet and VGG-16 architectures, which is important because it allows the FCN to directly benefit from progress made on classification by always using state of the art networks.

It might sound strange to even include pooling operations in architectures used for segmentation, because it causes loss of spatial information by reducing the input resolution. This is not a issue when doing classification because all we really care about is what the image contains, not where it is. The fact is that the segmentation network would perform better without them, however removing them is not a feasible option since preserving the input image dimension throughout the entire network would be to computationally expensive. Today such architectures are known as **encoder-decoder** architectures and the success of the fully convolutional network spawned many similar architectures.

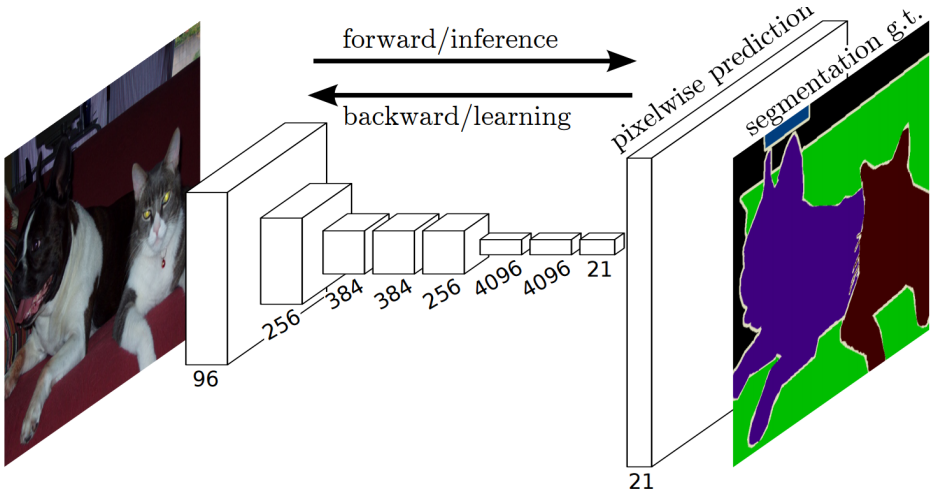


Figure 2.16: The Fully Convolutional Network (FCN) architecture. Figure from paper [11].

2.4.5 U-Net: Convolutional Networks for Biomedical Image Segmentation

In 2015 Ronneberger et al.[12] proposed the U-Net architecture for biomedical image segmentation. This architecture builds upon the fully convolutional network. However it modifies and extends it to allow for fast segmentation of neuronal structures in EM stacks with very few training images available. The biggest modification is that in the up-sampling part, in addition to using transposed convolutions for up-sampling, has a large number of feature channels. This makes it symmetrical to the down-sampling path, enabling the network to learn to assemble more precise outputs and giving it a U-shape. The way these additional feature channels are added is by applying two consecutive 3x3 convolutions for every concatenation with features from the down-sampling path. Another important concept presented in the paper is the overlap-tiling strategy. Since the network only uses the valid part of each convolution, there are missing pixels in the border regions. The overlap-tiling strategy extrapolates this missing context by mirroring the input image, yielding more accurate predictions. These ideas combined with excessive use of data augmentation gave the authors first place in the 2015 ISBI cell tracking challenge. The U-Net paper is considered very influential, and the architecture it proposes is thoroughly documented and tested on many different domains.

2.4.6 Deep Residual Learning for Image Recognition

When the depth of neural networks increase, one would expect in a best case scenario that more layers would yield better approximations of the function mapping we are trying to learn, thus reducing the error even more. In a worst case scenario it is expected that the first layers would act as a shallower network, while the remaining layers would learn the network identity function (learning the set of parameters such that input is equal to output). This is however not the case. Instead it has been shown that when depth of networks increase, the accuracy first get saturated before it starts degrading rapidly [25]. Intuitively one might think that the reason for this is overfitting, more layers do increase the capacity of the model, but is in reality caused by what is known as the *vanishing gradient problem*. With increasing depth, the gradients of the loss function approaches zero during back-propagation. This makes the network hard to train

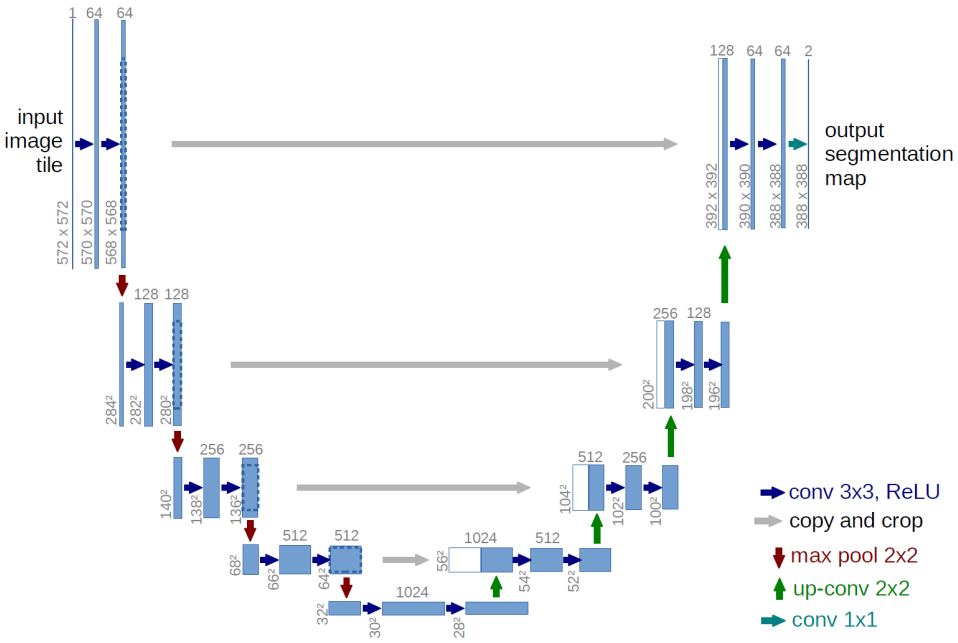


Figure 2.17: U-Net architecture, figure from original paper [12].

and leads to the inability to learn simple identity functions. Some activation functions like the Sigmoid can also contribute to this problem since its derivatives become very small.

An architecture that counteracts this problem is the residual network (ResNet) proposed by He, Kaiming et. al in the paper *Deep Residual Learning for Image Recognition* [26]. ResNet is built up of stacked residual blocks shown in Figure 2.18, that contains a short skip-connection that bypasses two weight layers and a ReLU activation, with the intuition that it is easier to learn parameters that makes $F(x) = 0$ and $y = x$ than learning the direct mapping $F(x) = y$. This allows the block to learn an identity function by relying solely on the skip-connection. Larger gradients can also be propagated back to earlier layers making deep networks consisting of residual blocks much easier to train.

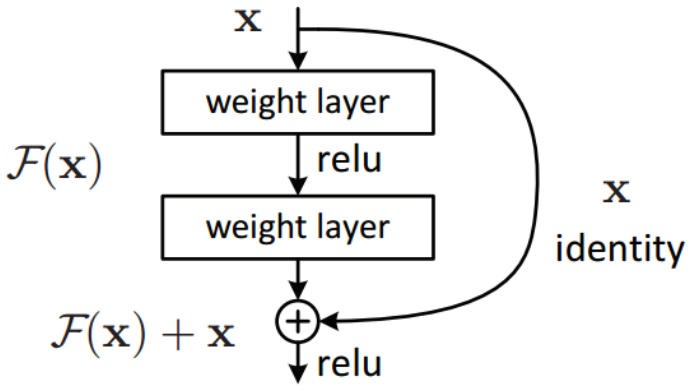


Figure 2.18: Single Residual Block, a building block. Figure adapted from [13].

To test their architecture [26] trained deep residual networks of 18, 34, 50, 101 and 151 layers on the ImageNet dataset and showed that each of the residual networks converged faster

compared to their plain counter parts. The 34 deep model archived a top-5 validation error of 3.57% winning the 2015 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [20]. Even tough the residual block was developed as a building block in an architecture that tries to solve a classification task, it is used to better enable training of various deep convolutional architectures aimed at other tasks. ResNet is relevant to us since the encoder part of a fully convolution network can essentially be the convolutional part of any classification network. Note that the **short** skip-connections used in residual networks should not be mistaken for the long skip connections used in encoder-decoder architectures. These connections serves another purpose, discussed in Section 2.4.4.

2.4.7 Segmentation of Coronary Arteries from CT-scans of the Heart Using Deep Learning

Kjerland 2017 [27], is a master thesis from NTNU that researched the use of a 3D convolutional neural network for segmenting 3D image volumes in three domains: coronary artery segmentation, brain tumour segmentation and digital rock segmentation. All segmentation tasks presented were trained by a 3D CNN architecture called DeepMedic. The DeepMedic architecture consists of 11 layers and use two paths working on different scales of input. Of the tasks presented in the thesis, the results of the digital rock segmentation is of most relevance to us. Because of memory constraints the rock dataset was split into subvolumes of 175x175x175 voxels. Three different models for segmentation is presented, one trained only on Bentheimer, one on Bentheimer and Berea, and one trained on Bentheimer, Berea and Carbonate. The networks perform well on categorizing grain and pore, with a mean dice score of 0.99 and 0.93 respectively, but struggles with multi-phase which only achieves 0.7. Below we list the most important findings when segmentation on each rock type on the different trained models are run.

1. Bentheimer
   (a) The network performs best when the network is solely trained on bentheiemr sandstone.
   (b) Second best is when the network is trained on Bentheimer and Berea.
   (c) Worst performance is seen when the network is trained on Bentheimer, Berea and Carbonate.
2. Berea
   (a) The network performs best when trained on Bentheimer and Berea.
   (b) The network performs worst when only trained on Bentheimer.
3. Carbonate
   (a) Carbonate struggles with multi phase segmentation when Carbonate itself was not included in network.

Furthermore, the results indicate that Berea is best able to learn features from bentheiemer while Carbonate is unable to utilize features learned from Bentheimer or Berea to any meaningful extent. Kjerland suggested in Section 6.2.3, Future Work, Digital Rock Segmentation: "*When*

training networks on different types of rocks there are several combinations that were not tested. These permutations should be tested to see if there are some types rock segmentation that benefits from being trained on multiple types.". We later present our experiments where Carbonate and Berea is tested against a network trained on itself and the other two, to see if Berea and Carbonate is able to better extract features from the other rock types than Bentheimer (which had best score when run on the network only trained on itself).

Chapter 3

Methodology

This chapter presents the Digital Rocks dataset and its creation. Then it details three different architectures for pore segmentation on this dataset. Finally, it discuss evaluation metrics used to compare our models.

3.1 Digital Rocks Dataset

The rock data that was supplied by Petricore was captured using micrometer computed tomography (micro-CT) imaging techniques on core samples of Bentheimer, Berea and Carbonate sandstone. See Figure 3.2 for pictures of the actual rock types. Micro-CT is used to characterize the 3D structures at the micrometer scales, which is 20-200 times more detailed than regular CT. The data supplied for use in this thesis are captured at a resolution between 3 and 17 microns, and at this scale one can reveal geometric properties and characterize the pore-space of a rock sample to calculate porosity and permeability. In the labeled/segmented volume each voxel is discretized into belonging to one out of three classes, pore, multi-phase and grain, where each class corresponds to some porosity level. Figure 3.1 shows the class distribution for each sandstone sample.

The data for each sandstone is contained in its own raw-file specifying a 3D-cube, with a corresponding raw-file containing the segmented ground truth. The Bentheimer micro-CT cube has a resolution of 1370x1370x1701 pixels, with a 16 bit depth. The corresponding file containing the segmentation has the same resolution, but with a 8 bit depth. Combined, this equals approximately 8.9 GB of data. Berea has a resolution of 1000x1000x2399 pixels = 6.75 GB. Carbonate has a resolution of 1500*1500*3000 pixels = 18.9 GB. This gives the rock dataset a total size of 34.6 GB, and due to GPU memory constraints, it is not possible to operate on a single rock volume simultaneously. This is not a problem for networks that uses 2D convolutional kernels to predict the segmentation map for a single slice of this volume at a time. Architectures based on 3D convolutions would have to divide the original volume into non-overlapping sub-volumes small enough to fit in memory. Figure 3.3 shows four slices from the Bentheimer, Berea and Carbonate volume.

Although 34.6 GB might seem like a lot of data, it is only 7100 depth slices, most computer vision tasks would benefit from more data. Acquiring more micro-CT images of core samples and labeling them is both expensive and time consuming. To increase the diversity and amount of data we employed several data augmentation techniques. To avoid a huge increase in dataset

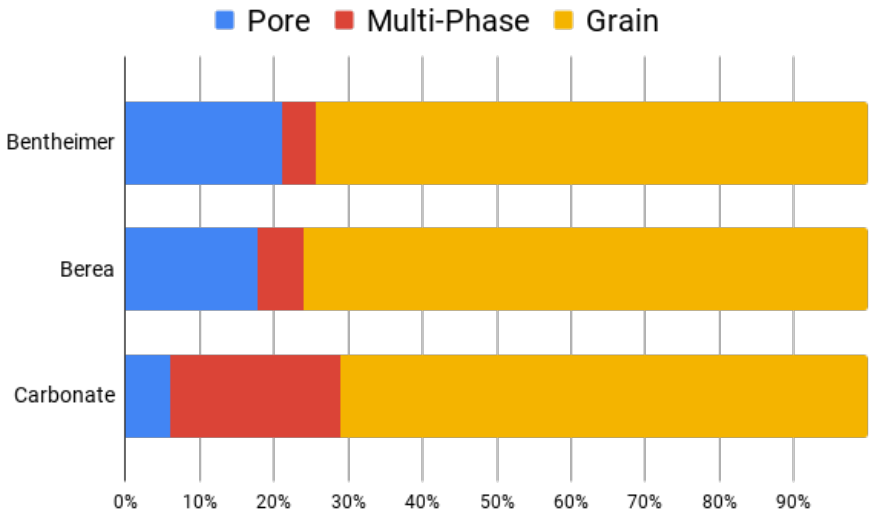


Figure 3.1: Shows the class distribution for the different sandstone types, Bentheimer and Berea has a very similar distribution, Carbonate differs slightly.

