Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Geoscience and Petroleum

Ådne Årevik Vikdal

Multi-Phase Segmentation of Imaged Fluid Distribution in Porous Media Using Deep Learning

Master's thesis in Engineering and ICT
Supervisor: Carl Fredrik Berg
June 2021

Ådne Årevik Vikdal

Multi-Phase Segmentation of Imaged Fluid Distribution in Porous Media Using Deep Learning

Master's thesis in Engineering and ICT
Supervisor: Carl Fredrik Berg
June 2021

Norwegian University of Science and Technology
Faculty of Engineering
Department of Geoscience and Petroleum

NTNU
Norwegian University of
Science and Technology

Abstract

Precise segmentation of CT images is a cornerstone in the creation of digital rock models. The traditional segmentation workflow is tedious as it relies on manual interaction and quality control, making it prone to operator bias. Most deep learning approaches use manually segmented images as ground truths, and the models suffer from inherent user bias. Existing published research has only focused on the segmentation of solids and pore space and the segmentation of the solids into minerals.

This thesis implements several deep learning models for automatic segmentation of micro-CT images into multiple fluid phases and solids. To the best of the author's knowledge, there does not exist any published work on deep learning approaches that segments fluid distribution in the pore space as of yet.

The thesis also explores the potential for using synthetic data in the training of deep learning models. The data is created by using a force-biased algorithm and the Lubachevsky-Stillinger algorithm to create jammed sphere packs. Morphological invasion is used to create a fluid distribution consisting of oil and water in the pore space of the sphere packs. Synthetic data creation removes the user bias from the ground truth segmentations that most existing deep learning approaches on porous media segmentation suffer from. Three deep learning models are trained on synthetic images. Their performance on authentic micro-CT images is compared to the segmentations created by a model trained on manual, pixel-wise annotations of the micro-CT images.

MultiRes U-Net is the best performing model trained on synthetic images. It generates accurate segmentations but still has some limitations. These limitations are assumed to be connected to a lack of variety of porosity and contact angles in the training dataset and should easily be resolved by modifying the training dataset.

Sammendrag

Nøyaktig segmentering av CT-bilder er en nøkkelfaktor i utviklingen av Digital Rock Models. Tradisjonell segmenteringsarbeidsflyt er arbeidskrevende ettersom den er avhengig av manuell samhandling og kvalitetskontroll. De fleste nevrale nettverkstilnærminger nytter manuelt segmenterte bilder som ground truths, og slike modeller blir derfor påvirket av operatøren som gjennomfører segmenteringen av ground truth-bildene.

Tidligere forskning har fokusert på segmentering av CT-bilder til faste stoffer og porerom, og segmentering av faste stoffer til mineral. Denne masteroppgaven implementerer flere nevrale nettverksmodeller for automatisk segmentering av mikro-CT bilder til fluidfaser og fast stoff. Såvidt forfatteren vet, er det ikke tidligere publisert forskning om nevrale nettverk som segmenterer fluiddistribusjonen i porerommet.

Masteroppgaven utforsker potensialet for å nytte syntetiske CT-bilder for å trene nevrale nettverksmodeller. En force-biased-algoritme og Lubachevsky-Stillinger-algoritmen blir brukt til å generere kulepakker. Videre blir morfologisk invadering benyttet til å danne en fluiddistribusjon med vann og olje i porevolumet. Disse kulepakkene blir deretter prosessert for å etterligne CT-bilder. Syntetisk datagenerering gjør at ground truth-bildene ikke blir påvirket av den som lagde dem. I denne oppgaven er tre nevrale nettverk trent med syntetiske bilder. Disse nettverkene er testet på ekte mikro-CT-bilder, og resultatene er sammenlignet med segmenteringene fra et nevralt nettverk trent med ekte bilder, der ground truth-bildene er laget ved hjelp av manuell annotering.

Av nettverkene som er trent på syntetiske bilder, er det MultiRes U-Net som gir de beste segmenteringene. Selv om den gir nøyaktige segmenteringer, har den likevel noen svakheter. Svakhetene er trolig knyttet til treningsdataen og kan enkelt fjernes ved å modifisere treningsdatasettet.

Acknowledgement

This master thesis is written at the Department of Geoscience and Petroleum, Norwegian University of Science and Technology. It is the final product of the master program in Engineering & ICT, with specialization in ICT & Petroleum Engineering.

I would like to extend my gratitude to my supervisor, Carl Fredrik Berg, for providing me with an interesting topic for this master thesis, and for your knowledge, guidance and interesting discussions during my work on this master thesis.

I would also like to thank Haili Long-Sanouiller and Petricore AS for providing me with micro-CT images. Thanks to my family for your support during my five years at NTNU, and thanks to my friends and classmates for making these years in Trondheim so great.