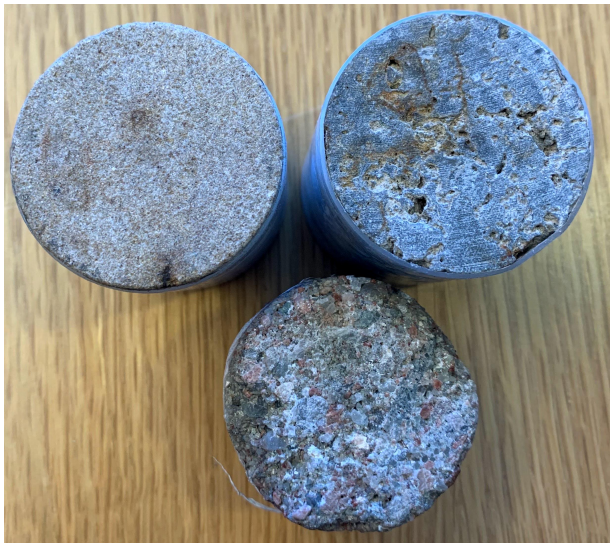
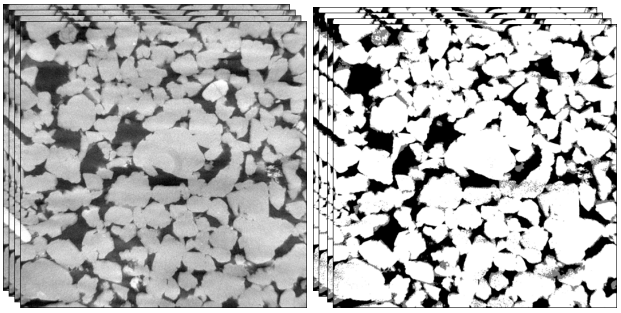
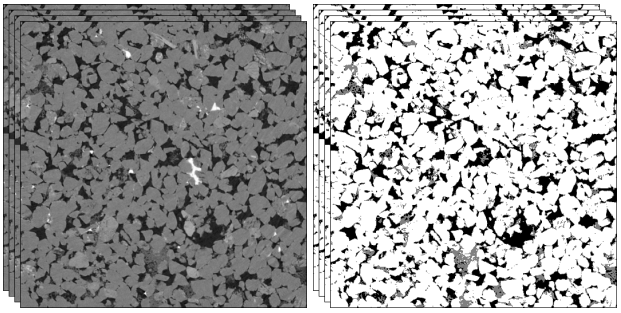


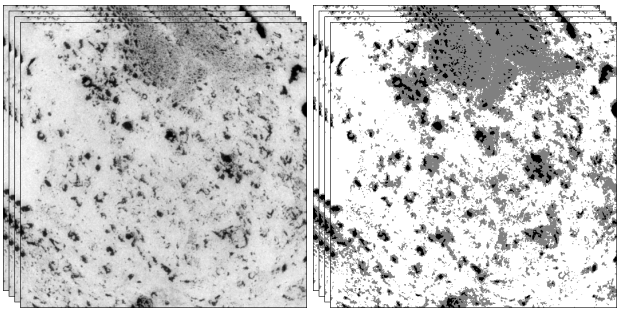
Figure 3.2: Core samples of sandstone, Berea (left), Bentheimer (bottom) and Carbonate (right).



(a) Bentheimer



(b) Berea



(c) Carbonate

Figure 3.3: Four slices from each sandstone volume. Left image shows the Micro-CT, to the right the corresponding segmented ground truth. In the ground truth slices black corresponds to pore, grey to multi-phase and white to grain.

size all transformations are applied directly to the mini batches that are feed to the network. The exact transformations are described in Section 4.2.2.

3.1.1 Generation of Ground Truth

The ground truth porosity segmentation provided by the digital rocks dataset was generated by combining the segmentation produced by five different users, configuring a multi-thresholding algorithm with parameters that they believed to be most suitable. The way these segments was combined was by taking the voxel-by-voxel most common class among all users, yielding a segmented average considered ground truth. Table 3.1 shows the intersection over union (IoU) between each user and the segmented average. This shows how much a segmentation differs depending on different users criteria, and allows us to compare that with a deep learning model that always produces the same segmentation.

	Bentheimer	**Berea**	**Carbonate**
User 1	0.976	0.967	0.826
User 2	0.918	0.905	0.911
User 3	0.942	0.960	0.692
User 4	NaN	0.956	NaN
User 5	0.957	0.924	0.925

Table 3.1: Intersection over Union between each user and the segmented average, and shows us that Carbonate gives users the most different results (a data parsing issues resulted in some missing values for user 4).

3.2 Developing a Digital Rock Segmentation Network

When selecting a base network architecture to extend upon to build a porosity segmentation network, several factors was considered: expected performance, model complexity and availability of pre-trained weights. The two dominant types of networks used for segmentation in computer vision literature are region proposal based networks and fully convolutional networks. Region proposals are mainly used for instance segmentation and are more computational expensive. We chose to proceed with the fully convolutional network, as this architecture is heavily documented with many variants. Additionally, most frameworks have available open source implementations.

Although the dataset we are trying to segment is volumetric, we chose to keep the model complexity low by only using 2D and not 3D convolutional kernels. 2D kernels has far fewer parameters but are only able to leverage context across the height and width of an image. In order to segment a full volume, segmentation maps have to be predicted one depth slice at a time. We hypothesize that this is a good trade off between efficiency and accuracy, especially with limited resources. The following sections present three architectures developed for digital rock pore segmentation.

3.2.1 Digital Rock Network (DRNet)

Digital Rock network (DRNet) is an extension of the original U-Net architecture which is the most well documented FCN. The dataset Ronneberger et. al developed this architecture for resembles our dataset, making it a natural starting point. U-Net was developed in 2015, advancements has been made to the computer vision field since, that we incorporated into DRNet. The up-sampling path consists of five blocks that uses transposed convolution in place of bi-linear interpolation. Each block has batch normalization layers in between convolution operations and activation functions. The down-sampling path is a residual network, this allows for experimentation with different layer depths to find the appropriate capacity for the task and still have the opportunity to use transfer learning. ResNet-18 $\longrightarrow$ ResNet-152 all have available weights online. Figure 3.4 shows the full architecture of DRNet.

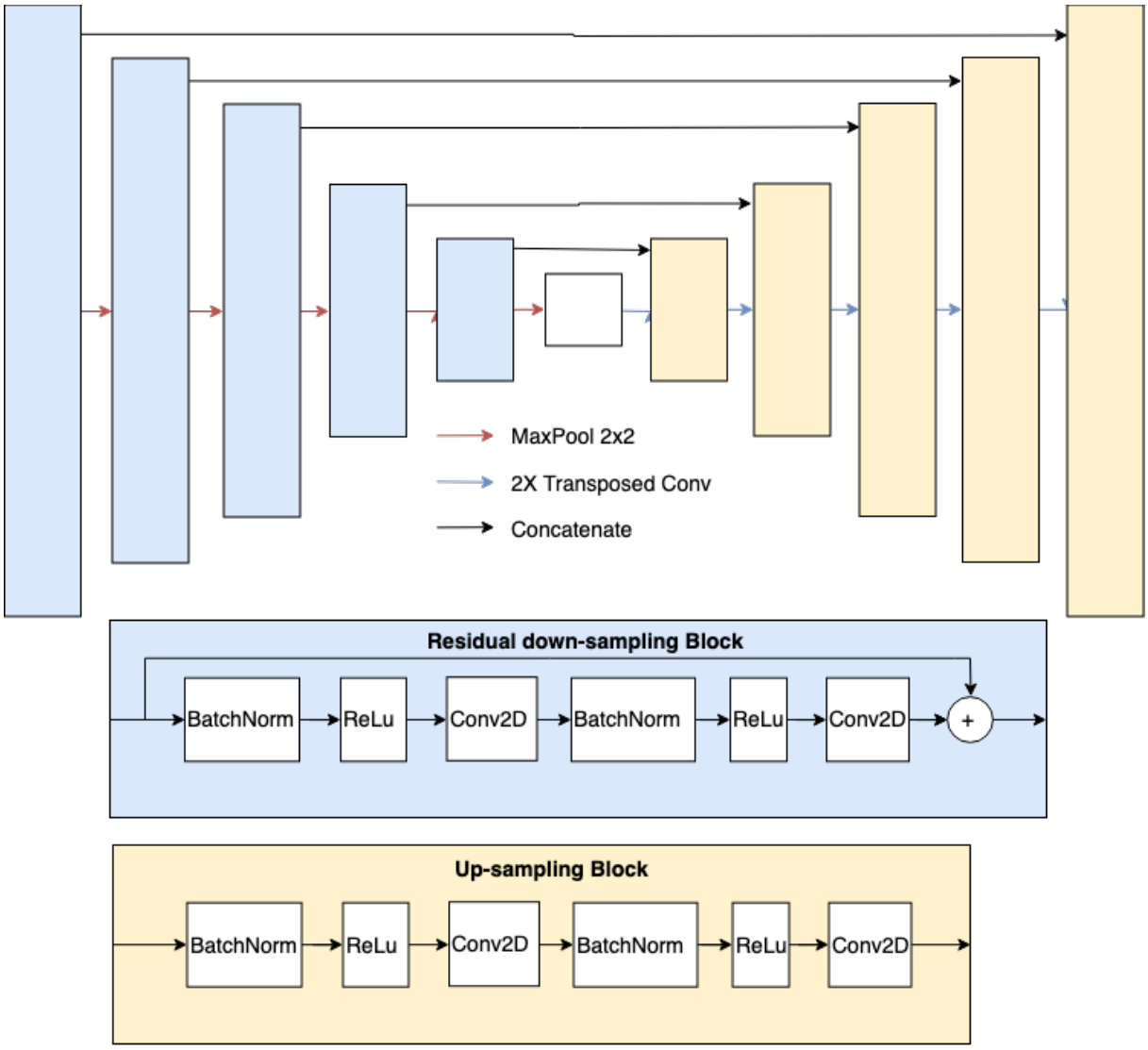


Figure 3.4: DRNet architecture, number of residual down-sampling blocks varies.

Given this encoder-decoder architecture there are several ways of implementing transfer learning. One possibility is to initialize the down-sampling path with pre-trained weights and

freeze them, and then only train the up-sampling weights. Weights in the down-sampling path never changes so saving the output of this network part to disk enables faster convergence, since there is no need to compute the activation of any node. This is only a good idea when the two datasets are similar, if not performance can be reduced. We were only able to find pre-trained weights on ImageNet and this dataset differs very from the digital rocks dataset. What we did instead was to only freeze the weights in the down-sampling path for two epochs. This was done order not to damage the properly trained weights with huge gradients during first steps of training because the loss is so large.

3.2.2 Pyramid Digital Rock Network (PDRNet)

PDRNet is based upon the Feature Pyramid Network (FPN) [28], a generic pyramid feature extractor initially proposed for object detection. It can however be extended to predict segmentation maps. The PDRNet architecture is similar to DRNet but there are some key differences. PDRNet conforms to a pyramid shape with a bottom-up and a top-down pathway, this is equivalent to the down-sampling and up-sampling paths in DRNet. However, while the long skip connections in DRNet directly copy feature maps before concatenation, PDRNet applies a 1x1 convolution to reduce channel dimensions before merging feature maps with element wise addition. Figure 3.5 shows the details.

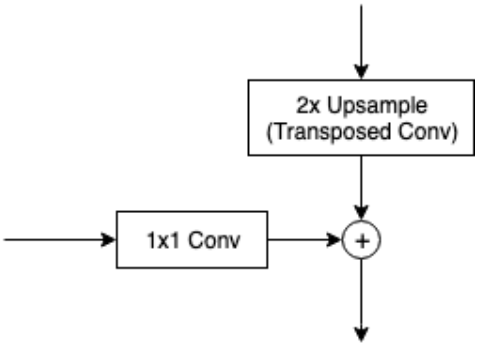


Figure 3.5: Lateral connection for PDRNet, one for each bottom-up and top-down block.

The biggest difference is that PDRNet use every block in the top-down pathway to predict segmentation maps. This yields predictions at different scales and a final segmentation is formed by merging these predictions. This should benefit small segments since the higher-resolution maps in the feature hierarchy are directly used for prediction. DRNet only predicts at the last stage.

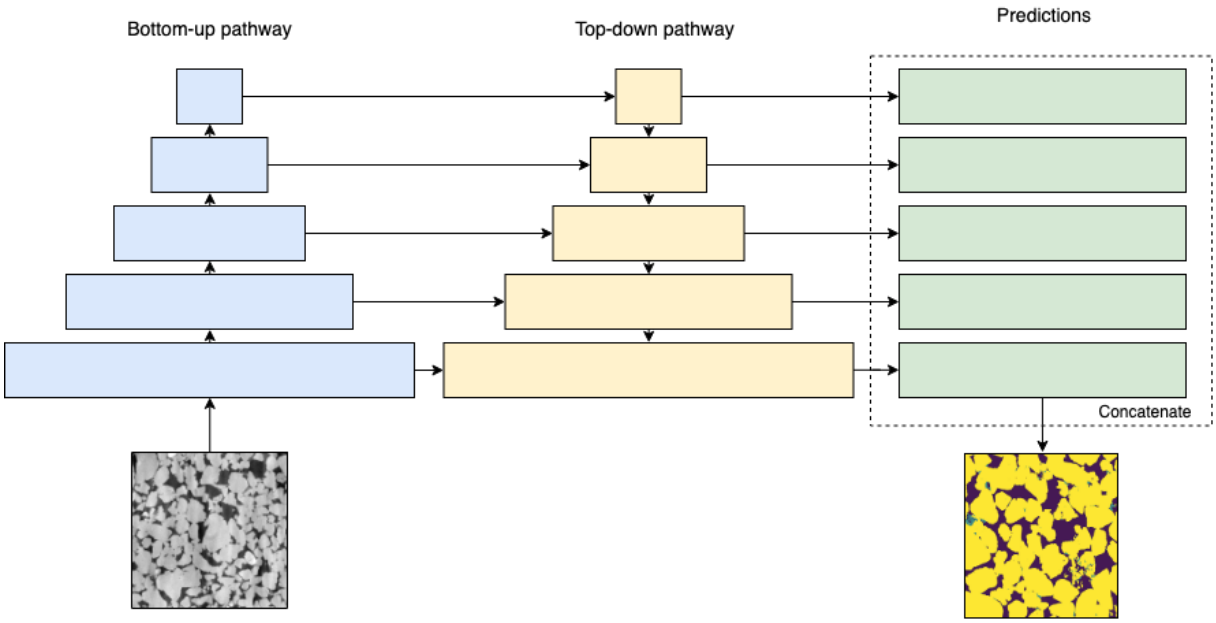


Figure 3.6: PDRNet high level architecture, details are left out to highlight differences with DRNet.

3.2.3 Hybrid Digital Rock Network (HDRNet)

The Hybrid Digital Rock Network (HDRNet) is an extension to DRNet and PDRNet that provides some depth slices as additional input channels. This allows the network to leverage context from adjacent slices. We hypothesize that this will boost performance without much additional computational cost.

An inherent problem with both DRNet and PDRNet is that they segment 3D volumes slice by slice, only considering neighbouring pixels in the width and height dimension. Depth information, although it is present in the data, is not utilized. One way to include this information in the prediction process is by replacing all 2D convolutions with 3D convolutions and train on 3-dimensional patches. 3D convolution has filters that moves in 3-directions (x, y, z). Every spatial dimension become equally important unless the model learns otherwise. Unfortunately, this has huge computational costs and it is only feasible to train on smaller subvolumes which again becomes limiting. Kjerland [27] trained DeepMedic, an 11-layer 3D-convolutional network on 120 175x175x175 voxel subvolumes of digital rock data. This only amounts to 0.065% of the total available data, and training converged in 8 hours. With HDRNet we propose a compromise by not using 3D-convolutions but instead include some of the adjacent slices to the current depth slice as input channels to the network. This increases memory requirements but can easily be tuned with varying the number of depth channels. Figure 3.7 shows a training input-output pair with 4-depth channels.

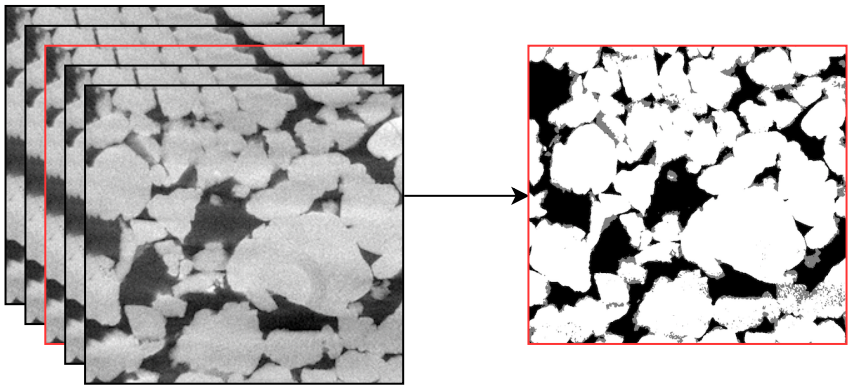


Figure 3.7: Visualization of input x and ground truth y used to train the hybrid model, two depth slices are added on each side of the current slice to segment (in red), as additional channels.

To make this work some architectural modifications are necessary. Employing transfer learning on networks trained on ImageNet or similar datasets with RGB-images force us to use 3-input channels, one for each color. This cannot change because the weights have been trained for this specific input configuration. Fortunately, there are ways to work around this. Training on grayscale images can be done by repeating the same image array 3 times on a new dimension. In our cases, where we have more than 3 channels we need a way to map N channels into 3. In HDRNet this accomplished by convolving a 1x1x3 kernel which takes the element wise product of the N depth channels and the filter before applying a ReLU non linearity. This acts like one layer in a fully connected neural network that learns a mapping between N and 3 channels.

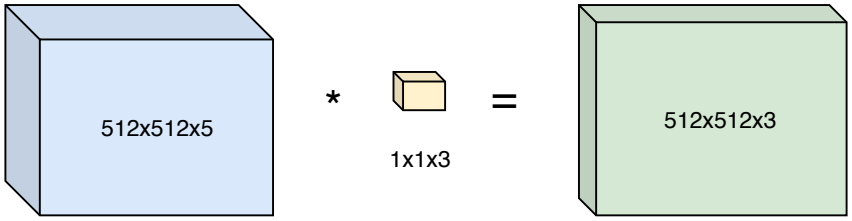


Figure 3.8: A 5-channeled input convolved with a 1x1x3 filter resulting in a 3-channeled output. Applying a non linearity (ReLU) to this output allows the network to learn a mapping that can utilizes the depth information available in the input.

3.3 Evaluation Metrics

In order to evaluate the performance of our models and compare them with existing research we need to choose appropriate metrics. Recall that the semantic segmentation task is to predict the class c_i of each pixel in a given image. But how do we evaluate how good a produced segmentation really is? There exists several metrics and the most commonly used is explained below. For each input image and predicted segmentation map, there is also a ground truth segmentation. The goal is to produce a segmentation as close to ground truth as possible. When comparing the class predicted for a pixel in the segmentation mask with the corresponding pixel in the ground truth mask, there are four possible outcomes listed below. Many existing metrics are expressed as combinations of these outcomes, but we will only cover the most relevant for evaluating semantic segmentation.

- True Positive (TP); the pixel is predicted to be of class c_i and the pixel is indeed a part of c_i in the ground truth mask.
- True negative (TN); the pixel is predicted to *not* be of class c_i and the pixel is *not* a part of c_i in the ground truth mask.
- False Positive (FP); the pixel is predicted to be a part of class c_i, but belongs to some other class c_j in the ground truth mask.
- False Negative (FN); the pixel is predicted to *not* be a part of class c_i but belongs to class c_i in the ground truth mask.

Accuracy is one of the simplest and most intuitive measures. In the semantic segmentation task it is equal to the percent of pixels that are correctly classified and commonly reported for each of the classes present in the image averaged across all classes. Accuracy is easy to understand and a widely adopted measure, but can give some misleading results when there exists large class imbalances in the image. Given a binary segmentation task with two classes c_1 and c_2 where c_2 only makes up 1% of the pixels in the given image, a classifier that produces a segmentation with only class c_1 will have an pixel accuracy of 99%. For this reason accuracy is not regraded as well suited for evaluating image segmentation since large class imbalances are very common in many domains.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

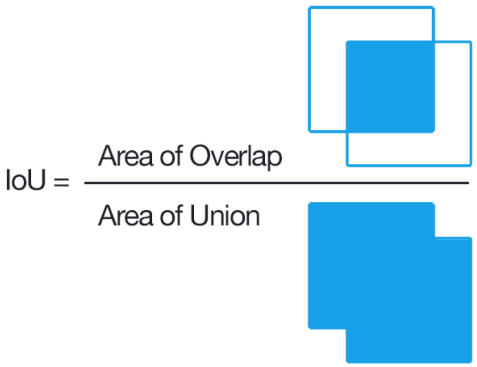


Figure 3.9: Intersection over Union.

Intersection over Union (IoU), also known as the jaccard similarity coefficient, quantifies the overlap between the predicted and the ground truth segmentation map. The result is equal to the number of pixels shared between the two segmentation masks, divided by present pixels across the two masks. IoU does not suffer from problems with large class imbalance. IoU is easier to understand visually than with a formula of TP, TN, FP and FN. IoU was selected as the main metric to judge the performance of models, because it is the best suited metric to evaluate the quality of segmentation along with the Dice coefficient. IoU is however most used in semantic segmentation challenges in addition to being the most intuitive of the two. Figure 3.9 shows a geometrical example of how IoU is calculated.

Dice Coefficient is another metric commonly used to evaluate segmentation maps and is closely related to IoU ($IoU = \frac{Dice}{2-Dice}$). Dice is hard to describe geometrically, thus it is expressed in terms of true positives, false negatives and false positives. The Dice coefficient was reported in addition to IoU, to make comparisons with related work that only reports Dice easier.

$$Dice = \frac{2TP}{(TP + FP) + (TP + FN)} \tag{3.2}$$

Confusion Matrix is not a metric in itself but an error matrix consisting true positives, that can give insights to which classes a model struggles with and often confuses with other classes. Can be normalized to show percentages

Chapter 4

Experiments and Results

This chapter starts by presenting the experimental plan. Then we talk about the experimental setup, including hardware, tools data pre-processing, augmentations and hyperparameters. Finally we list the results obtained from running the experiments.

4.1 Experimental Plan

The goal of our experimental plan is to help answer the research questions. The first step in our plan is to determine which of the two network architectures, PDRNet and DRNet, that perform best on the task of segmenting digital rocks. Our plan is to do this using the computer setup detailed in the next section and using the evaluation metrics presented in Section 3.3, in order to quantify performance. Next, the best performing architecture is extended with depth channels and named HDRNet as detailed in 3.2.3. This is done to see if providing just some depth context can increase performance, without a high increase in computational time. Finally, we train the best performing architecture multiple times with different subsets of the original dataset, containing only some sandstone types. This is to see if the rocks are somewhat drawn from the same probability distribution, such that models trained on subsets of rock types can generalize to others.

4.2 Experimental Setup

This section presents the setup used to conduct experiments. This includes the hardware, software, data prepossessing techniques and hyperparameters used for model training and evaluation, to ensure that our work is reproducible.

4.2.1 Hardware, Software and Environment

Due to the computational requirements needed to viably perform subsequent development and testing of different models, a dedicated computer was built in cooperation with Petricore with the following specifications.

- 2 x 11 GB Nvidia GeForce GTX 1080 Ti GPU
- 6 Core Intel Xeon E5-2650 Processor

- 64 GB 1600MHz DDR3 RAM
- 128 GB Samsung SSD
- CUDA Version 10.1
- Nvidia Version 418.56
- Tensorflow GPU 1.13.1

CUDA is a parallel computing platform and programming model developed by Nvidia for computing on a GPU. CUDA greatly increase computing speed by letting the compute intensive portion of the model runs on thousands of GPU cores and the sequential parts of the program runs on the CPU, which is optimized for single threaded performance.

TensorFlow is a library for numerical computation using data flow graphs that makes research, development and deployment of deep neural networks and machine learning models faster and easier. TensorFlow can run computations across a variety of hardware (CPU, GPU, TPU). It also includes TensorBoard, a data visualization toolkit which we used to both monitor and debug model training. TensorFlow was originally developed by researchers and engineers working on the Google Brain team within the Machine Intelligence Research organization at Google, but has since been open sourced [29].

Keras is a high-level neural network API that is "designed for human beings, not machines," and "follows best practices for reducing cognitive load." [30]. Keras does not run any computation, but can delegates it to several back-end engines, TensorFlow being one of them. Network models created for this thesis used the Keras functional API, which eases the development process and enables rapid experimentation while maintaining full flexibility to customize models.

Jupyter Notebook is an open-source web application that makes it easy to create and share documents that contain live code. Jupyter is well suited for use with machine learning where one of the the main benefits is the ability to hold datasets in memory between experiments. This optimizes development speed in that large datasets is loaded once and can then be reused.

4.2.2 Data Preprocessing and Augmentation

Of the original digital rocks dataset 80% of the data is used for training while the remaining 20% is split equally between the validation and test set. To ensure balanced and unbiased datasets, the different sandstones is distributed equally between all sets. Training and validation batches of images are produced with Python generators that reads image slices from disk and normalizes pixel values, before extracting random 512x512 patches from one slice of Bentheimer, Berea and Carbonate. This is done to reduce the memory requirements and avoid training with a extremely low batch size. Even tough the models are trained on smaller patches, they can still be used to predict segmentation maps on whole slices, because they are fully convolutional and not dependant on the width and height of the input array. After preprocessing, image augmentation is applied, but only to training batches with some probability p using a python library called Albumentations [31]. This library was chosen because it has been shown to be the fastest on most transformations, which is important when images are augmented in real time during training, because slow transformations will increase total training time. We experimented with

three different augmentation schemes shown in Table 4.1, 4.2 and 4.3, each transformation in a scheme has a probability of being applied. Some of the transformations is part of a set, where one of the members are chosen according to the specified probability, if no probability in these sets are given, the probability is distributed equally.

Augmentation	**Parameters**	**Probability**
Horizontal Flip		0.5
Random Brightness / Contrast	Brightness limit = 0.2 Contrast limit = 0.2	0.2
Shift, Scale, Rotate	Shift limit = 0.162 Scale limit = 0.6 Rotate limit = 0	0.7

Table 4.1: Weak Augmentation

Augmentation	**Parameters**	**Probability**
Horizontal Flip		0.5
Vertical Flip		0.5
Random Rotate 90		0.5
One Of [Elastic Transfrom, Grid Distortion Optical Distortion]	 p=0.5, alpha=120, sigma=120*0.05 p=0.5 p=1, distort_limit=2, shift_limit=0.5	0.8
Random Brightness Contrast		0.8
Random Gamma		0.8

Table 4.2: Medium Augmentation

Augmentation	**Parameters**	**Probability**
Flip		
Transpose		
Random Rotate 90		0.5
One Of [Additive Gaussian Noise, Gauusian Noise]		0.2
One Of [Motion Blur, Median Blur, Blur]	 p=0.2 blur_limit=3, p=0.1 blur_limit=3, p=0.1	0.2
Shift, Scale, Rotate	shift_limit=0.062, scale_limit=0.2, rotate_limit=45	0.2
One Of [Optical Distortion, Grid Distortion, Piecewise Affine]	 p=0.3 p=0.1 p=0.3	0.2
One Of [Sharpen, Emboss]		0.3
Hue Saturation		0.3

Table 4.3: Strong Augmentation

4.2.3 Hyperparameters

Training was performed using the RMSProp optimizer without decay on randomly sampled 512x512 pixels patches augmented using one of schemes presented in the previous section. Some hyperparameters were experimented with across models and are provided along with results, the common for all models are given in the list below.

- Learning rate = 0.001
- Epochs = 60
- Batch Size = 32
- Augmentation Probability = 0.5
- Early Stopping Patience = 10 epochs
- Reduce Learning Rate Patience = 7 epochs
- Reduce Learning Rate Factor = 0.25 (0.000001 minimum)

4.3 Experimental Results

This section presents the results obtained from running different experiments with the three different CNN networks. Chapter 5 discuss the implications of these results.

When reporting results we refer to IoU scores of the trained models. We chose to use IoU as the main metric because this is the metric most commonly used in other segmentation papers. The dice coefficient is also reported to make it easier to compare results with related work.

4.3.1 DRNet vs PDRNet

The results presented in Table 4.4 and Table 4.5, shows that DRNet outperforms PDRNet by a large margin across all tested hyperparameters.

Encoder	Frozen Epochs	Weights	Augmentation	IoU	Accuracy	Dice
ResNet18	0	ImageNet	Weak	**0.8638**	0.9736	0.9219
ResNet34	0	ImageNet	Weak	0.8637	0.9732	0.9216
ResNet50	2	ImageNet	Weak	0.8536	0.969	0.915
ResNet34	2	ImageNet	Weak	0.8516	0.9818	0.9103
ResNet34	0	None	Weak	0.8512	0.9695	0.9135
ResNet34	0	ImageNet	Medium	0.7605	0.9259	0.8477

Table 4.4: DRNet, test set results.