Contents

Figures

Tables

Acronyms

Adam	Adaptive Moment Estimation. 37
ANN	Artificial Neural Network. 13, 15, 17
CNN	Convolutional Neural Network. 14, 15, 17, 20, 21
FCN	Fully Convolutional Network. ix, 29, 31
IoU	Intersection over Union. 61
ReLU	Rectified Linear Unit. 15, 16, 32
SGD	Stochastic Gradient Descent. 11, 37, 38, 66
SVM	Support Vector Machine. 12, 38, 56

Chapter 1

Introduction

Micro-CT images of core samples of porous media describe the pore geometry in high resolution. The images can be used to derive rock and fluid properties, which may lead to a deeper understanding of the rock properties, pore structure and the processes that control the transport of fluids in porous media. With digitized core samples, these properties will be determined faster than in a traditional laboratory.

In order to develop precise digital rock models, a crucial step is the segmentation of the micro-CT images. Traditional segmentation methods are often tedious as they require manual interaction and quality control, and are hence prone to operator bias. Semantic segmentation with neural networks has made tremendous progress in recent years due to advances in convolutional neural networks.

This master thesis is a continuation of a specialization project completed during the fall semester of 2020. The project studied machine learning approaches that automatically creates precise segmentations of micro-CT images of a bead pack filled with oil and water. One deep learning model, one conventional machine learning model, and one grayscale thresholding algorithm were implemented and compared. These are a 2D U-Net, a Support Vector Machine and Otsu segmentation, respectively. The first two models were trained with manually annotated ground truth images. To make this thesis self contained, the results from the specialization projects are included in the results section.

The deep learning model from the specialization project has some issues. One of them is the ability to correctly segment small cross-sections of the solid particles. A lot of noise and blur often surrounds these cross-sections, as may be seen in Figure 1.1. This makes the images very hard for the operator to annotate because it is difficult to determine where the boundaries between the solid and the pore space are. The annotations will therefore be biased by the operator and not consistent.

This is the main issue with deep learning approaches; in order to create a good model, one needs an extensive dataset with precise, unbiased ground truths. These are hard to come by, as both manual pixel-wise annotations and traditional multi-step segmentations are almost impossible to create without operator bias. According to Wang, Shabaninejad, et al. (2021), the issue of segmentation user

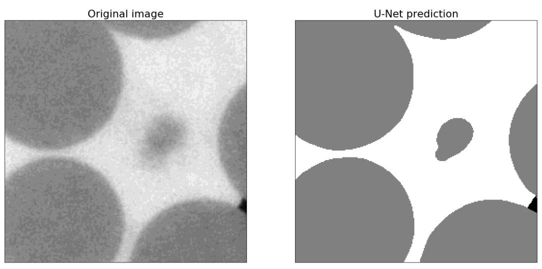


Figure 1.1: Segmentation from the specialization project. The left image is the original image, while the right image is the result from U-Net. The model struggles to segment small cross sections of the solid particles.

bias is an as of yet unaddressed issue in Digital Rock Physics.

This is the reason for the change in methodology. Instead of creating ground truths to existing micro-CT images, the opposite is done. Ground truths are created and processed to resemble micro-CT images. This way, the ground truths will be objective and pixel perfect and allow for 3D images, which was not possible to annotate with the previous methodology. The use of 3D images allows information to be extracted from the depth dimension of the images, which should improve the segmentation of the small cross-sections mentioned above.

Synthetic images also allow for the fluid distribution in the pore space to be precisely segmented. In the last years, much research has been conducted on multi-mineral segmentation and pore space segmentation, but according to Wang, Blunt, et al. (2021), work on estimation of fluid phase distribution using Deep Learning techniques has not yet been published.

In this thesis, the ground truths are created by generating sphere packs that contain oil and water. Then processing is done to transform the ternary ground truth images into micro-CT images. Data augmentation is implemented to extend the dataset and add variance, before it is used as input to the deep learning training. Three versions of U-Net is implemented in this study; a 2D U-Net, a 3D U-Net and a 2D MultiRes U-Net. U-Net is a fully convolutional neural network proposed in Ronneberger et al. (2015). By using a contracting and expanding path combined with connections between the paths and a large number of feature channels, the U-Net is capable of producing precise segmentations with good localization.

The research question in this thesis is as follows: *Is it possible to create a model that can be used to automatically digitize the fluid phases and matrix of micro-CT images precisely*? Due to the difficulties in obtaining datasets with precise ground truth segmentations, a sub-question is: *Is it possible to achieve quality segmentations on authentic micro-CT images with models that are trained on synthetic images*?

In Chapter 2, background material that is needed for later sections is introduced and explained. Chapter 3 shows details of the theoretical foundations of sphere pack generation, morphological invasion and deep learning with convolutional neural networks. Chapter 4 describes the methodology of the study and explains how the data is created and processed, in addition to how the models were implemented, trained and evaluated. In Chapter 5 the results from the segmentation of the testing set and the authentic micro-CT images are presented and discussed. Chapter 6 concludes the study, and Chapter 7 presents ideas for further work.

Because this master thesis is a continuation of the specialization project, some sections are reworked and expanded versions of the corresponding sections in the specialization project. These sections include most of the Background chapter, Section 3.3, 3.4, 3.6 and 3.7 in the Theory chapter, and Section 4.1, 4.2, 4.3, 4.5, 4.6.1 and 4.7 in the Methodology chapter.