Encoder	Frozen Epochs	Weights	Augmentation	IoU	Accuracy	Dice
ResNet34	0	ImageNet	Weak	**0.7965**	0.946	0.8775
ResNet18	0	ImageNet	Weak	0.7782	0.9412	0.8642
ResNet34	2	ImageNet	Weak	0.7719	0.9412	0.8569
ResNet50	2	ImageNet	Weak	0.7694	0.9322	0.8594
ResNet34	0	None	Weak	0.7683	0.9398	0.8569
ResNet34	0	ImageNet	Medium	0.7056	0.8974	0.8078

Table 4.5: PDRNet, test set results.

4.3.2 HDRNet

We extend DRNet to include depth channels, Table 4.6 shows results obtained by HDRNet on the test set. Figure 4.1 compares training time and performance of this network, trained with different amounts of depth channels.

Encoder	Frozen Epochs	Weights	Aug.	Depth	IoU	Accuracy	Dice
ResNet34	2	ImageNet	Weak	6	**0.8860**	0.9779	0.935
ResNet34	2	ImageNet	Weak	4	0.8859	0.9792	0.9358
ResNet50	2	ImageNet	Weak	4	0.8802	0.9769	0.9319
ResNet34	0	ImageNet	Medium	4	0.8662	0.974	0.9225
ResNet34	2	ImageNet	Weak	2	0.8614	0.9753	0.9191
ResNet34	0	ImageNet	Weak	4	0.8404	0.9461	0.9063

Table 4.6: HDRNet, test set results.

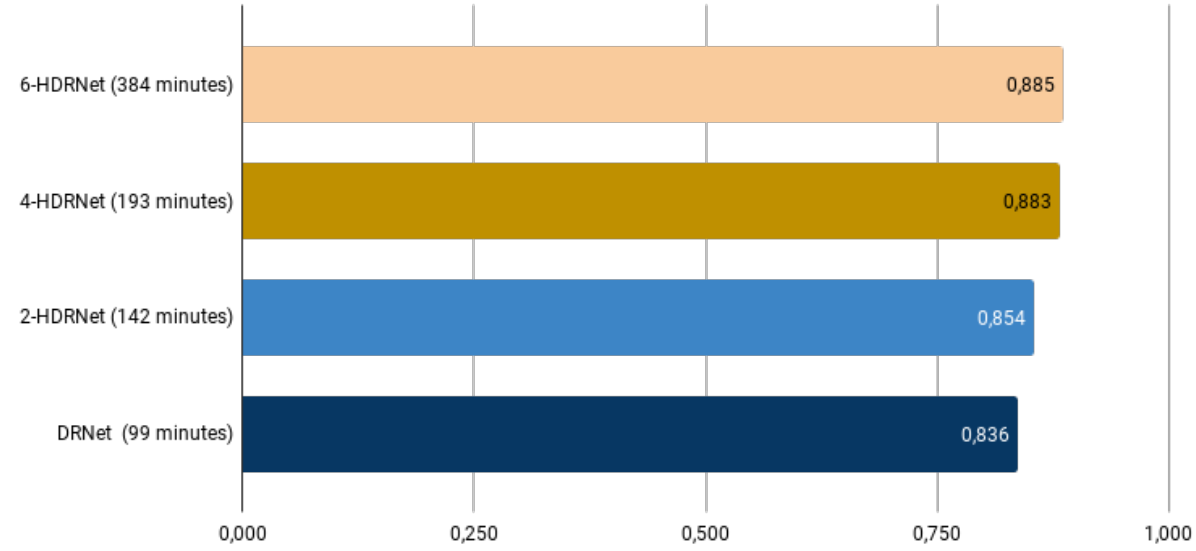
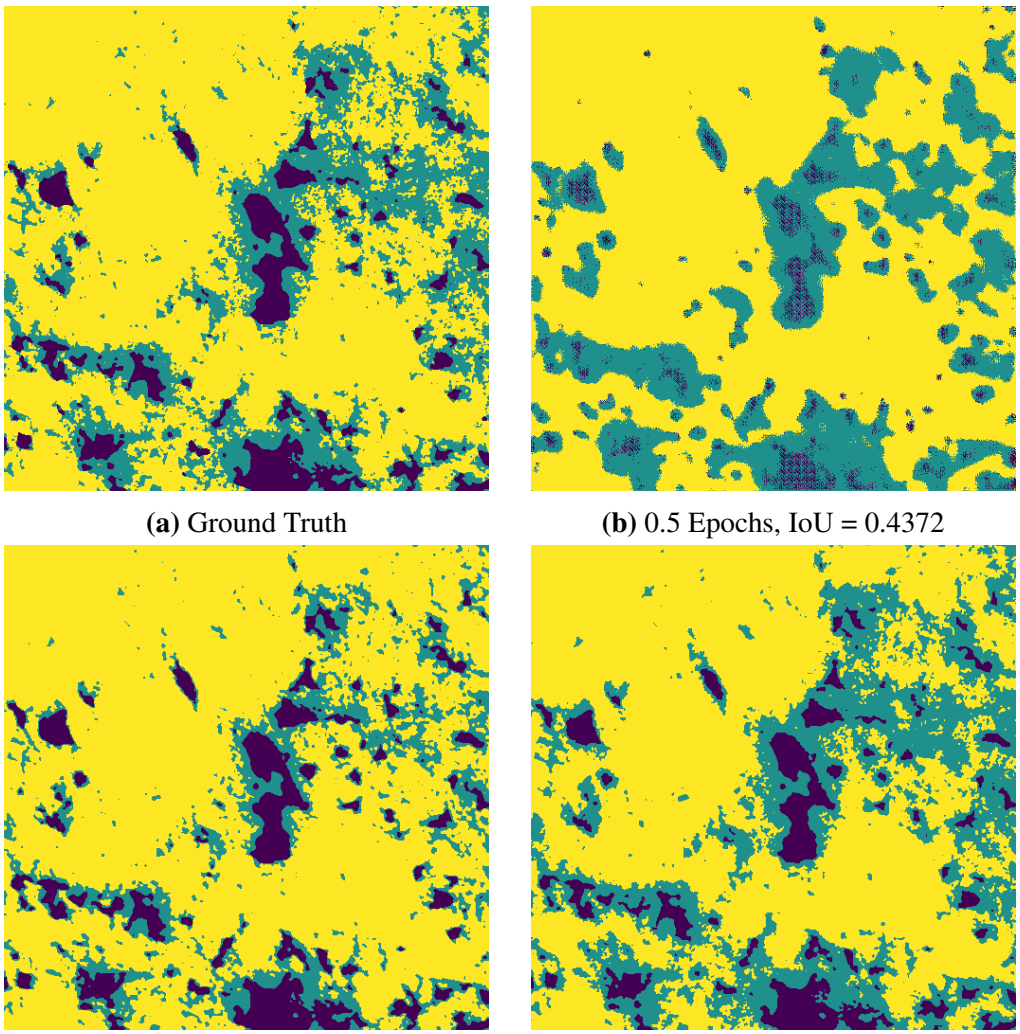


Figure 4.1: Validation IoU and training time for HDRNet with increasing amount of depth channels. Results show that performance increases with additional channels before it saturates, training time also increases quite a lot. We argue that HDRNet with 4 depth channels is a good compromise between performance and training time for this task. HDRNet with zero depth channels is equal to DRNet and provided as a baseline. Data extracted from Tensorboard and plotted manually.

4.3.3 Results on Different Types of Sandstone

Tables 4.7-4.13 presents the results for the experiments that were performed when networks were trained on different combinations of rock types. We choose to do this comparison using HDRNet with 4 depth channel, shown in previous experiments to be the best trade off between training time and performance.



(a) Ground Truth

(b) 0.5 Epochs, IoU = 0.4372

(c) 3 Epochs, IoU = 0.6921

(d) 30 Epochs, IoU = 0.8632

Figure 4.2: Plotted predicted segmentation maps for a slice of Carbonate during different stages of training and ground truth (grain in yellow, multi-phase in green and pore in purple). These plots indicate that the network only uses half a epoch before it is able to differentiate between pore and grain, and that most of the training time is spent learning how to properly segment multi-phase.

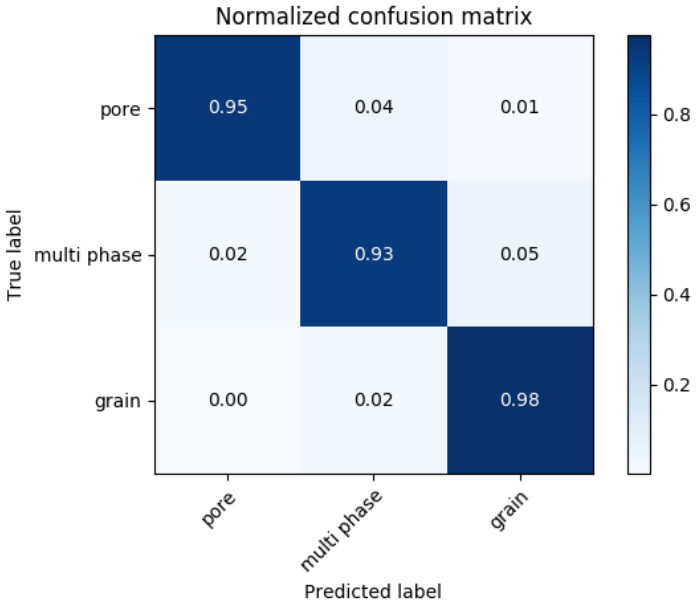


Figure 4.3: Confusion matrix shows that the model seldom confuses grain and pore, this only happens 1% of the time. Most confusion occurs between pore and multi-phase, together with multi-phase and grain. Results also shows that grain is the easiest class to predict thereafter pore and multi-phase which is the most challenging. Matrix was generated with Matplotlib using the best performing model on the test set.

Type	IoU	Accuracy	Dice
Bentheimer	0.8598	0.9847	0.9152
Berea	0.3446	0.7815	0.3841
Carbonate	0.3386	0.72430	0.4152
Mean	**0.5143**	0.8302	0.5715

Table 4.7: Hybrid network trained only on Bentheimer, tested on all three rock types separately.

Type	IoU	Accuracy	Dice
Bentheimer	0.6920	0.93480	0.7616
Berea	0.8918	0.9831	0.9401
Carbonate	0.4207	0.7348	0.5143
Mean	**0.6682**	0.8842	0.7387

Table 4.8: Hybrid network trained only on Berea, tested on all three rock types separately.

Type	IoU	Accuracy	Dice
Bentheimer	0.0964	0.2122	0.1482
Berea	0.1848	0.2707	0.0300
Carbonate	0.8428	0.9552	0.9119
Mean	**0.3747**	0.4794	0.4533

Table 4.9: Hybrid network trained only on Carbonate, tested on all three rock types separately.

Type	IoU	Accuracy	Dice
Bentheimer	0.8818	0.9868	0.9306
Berea	0.8980	0.9849	0.9437
Carbonate	0.5274	0.8001	0.6431
Mean	**0.7691**	0.9239	0.8391

Table 4.10: Hybrid network trained on Bentheimer and Berea, tested on all three rock types separately.

Type	IoU	Accuracy	Dice
Bentheimer	0.8548	0.9848	0.9108
Berea	0.3756	0.7923	0.4296
Carbonate	0.8565	0.9623	0.9204
Mean	**0.6956**	0.9130	0.7536

Table 4.11: Hybrid network trained on Bentheimer and Carbonate, tested on all three rock types separately.

Type	IoU	Accuracy	Dice
Bentheimer	0.5163	0.6877	0.6110
Berea	0.8880	0.9828	0.9375
Carbonate	0.8624	0.9540	0.9239
Mean	**0.7555**	0.8748	0.8241

Table 4.12: Hybrid network trained on Berea and Carbonate, tested on all three rock types separately.

Type	IoU	Accuracy	Dice
Bentheimer	0.876	0.9876	0.9268
Berea	0.9005	0.9853	0.9451
Carbonate	0.8859	0.9792	0.9355
Mean	0.8874	0.9840	0.9358

Table 4.13: Hybrid network trained on Bentheimer, Berea and Carbonate, tested on all three rock types separately.

Chapter 5

Discussion and Conclusion

This chapter concludes the work presented in this thesis. It starts by discussing the findings presented in Chapter 4, before presenting the conclusion. Finally, we discuss future work that can potentially improve our results and provide insights.

5.1 Discussion

This section discuss our results with respect to the research questions.

RQ1: How does 2D-convolutional architectures compare to a 2.5D hybrid architecture that includes depth as input channels?

In order to address RQ1 we compared the results of experimenting with two different convolutional networks, DRNet and PDRNet. Table 4.5 and Table 4.6 show that DRNet outperformed PDRNet in every experiment. We extend DRNet to include depth channels, Table 4.6 shows results and training time when training HDRNet. By increasing depth channels, the network yields better results, but with increased computational cost. When trained with 6 depth channels, it uses 3.2 times longer when comparing with 0 depth channels. We argue that 4-depth channels are a good compromise between performance and computation (1.77 times longer training time). It can be observed that the best run of HDRNet achieves an IoU of 88.6%, which is 2.2% higher than the best run of DRNet.

RQ2: Can deep learning models trained to segment porosity of one rock type generalize and accurately segment other types of rock?

In Section 4.3.3 we presented results from the experiments that were performed to help answer RQ2. Table 4.7, 4.8 and 4.9 indicate that all models trained on individual stones are able to perform reasonably well when segmentation is performed on themselves. An interesting observation is that Bentheimer is better able to use features from Berea than the other way around. When trained only on Bentheimer, Berea get an IoU of 0.3446, but when trained only on Berea, Bentheimer get an IoU of 0.6920. We can also see that Carbonate perform better when trained on Berea than Bentheimer. Table 4.9 clearly shows that Carbonate is the hardest stone to extract common features from by yielding IoUs of only 0.0964 and 0.1848 for Bentheimer and Berea respectively. When combining two rock types in the training we can observe a general

increase in both rock types of which the network was trained. Carbonate benefit when Berea is included in the training, but Berea IoU is slightly decreased. Both Berea and Bentheimer benefit from including the other in the training process. When trained on Bentheimer and Carbonate, Bentheimer stays nearly constant but Carbonate IoU is increased. Both Bentheimer and Carbonate is able to use information from a network trained on Berea + Carbonate, and Bentheimer + Berea, respectively (IoUs of 51.5%-52.7%). By looking at the mean scores of including a second stone in the training process, we observe almost double mean IoU. Of the three combinations we see that both Bentheimer and Berea performs better with each other than when they are combined with Carbonate. To answer the RQ, we conclude that the network is able to generalize across different rock types to some extent.

A observation made when analyzing the result presented in Table 4.4-4.6, is that deeper encoders does not increase performance as expected (not to be confused with depth channels mentioned earlier). Furthermore, by applying medium or strong data augmentations, the IoU actually decreases. One would expect that stronger augmentations, i.e more training data, would yield better results. We theorize that this might not be the case for digital rock volumes, as this might introduce too much noise to features important to the final segmentation.

When comparing our results with the results of Kjerland 2017 thesis, we see a general improvement in both computational time and dice scores. This is however most evident when the rock type that is tested on is included in the training process. See Table 5.1 for a list of the results presented in Kjerland and ours. The table indicate that DeepMedic, the 3D based convolutional network, is better able to extract features from other rock types when only one rock type is included in training. However, once two samples are included our network perform equal or better. Because Kjerland did not run experiments for every possible combination, we are limited to comparing the three listed combinations.

Trained On	**Tested On**	**Dice (Kjerland)**	**Dice (Our)**
BT	Bentheimer	0.89	**0.92**
BT + BR	Bentheimer	0.87	**0.93**
BT + BR + CA	Bentheimer	0.85	**0.94**
BT	Berea	**0.78**	0.38
BT + BR	Berea	0.94	0.94
BT + BR + CA	Berea	0.94	**0.95**
BT	Carbonate	**0.53**	0.41
BT + BR	Carbonate	0.57	**0.64**
BT + BR + CA	Carbonate	0.91	**0.93**

Table 5.1: Table showing the dice scores of Kjerland and our thesis, when trained with different combinations of rock types. BT = Bentheimer, BR = Berea, CA = Carbonate. Kjerland dice scores are computed by averaging the scores for each experiment listed in the Appendix, Section B, Kjerland page 51.

5.2 Conclusion

In this thesis, we have presented our work on the task of semantic segmentation of digital rocks. The produced segments are classified as either grain, micro-phase and pore. The deep learning architectures have been trained on three different rock types, Bentheimer, Berea and Carbonate. We have developed three different deep learning models, all of which are implementations of 2D convolutional neural networks. First we developed DRNet, which is a network inspired by the original U-Net. Second, the PDRNet was developed as an extension of the Feature Pyramid Network (FPN). Third, we developed a further extension of the two above networks, called HDRNet. A problem with the two previously mentioned architectures is that they segment 3D volumes slice by slice. The decision of not doing 3D convolutions was originally made because of the computational complexity. HDRNet is proposed as a compromise between 2D and 3D architectures, where we use the depth information available by feeding the network adjacent slices as channels. HDRNet can be seen as a 2.5D architecture and supports an arbitrary number of extra depth input channels.

The models are able to extract features from other stones to some degree, where Berea is the type that both of the others are best able to learn from. Carbonate is difficult to extract common features from, and does in fact slightly decrease IoU for the other stone types when included. This is probably due to the high amount of multi-phase pixels, which is the hardest class to predict (shown by the confusion matrix in Figure 4.3).

We have showed that, when including all sandstones in the training process, HDRNet (the 2.5D based approach) is able to outperform the DeepMedic architecture which is a 3D convolutional network. We believe however, that once hardware capabilities reach a level where it is viable train 3D-convolutional networks with larger patches and utilize all available data, this will be a better approach.

The biggest limitation of our work is that the provided dataset ground truth is produced by averaging several thresholding algorithms. This means that in effect, our solution is in fact learning the parameters of those algorithms. Section 5.3 presents some ideas on how to improve this datasets, as well as our view on future work.

5.3 Future Work

Improving the quality and size of the Digital Rocks dataset would be a good starting point for improving the models presented in this thesis. Our results shows that the impact transfer learning has is limited. Using pre-trained weights on a dataset more similar to ours, could speed up training time while also improve results. We have not been able to find such a dataset. However, with the increasing number of research papers published regarding digital rocks, one is likely to appear in the future. An interesting experiment, would be to train and evaluate networks on data from the exact same core sample, but produced by two separate scans. This could give some insights into how much variation that is present in a scan, and how much this affects performance.

Another interesting research topic that we think should be looked into, is the use of Single Image Super Resolution (SISR) as a digital rock segmentation preconditioner. SISR is a challenging problem that has received a lot of attention the last two years. The goal is to obtain a high resolution image from a low resolution image [32]. Y. Da Wang et al. [33] used Generative Adversarial Networks (GANs) to create a model that is capable of obtaining a higher resolution

micro-CT from a lower one. This can compensate for the limited field of view in high resolution scanning devices. By doing this, it is possible create higher quality training data, and directly improve upon the work presented in this thesis.

Bibliography

[1] A. Braathen, J. Tveranger, H. Fossen, T. Skar, N. Cardozo, S. E. Semshaug, E. Bastesen, and E. Sverdrup, "Fault facies and its application to sandstone reservoirs," *AAPG Bulletin*, vol. 93, no. 7, pp. 891–917, 2009.

[2] J. Mathew, "Medium article: Deep learning for image segmentation." `https://medium.com/datadriveninvestor/deep-learning-for-image-segmentation-d10d19131113`, 2019.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] L. Deka and M. Quddus, "Network-level accident-mapping: Distance based pattern matching using artificial neural network," *Accident; analysis and prevention*, vol. 65C, pp. 105–113, 12 2013.

[5] V. Gupta, "Understanding feedforward neural networks." `https://www.learnopencv.com/understanding-feedforward-neural-networks/`, 2019.

[6] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[7] M. Stewart, "Introduction to convolutional neural networks," 2019. See appendix for URL.

[8] Rice, "Other methods of edge detection." `https://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/moredge.html`, 2019.

[9] Unknown, "Max-pooling / pooling." `https://computersciencewiki.org/index.php/Max-pooling_/_Pooling`, 2019.

[10] A. Bzt, "Kaggle notebook: 3d convolutions," 2019.

[11] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[12] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *Lecture Notes in Computer Science (including subseries Lecture*

Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9351, pp. 234–241, 2015.

[13] S. Sahoo, "Residual blocks — building blocks of resnet," 2018. See appendix for URL.

[14] S. T. March and G. F. Smith, "Design and natural science research on information technology," *Decision Support Systems*, vol. 15, no. 4, pp. 251–266, 1995.

[15] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[16] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv preprint arXiv:1712.04621*, 2017.

[17] D. G. Pinaki Pratim Acharjya, "An Overview on Watershed Transform and Its Consequences," *International Journal of Engineering and Innovative Technology (IJEIT)*, vol. 1, no. 5, pp. 168–172, 2012.

[18] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and checkerboard artifacts," *Distill*, 2016.

[19] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[21] D. Budgen and P. Brereton, "Performing systematic literature reviews in software engineering," *Proceeding of the 28th international conference on Software engineering - ICSE '06*, p. 1051, 2006.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances In Neural Information Processing Systems*, 2012.

[23] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[24] S. Karimpouli and P. Tahmasebi, "Segmentation of digital rock images using deep convolutional autoencoder networks," *Computers and Geosciences*, vol. 126, no. October 2018, pp. 142–150, 2019.

[25] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *CoRR*, vol. abs/1505.00387, 2015.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016.

[27] Ø. Kjerland, "Segmentation of coronary arteries from ct-scans of the heart using deep learning," Master's thesis, NTNU, 2017.

[28] T. Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017.

[29] Google, "Why tensorflow." `https://www.tensorflow.org/`, 2019.

[30] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[31] E. K. V. I. I. A. Buslaev, A. Parinov and A. A. Kalinin, "Albumentations: fast and flexible image augmentations," *ArXiv e-prints*, 2018.

[32] W. Yang, X. Zhang, Y. Tian, W. Wang, and J.-H. Xue, "Deep Learning for Single Image Super-Resolution: A Brief Review," pp. 1–17, 2018.

[33] Y. Da Wang, R. Armstrong, and P. Mostaghimi, "Super resolution convolutional neural network models for enhancing resolution of rock micro-ct images," *arXiv preprint arXiv:1904.07470*, 2019.

Appendix A

Seismic and Salt Identification Challenge

This section contains material related to seismic data which was the original research focus.

Seismic data

Seismic surveys is the process of mapping the subsurface. In this project the seismic data consists of images of the geological structures under the seabed. The mapping is done by emitting sound waves into the rock formations below the sea. The sound waves are reflected back to sensors which are usually towed behind a vessel performing the survey. The strength of reflection is proportional to the different properties in the rock layers. 3D images are produced using multiple parallel sensor cables, providing higher resolution compared to 2D images. A seismic image shows boundaries between different rock types, but it can be difficult to accurately identify what class a rock belongs to. Figure A.1 shows how a seismic survey is carried out on sea. Most seismic 3D data volumes comes in a .segy format which does not always follow the same specification. This makes it difficult to develop a general method of parsing the datasets and feeding the information to a network. A typical slice (one image in a 3D volume) of a seismic dataset is illustrated in Figure A.2.

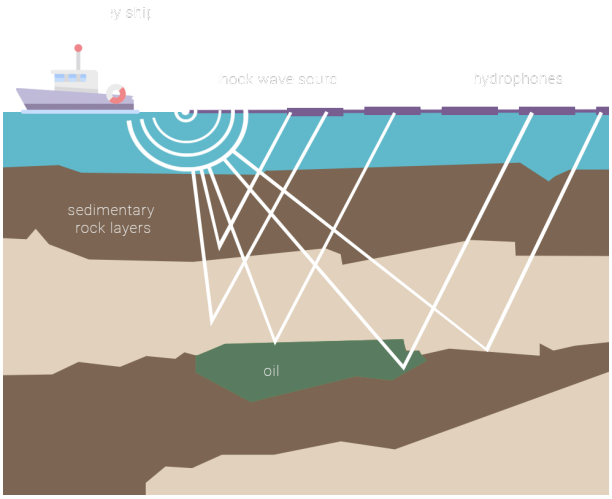


Figure A.1: Seismic survey by vessel (illustration from RagnarockGeo AS).

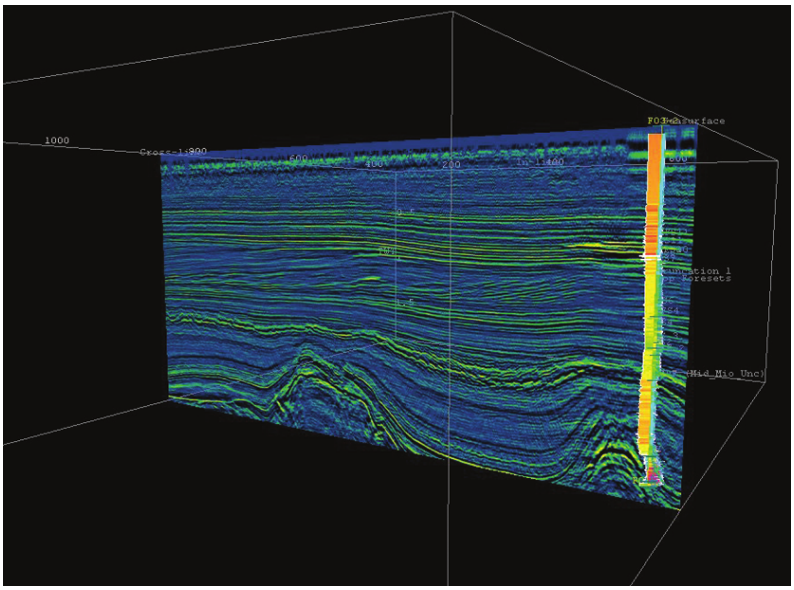


Figure A.2: Illustration of a slice in a 3D seismic volume

Problem leading to dismissal of seismic

This thesis was originally going to focus primarily on the task of automatically segment seismic 3D volumes. Two different datasets were explored, before the conclusion of abandoning seismic in favor of digital rocks.

F3 Netherlands Offshore F3 Block
The first dataset on which preliminary experiments were carried out. The dataset has one major drawback, and that is the fact that only four slices are labelled, where one is used for validation. At first this was not an issue regarding the development of the planned model, as datasets with more labels were to be produced and delivered later. F3 was a good starting point and different data augmentation techniques were applied in order to generate more training data.

The Volve dataset
Volve was recently (June, 2018) published by Equinor. The dataset consist of very detailed and large amounts data from a field on the Norwegian continental shelf, with measurements from 2008-2016. The whole dataset amounts to over 5 TB and contains a huge amount of labelled data. However, a major problem with this dataset is that it specifies every sensor value with respect to real world coordinates. This results in a very difficult problem of converting the input from coordinates to fit in a matrix that can be used to feed a network. Originally we were promised a working solution to transform the dataset into matrices, but was unfortunately not delivered. This ultimately became the reason the thesis had to shift focus.

TGS Salt Challenge

While trying to parse the Volve dataset we participated in the TGS Salt Identification challenge on Kaggle.

Data

There are many locations on earth where there are large deposits of salt in the subsurface. A challenge in seismic data is to identify which part of the subsurface is salt. The dataset provided by the TGS Salt Identification challenge consists of images of 101x101 pixels chosen from different locations below the seabed. Each pixel is classified as either sediment or salt. All images are in .png format and in addition to the image the depth of the location is provided.

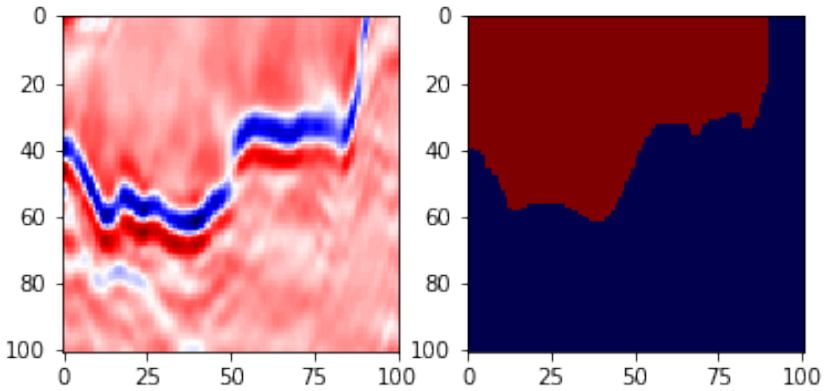


Figure A.3: TGS Salt Data. The left image shows the original image, while the right is the mask where blue pixels belong to sediment and red salt

Results

Our best submission scored a private IoU of 0.77030 and 0.74344 on the public leaderboard. The following pages documents this submission.

In [1]:

```
%matplotlib inline
import tensorflow as tf
import pandas as pd
from notify_slack import notify
from skimage.transform import resize
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import array_to_img, load_img
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Dense, Dropout, Flatten, Input, Concatenate,
from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D, Conv2DTr
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLRO
from tensorflow.keras.layers import Lambda, RepeatVector, Reshape, concatenate,
from tensorflow.keras.utils import multi_gpu_model
from tensorflow.keras.optimizers import Adam
import os
```

In [2]:

```
GPU_NUMBER = 0
if GPU_NUMBER != None:
    os.environ["CUDA_VISIBLE_DEVICES"] = str(GPU_NUMBER)
```

In [3]:

```
def read_images(path: str):
    images = []
    filenames = []
    for filename in os.listdir(path):
        image_array = np.asarray(load_img(path + filename, grayscale=True)) / 25
        images.append(image_array)
        filenames.append(filename.split(".")[0])
    return filenames, images
```

In [4]:

```
depths_df = pd.read_csv("data/depths.csv", index_col="id")
train_df = pd.read_csv("data/train.csv", index_col="id", usecols=[0])
train_df = train_df.join(depths_df)
test_df = depths_df[~depths_df.index.isin(train_df.index)]
```

In [5]:

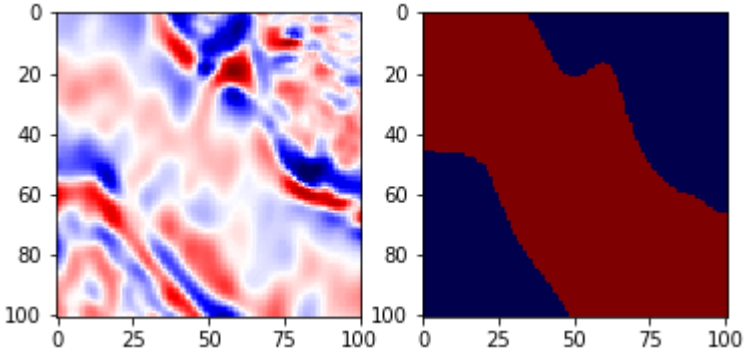
```
filenames, images = read_images("data/train/images/")
train_df = pd.merge(pd.DataFrame({"id": filenames, "images": images}), train_df,
filenames, masks = read_images("data/train/masks/")
train_df = pd.merge(pd.DataFrame({"id": filenames, "masks": masks}), train_df, o
train_df.head()
```

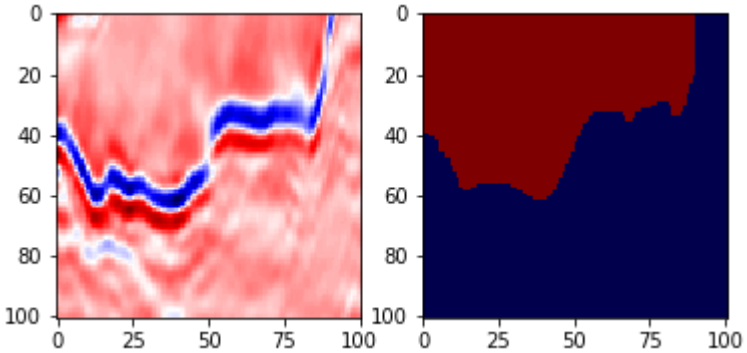
Out[5]:

	id	masks	images	z
0	b71915a0fe	[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...	[[0.5725490196078431, 0.4862745098039216, 0.4...	288
1	8320911258	[[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,...	[[0.5254901960784314, 0.5254901960784314, 0.51...	290
2	77608c7770	[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...	[[0.36470588235294116, 0.3686274509803922, 0.3...	512
3	f122ead5f2	[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...	[[0.7019607843137254, 0.6235294117647059, 0.54...	513
4	0c02f95a08	[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...	[[0.5372549019607843, 0.5137254901960784, 0.48...	681

In [6]:

```
def visualize(id: str) -> None:
    row = train_df.loc[train_df["id"] == id]
    img = row["images"].values[0]
    mask = row["masks"].values[0]
    plt.figure()
    plt.subplot(121)
    plt.imshow(img, cmap="seismic")
    plt.subplot(122)
    plt.imshow(mask, cmap="seismic")
visualize("0a1742c740")
visualize("f5c2e66754")
```





In [16]:

```
def get_iou_vector(A, B):
    batch_size = A.shape[0]
    metric = []
    for batch in range(batch_size):
        t, p = A[batch]>0, B[batch]>0
        intersection = np.logical_and(t, p)
        union = np.logical_or(t, p)
        iou = (np.sum(intersection > 0) + 1e-10 )/ (np.sum(union > 0) + 1e-10)
        thresholds = np.arange(0.5, 1, 0.05)
        s = []
        for thresh in thresholds:
            s.append(iou > thresh)
        metric.append(np.mean(s))

    return np.mean(metric)

def iou(label, pred):
    return tf.py_func(get_iou_vector, [label, pred>0.5], tf.float64)

def iou2(label, pred):
    return tf.py_func(get_iou_vector, [label, pred >0], tf.float64)
```

In [15]:

```
def conv2d_block(input_tensor, n_filters, kernel_size=3, batchnorm=True):
    x = Conv2D(filters=n_filters, kernel_size=(kernel_size, kernel_size),paddin
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = Conv2D(filters=n_filters, kernel_size=(kernel_size, kernel_size), paddi
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation("relu")(x)
    return x

def get_unet(input_img, n_filters=32, dropout=0.5, batchnorm=True):
    # contracting path
    c1 = conv2d_block(input_img, n_filters=n_filters*1, kernel_size=3, batchnorm
    p1 = MaxPooling2D((2, 2)) (c1)
    p1 = Dropout(dropout*0.5)(p1)
    c2 = conv2d_block(p1, n_filters=n_filters*2, kernel_size=3, batchnorm=batchn
    p2 = MaxPooling2D((2, 2)) (c2)
    p2 = Dropout(dropout)(p2)
    c3 = conv2d_block(p2, n_filters=n_filters*4, kernel_size=3, batchnorm=batchn
    p3 = MaxPooling2D((2, 2)) (c3)
    p3 = Dropout(dropout)(p3)
    c4 = conv2d_block(p3, n_filters=n_filters*8, kernel_size=3, batchnorm=batchn
    p4 = MaxPooling2D(pool_size=(2, 2)) (c4)
    p4 = Dropout(dropout)(p4)
    c5 = conv2d_block(p4, n_filters=n_filters*16, kernel_size=3, batchnorm=batch
    # expansive path
    u6 = Conv2DTranspose(n_filters*8, (3, 3), strides=(2, 2), padding='same') (c
    u6 = concatenate([u6, c4])
    u6 = Dropout(dropout)(u6)
    c6 = conv2d_block(u6, n_filters=n_filters*8, kernel_size=3, batchnorm=batchn
    u7 = Conv2DTranspose(n_filters*4, (3, 3), strides=(2, 2), padding='same') (c
    u7 = concatenate([u7, c3])
    u7 = Dropout(dropout)(u7)
    c7 = conv2d_block(u7, n_filters=n_filters*4, kernel_size=3, batchnorm=batchn
    u8 = Conv2DTranspose(n_filters*2, (3, 3), strides=(2, 2), padding='same') (c
    u8 = concatenate([u8, c2])
    u8 = Dropout(dropout)(u8)
    c8 = conv2d_block(u8, n_filters=n_filters*2, kernel_size=3, batchnorm=batchn
    u9 = Conv2DTranspose(n_filters*1, (3, 3), strides=(2, 2), padding='same') (c
    u9 = concatenate([u9, c1], axis=3)
    u9 = Dropout(dropout)(u9)
    c9 = conv2d_block(u9, n_filters=n_filters*1, kernel_size=3, batchnorm=batchn
    output = Conv2D(1, (1, 1), activation=None)(c9)
    prediction = Activation("sigmoid")(output)
```

```
58
59      model = Model(inputs=[input_img], outputs=[prediction])
60      return model
61
62
63  input_img = Input((128, 128, 1), name='img')
64
65  model = get_unet(input_img, n_filters=16, dropout=0.15, batchnorm=True)
66
67  summary = model.summary()
68
69  if GPU_NUMBER == None:
70      model = multi_gpu_model(model, gpus=2, cpu_merge=True, cpu_relocation=False
71
72  model.compile(loss="binary_crossentropy", optimizer='adam', metrics=["acc", iou
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   Conne
cted to
=================================================================
img (InputLayer)             (None, 128, 128, 1)       0
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 128, 128, 16)      160       img
[0][0]
_________________________________________________________________
batch_normalization_36 (BatchNo (None, 128, 128, 16)   64        conv2
d_38[0][0]
_________________________________________________________________
activation_38 (Activation)   (None, 128, 128, 16)      0         batch
 normalization 36[0][0]
```

In [8]:

```
images = []
for image in train_df.images.values:
    image = np.stack((image,)*1, -1)
    image = resize(image, (128, 128, 1), anti_aliasing=True, mode='constant', pr
    images.append(image)

segments = []
for mask in train_df.masks.values:
    segment = np.stack((mask,)*1, -1)
    segment = resize(segment, (128, 128, 1), anti_aliasing=True, mode='constant
    segments.append(segment)

x = np.array(images)
y = np.array(segments)

x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size=0.1, rando

# Data Augmentation

# Vertical Flip
Vx = [np.flip(i, axis=0) for i in x_train]
Vy = [np.flip(i, axis=0) for i in y_train]

# Horizontal Flip
Hx = [np.flip(i, axis=1) for i in x_train]
Hy = [np.flip(i, axis=1) for i in y_train]

# Horizontal Vertical Flip
HVx = [np.flip(i, axis=1) for i in Vx]
HVy = [np.flip(i, axis=1) for i in Vy]

# Appending the augmented image and mask to the main dataset.
x_train = np.append(x_train, Vx, axis=0)
x_train = np.append(x_train, Hx, axis=0)
x_train = np.append(x_train, HVx, axis=0)

y_train = np.append(y_train, Vy, axis=0)
y_train = np.append(y_train, Hy, axis=0)
y_train = np.append(y_train, HVy, axis=0)


```

In [9]:

```
early_stopping = EarlyStopping(patience=20, verbose=1)
model_checkpoint = ModelCheckpoint("models/model_checkpoint.h5", monitor='iou',
reduce_lr = ReduceLROnPlateau(factor=0.5, patience=6, min_lr=0.0001, verbose=1)
```

In [14]:

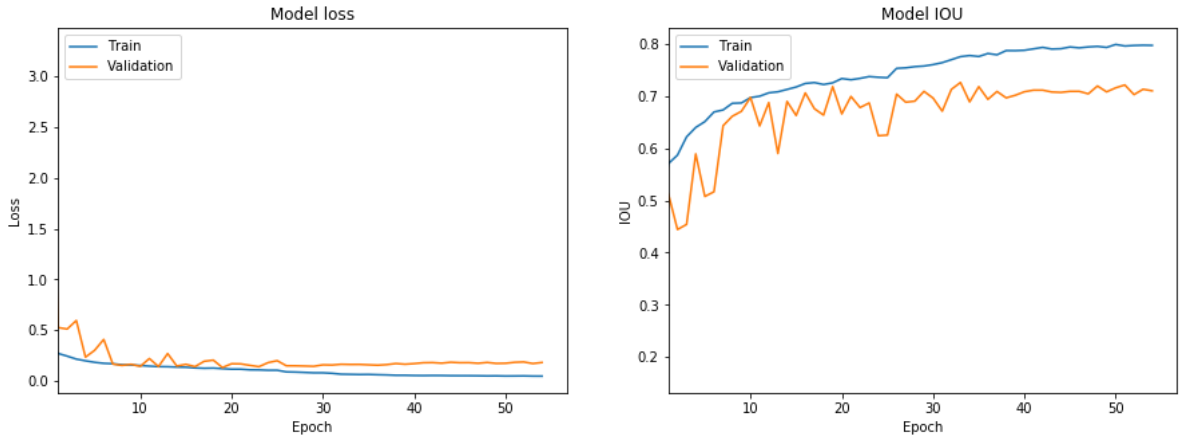
```
epochs = 55
batch_size = 32

if GPU_NUMBER == None:
    batch_size = 2 * batch_size

history = model.fit(x=x_train,
                    y=y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_data=(x_valid, y_valid),
                    callbacks=[model_checkpoint, reduce_lr])

val_loss, val_acc, val_iou = model.evaluate(x=x_valid, y=y_valid, batch_size=bat
filename = f"model - epochs: {epochs} - iou: {round(val_iou,3)} - loss: {round(v
model.save(f"models/{filename}.h5")


```

Out[14]:

```
'\nhistory = model.fit(x=x_train,\n                    y=y_train,\n
batch_size=batch_size,\n                    epochs=epochs,\n
validation_data=(x_valid, y_valid),\n                    callbacks=[mo
del_checkpoint, reduce_lr])\n\n#val_loss, val_acc, val_iou = model.eva
luate(x=x_valid, y=y_valid, batch_size=batch_size, verbose=1)\n#filena
me = f"model - epochs: {epochs} - iou: {round(val_iou,3)} - loss: {rou
nd(val_loss,3)} acc: {round(val_acc,3)}"\n#model.save(f"models/{filena
me}.h5")\n'
```

In [17]:

```
if history.history:
    plt.figure(figsize=(15, 5))
    plt.subplot(121)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.xlim(1)
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.subplot(122)
    plt.plot(history.history['iou'])
    plt.plot(history.history['val_iou'])
    plt.title('Model IOU')
    plt.ylabel('IOU')
    plt.xlabel('Epoch')
    plt.xlim(1)
    plt.legend(['Train', 'Validation'], loc='upper left');
    #plt.savefig(f"models/{filename}.png")
```



Appendix B

2017 Kjerland, Digital Rocks Results

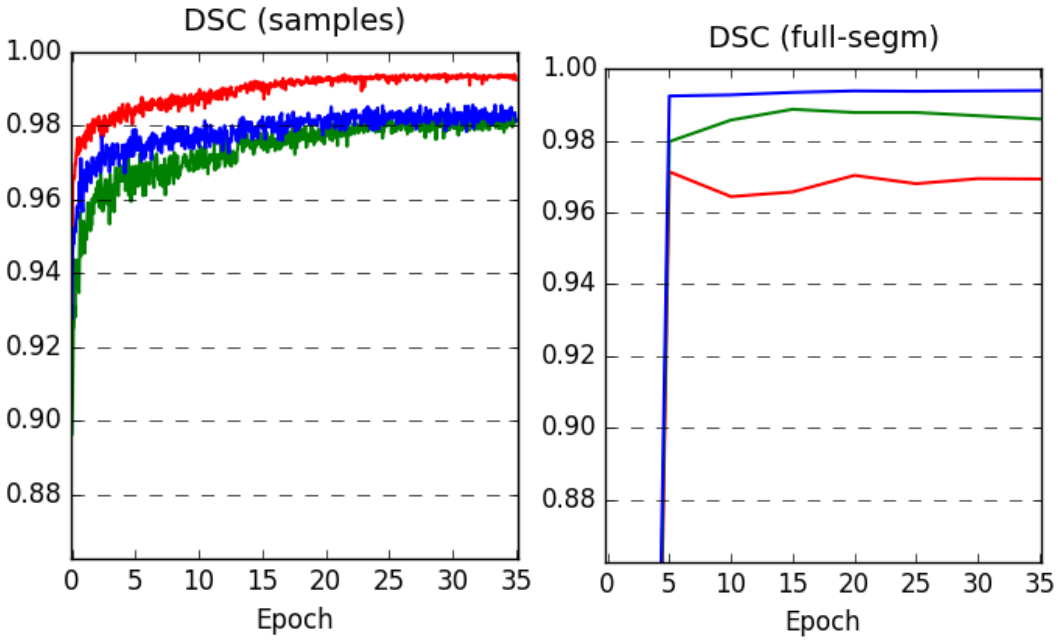
Below are pages 47-54 of Kjerland 2017 masters thesis [27]. The content presents the results he made when segmenting digital rocks with DeepMedic.

4.3 Digital rock segmentation

Figures 4.16-4.18 show plots of the training and validation DSC on pore, multi phase and grain segmentation using the networks $RockCNN_1$, $RockCNN_{10}$ and $RockCNN_{40}$. These networks were trained on different amount of subvolumes from the bentheimer sandstone volume. Figures 4.19-4.19 show the same type of plots but with $RockCNN_{BT}$, $RockCNN_{BT+BR}$ and $RockCNN_{BT+BR+CA}$ instead. The training and validation DSC for all networks are summarized in table 4.5. The results for $RockCNN_{BT}$ are left out as these are the same as the results of $RockCNN_{40}$.

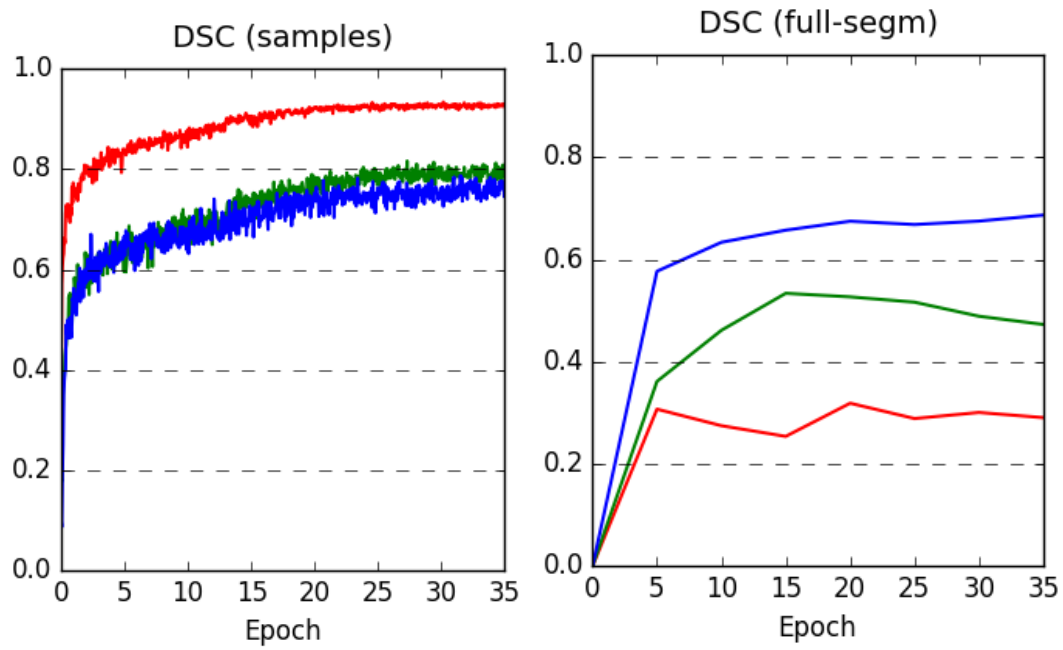
The test DSC for experiments on training and testing on different types of rock are shown in tables 4.6-4.9.

Figure 4.22 show a slice of a bentheimer sandstone subvolume, corresponding ground truth and segmentation results of $RockCNN_1$, $RockCNN_{10}$ and $RockCNN_{40}$. A similar comparison of segmentation results and ground truth for $RockCNN_{BT}$, $RockCNN_{BT+BR}$ and $RockCNN_{BT+BR+CA}$ is given by figure 4.23.



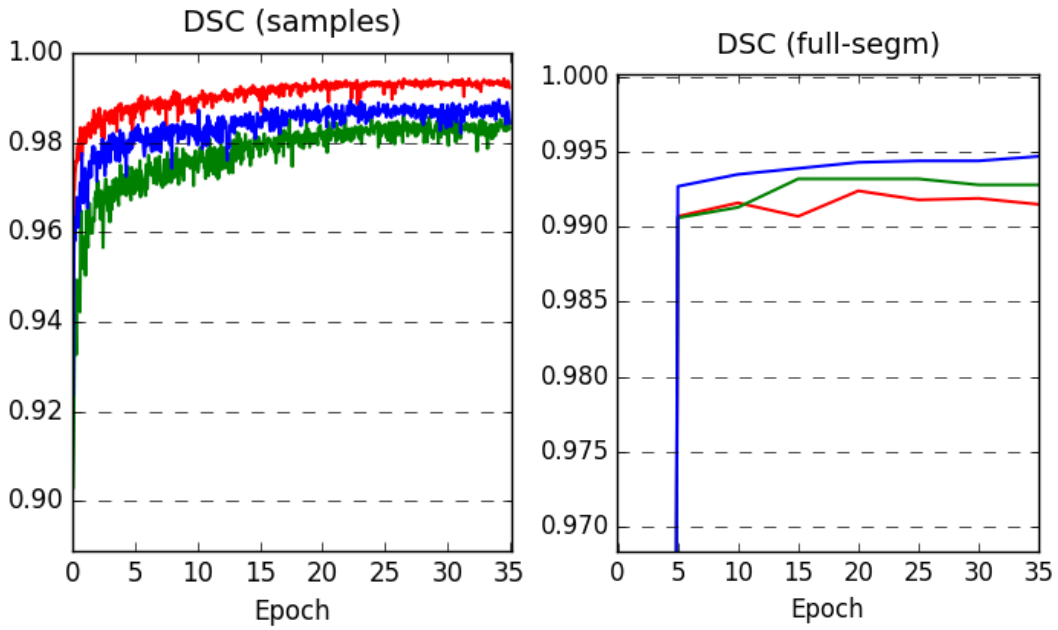
(a) Training **(b)** Validation

Figure 4.16: Training and validation plots for **pore** segmentation experiments $RockCNN_1$ shown in red, $RockCNN_{10}$ shown in green and $RockCNN_{40}$ shown in blue.



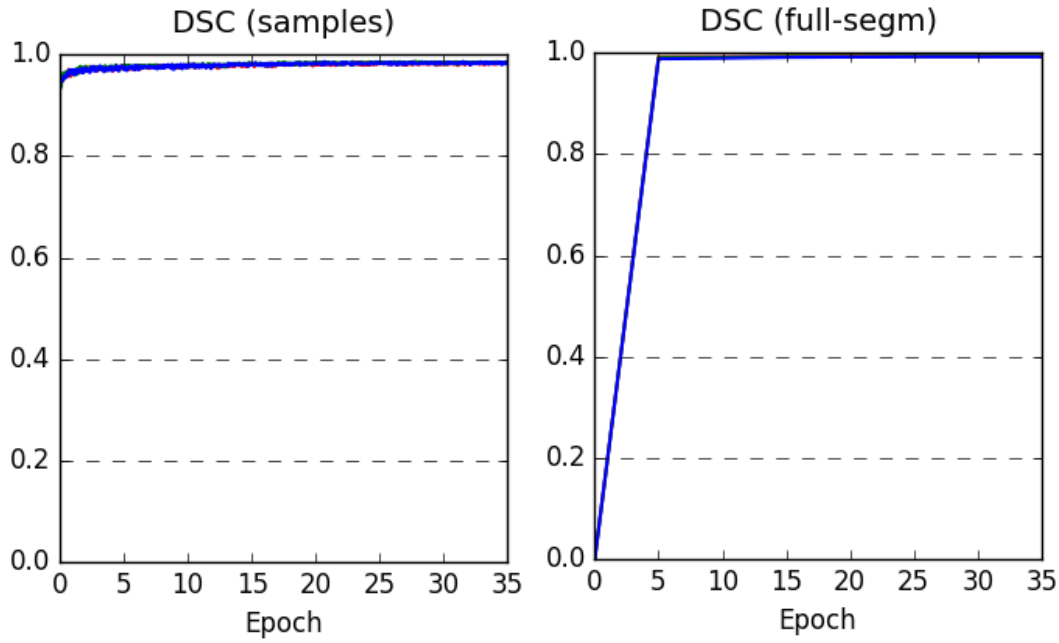
(a) Training **(b)** Validation

Figure 4.17: Training and validation plots for **multi phase** segmentation experiments $RockCNN_1$ shown in red, $RockCNN_{10}$ shown in green and $RockCNN_{40}$ shown in blue.



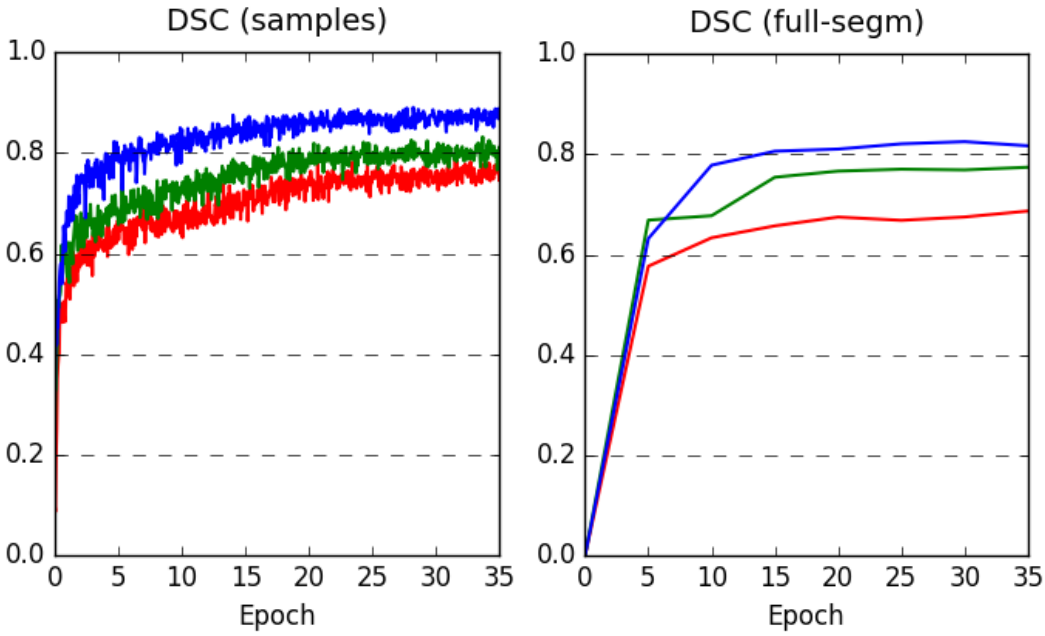
(a) Training **(b)** Validation

Figure 4.18: Training and validation plots for **grain** segmentation experiments $RockCNN_1$ shown in red, $RockCNN_{10}$ shown in green and $RockCNN_{40}$ shown in blue.



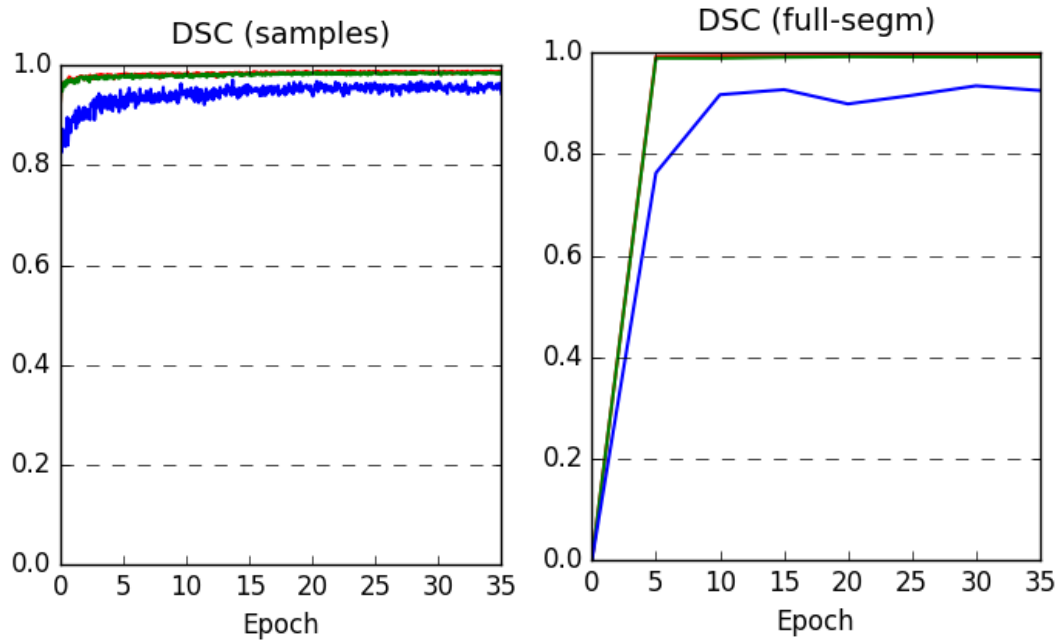
(a) Training **(b)** Validation

Figure 4.19: Training and validation plots for **pore** segmentation experiments $RockCNN_{BT}$ shown in red, $RockCNN_{BT+BR}$ shown in green and $RockCNN_{BT+BR+CA}$ shown in blue.



(a) Training **(b)** Validation

Figure 4.20: Training and validation plots for **multi phase** segmentation experiments $RockCNN_{BT}$ shown in red, $RockCNN_{BT+BR}$ shown in green and $RockCNN_{BT+BR+CA}$ shown in blue.



(a) Training **(b)** Validation

Figure 4.21: Training and validation plots for **grain** segmentation experiments $RockCNN_{BT}$ shown in red, $RockCNN_{BT+BR}$ shown in green and $RockCNN_{BT+BR+CA}$ shown in blue.

	DSC Training			DSC Validation		
Experiment	**Pore**	**Multi phase**	**Grain**	**Pore**	**Multi phase**	**Grain**
$RockCNN_1$	0.9933	0.9273	0.9933	0.9693	0.2907	0.9915
$RockCNN_{10}$	0.9809	0.7935	0.9839	0.9860	0.4731	0.9928
$RockCNN_{40}$	0.9828	0.7624	0.9872	0.9940	0.6387	0.9952
$RockCNN_{BT}$	-	-	-	-	-	-
$RockCNN_{BT+BR}$	0.9843	0.8018	0.9849	0.9939	0.7745	0.9919
$RockCNN_{BT+BR+CA}$	0.9835	0.8724	0.9564	0.9924	0.8173	0.9257

Table 4.5: Training and validation results.

Experiment	**DSC Pore**	**DSC Multi phase**	**DSC Grain**
$RockCNN_{BT}$	0.9946	0.6922	0.9956
$RockCNN_{BT+BR}$	0.9935	0.6434	0.9954
$RockCNN_{BT+BR+CA}$	0.9921	0.5889	0.9949

Table 4.6: Test DSC scores for type 1, bentheimer sandstone

Experiment	**DSC Pore**	**DSC Multi phase**	**DSC Grain**
$RockCNN_{BT}$	0.9789	0.4060	0.9751
$RockCNN_{BT+BR}$	0.9942	0.8558	0.9888
$RockCNN_{BT+BR+CA}$	0.9937	0.8467	0.9880

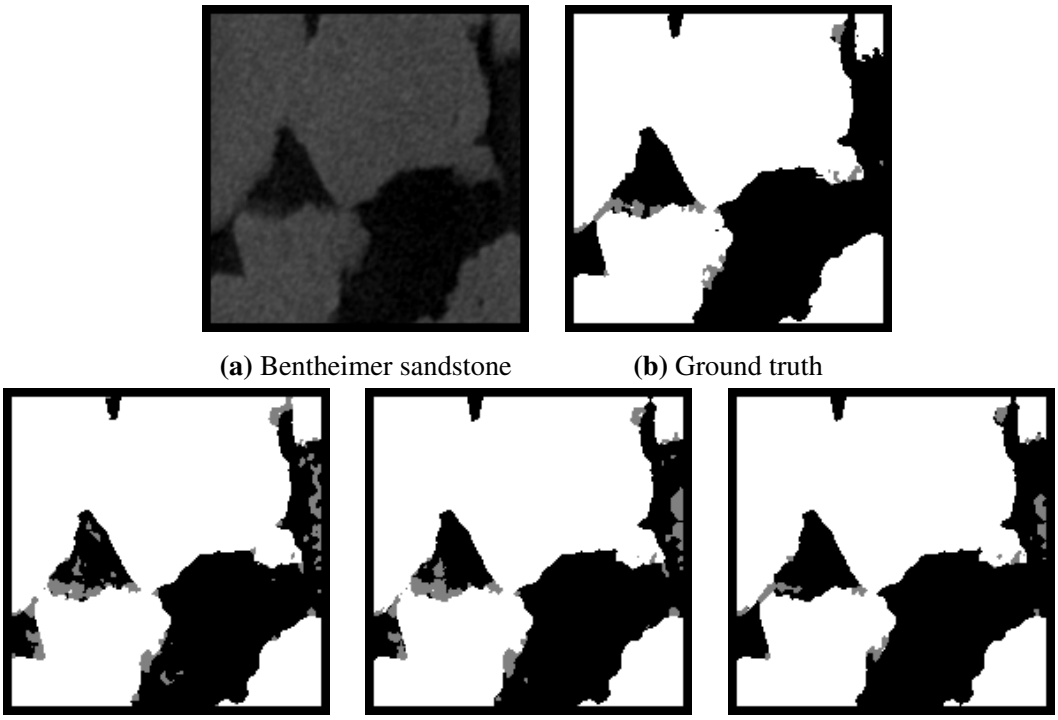
Table 4.7: Test DSC scores for type 2, berea sandstone

Experiment	**DSC Pore**	**DSC Multi phase**	**DSC Grain**
$RockCNN_{BT}$	0.9866	0.1229	0.5048
$RockCNN_{BT+BR}$	0.9895	0.2146	0.5164
$RockCNN_{BT+BR+CA}$	0.9940	0.9117	0.8260

Table 4.8: Test DSC scores for type 1, carbonate

Experiment	**DSC Pore**	**DSC Multi phase**	**DSC Grain**
$RockCNN_{BT}$	0.9867	0.4070	0.8252
$RockCNN_{BT+BR}$	0.9924	0.5713	0.8335
$RockCNN_{BT+BR+CA}$	0.9933	0.7824	0.9363

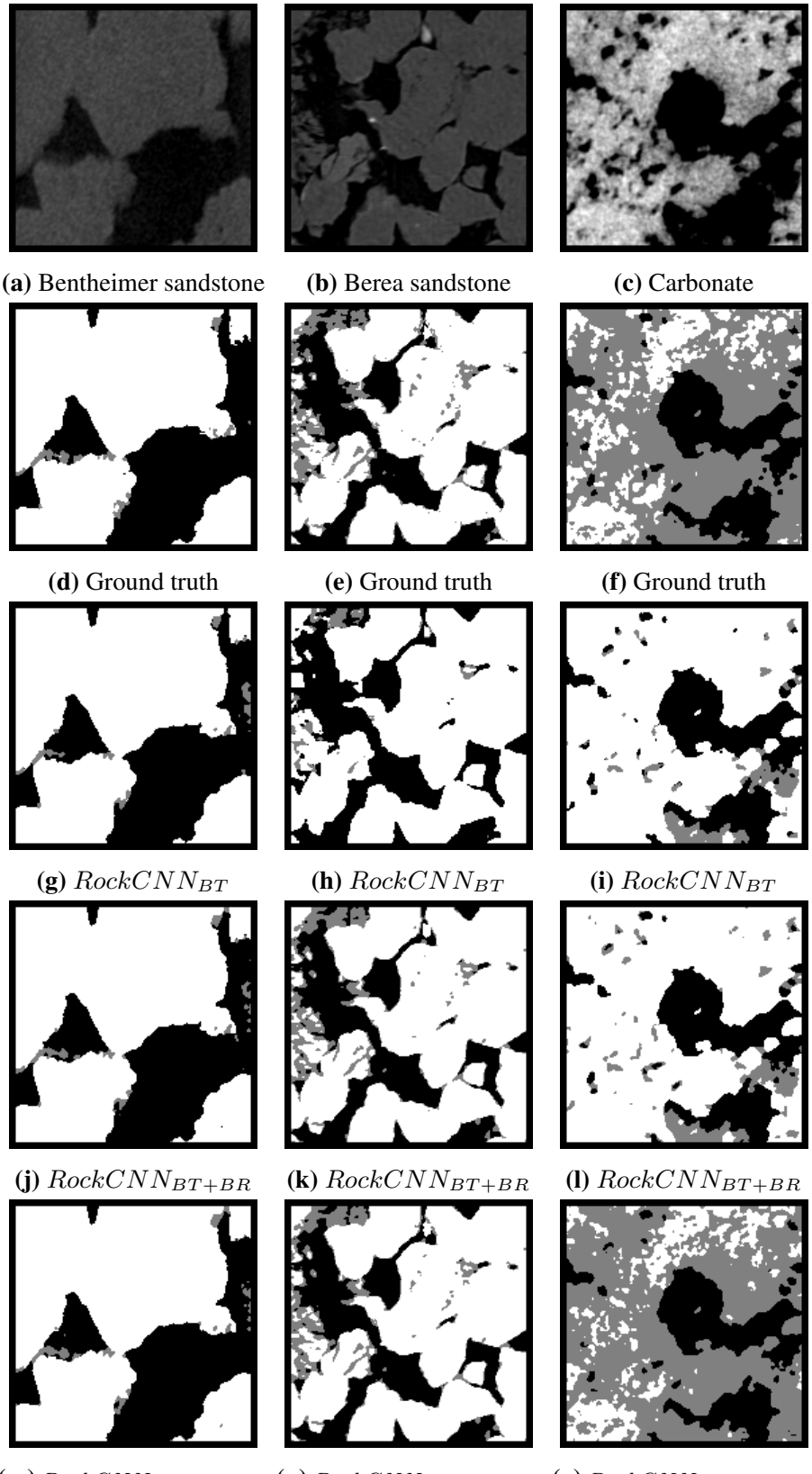
Table 4.9: Mean DSC across all three types.



(a) Bentheimer sandstone **(b)** Ground truth

(c) $RockCNN_1$ **(d)** $RockCNN_{10}$ **(e)** $RockCNN_{40}$

Figure 4.22: Results from experiments $RockCNN_1$, $RockCNN_{10}$ and $RockCNN_{40}$. For the ground truth and segmentation results, pore is represented as black, multi phase as gray and grain as white.



(a) Bentheimer sandstone **(b)** Berea sandstone **(c)** Carbonate

(d) Ground truth **(e)** Ground truth **(f)** Ground truth

(g) $RockCNN_{BT}$ **(h)** $RockCNN_{BT}$ **(i)** $RockCNN_{BT}$

(j) $RockCNN_{BT+BR}$ **(k)** $RockCNN_{BT+BR}$ **(l)** $RockCNN_{BT+BR}$

(m) $RockCNN_{BT+BR+CA}$ **(n)** $RockCNN_{BT+BR+CA}$ **(o)** $RockCNN_{BT+BR+CA}$

Figure 4.23: Comparison of the segmentations by the networks. For the ground truth and segmentation results, pore is represented as black, multi phase as gray and grain as white.

4.4 Timing

The training and inference times of the different networks are presented in table 4.10.

Experiment	Training time	Mean inference time per volume
$CoronaryCNN$	71254s ≈ 20h	285s ≈ 5m
$CoronaryCNN_{0.80mm}$	28691s ≈ 8h	40s
$CoronaryCNN_{flip}$	69714s ≈ 19h	289s ≈ 5m
$CoronaryCNN_{ROI}$	70332s ≈ 20h	290s ≈ 5m
$AortaCNN_{0.40mm}$	67865s ≈ 19h	285s ≈ 5m
$AortaCNN_{0.80mm}$	28721s ≈ 8h	41s
$BrainTumorCNN$	28356s ≈ 8h	55s
$RockCNN_{1}$	28504s ≈ 8h	30s
$RockCNN_{10}$	28986s ≈ 8h	28s
$RockCNN_{40}$	28599s ≈ 8h	28s
$RockCNN_{BT+BR}$	28794s ≈ 8h	26s
$RockCNN_{BT+BR+CA}$	28668s ≈ 8h	30s

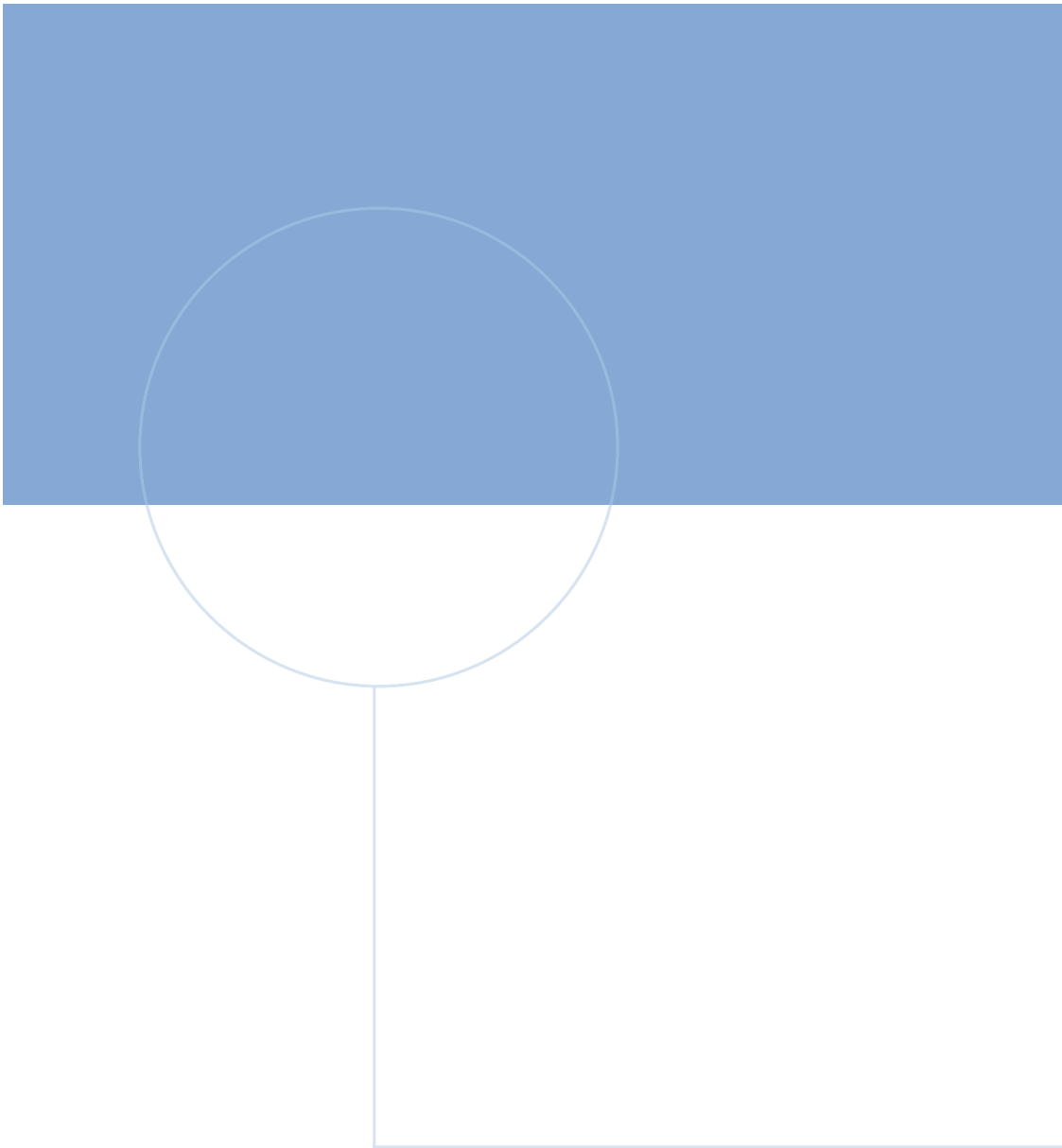
Table 4.10: Training and mean inference times for the networks trained in this thesis. The inference time of the networks used for aorta and coronary artery segmentation were obtained by evaluating them on the pilot dataset. The table shows that networks trained on smaller volumes have similar training and inference times. Networks trained on larger volumes use on average 11 to 12 hours more for training.

Appendix C

Bibliography URLs

Below are links that are too long to fir properly in the bibliography.

- https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac [7]
- https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec [13]



NTNU
Norwegian University of
Science and Technology