Chapter 2

Background

In this section, a brief introduction to the concepts needed as background for the theory and methodology sections is given.

2.1 Digital Rock Physics

The purpose of Digital Rock Physics, as described by Andrä et al. (2013), is "to discover, understand and model relations between remotely-sensed geophysical observables and in-situ rock properties". Traditionally, rock physics models were based on empirical relations from lab measurements, or on theoretical models based on idealized microstructures calibrated with measurements. These models are often oversimplified with regards to the geometry they represent and the physical interactions within them. Cross-property analyses are challenging, as the various rock property models represent the rock microstructure differently.

With micro-CT images of core samples, the pore geometry is represented in high resolution. These images may be segmented to be used as input in a range of advanced simulations of the physical processes. Among them are fluid flow to quantify permeability, electrical current flow to quantify resistivity, and elastic deformation to quantify elastic moduli and the elastic wave velocity (Andrä et al. 2013). Because the features are captured in a non-invasive and non-destructive manner, the core samples can still be analyzed in the lab afterwards (Wildenschild et al. 2013).

A prerequisite for digital rock physics to be successful is the precision and efficiency of the segmentation of the micro-CT images. Segmentation is the process of separating the images into multiple sub-volumes and assign different labels to different sub-volumes. A common way to do this is described in Section 2.3.

A challenge in developing high-quality digital rock models has been the limited access to datasets that include unbiased ground truths that can be used for training supervised models and benchmarking. Often, the researchers need to create their own ground truths, which is nearly impossible to do without bias.

2.2 Acquisition of Micro-CT Images

Wildenschild et al. (2013) describes X-ray microtomography to be the most common technique to image core samples. It is able to capture information in all three dimensions in a non-destructive manner in a scale that is well suited to study the processes and variables of importance to subsurface flow in porous media.

The CT scanner consists of an X-ray source and a series of detectors. The X-rays originating from the source pass through the object, and the detectors, which are on the other side of the object, measures how much the X-ray signal has been attenuated. The detector's response to one pass through the object is called a view. Multiple views are obtained by either rotating the source and detectors around the object or rotating the object itself. An image of the scanning configuration can be found in Figure 2.1. The views from one rotation of the object are reconstructed to form a two-dimensional image called a slice. By scanning different planes of the object, i.e. different heights, multiple slices are created and construct a three-dimensional image of the object (Ketcham et al. 2001). An example of a slice from this thesis is seen in Figure 2.2.

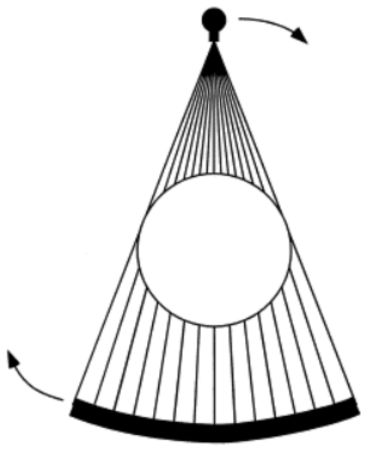


Figure 2.1: Illustration of the scanning configuration. The black object at the top represents the X-ray source, and the curved, black bar at the bottom represents the detectors. The round, white element in the middle represents the object to be scanned, and the arrows represent rotational motion. This can be caused by either rotating the object or the detectors and source (Ketcham et al. 2001).

Because rock samples do not have any limitations regarding acceptable radiation dose, industrial CT systems are of higher energy and have longer exposure times than in the medical industry, where these systems were originally developed. Therefore, smaller detectors may be used, which give higher resolution, since the reduction in a signal that follows from the reduction in surface area on the detector is compensated for by using higher X-Ray intensities or exposure times (Ketcham et al. 2001).

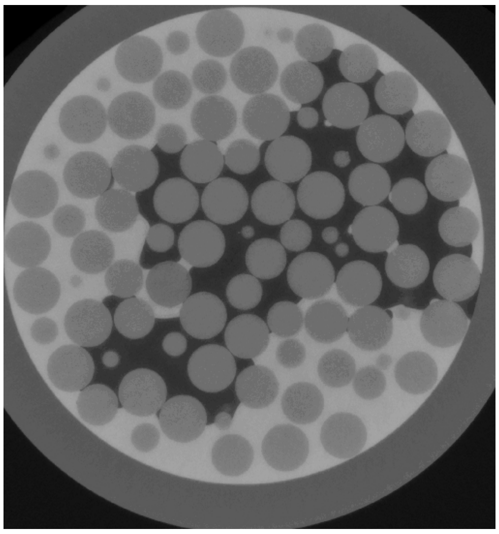


Figure 2.2: Original CT slice. The small spheres are cross-sections of the glass beads, the bright matter surrounding the beads is water and the dark is oil. The circle surrounding all the beads is the cross-section of the glass container in which the beads are placed.

2.3 Traditional Segmentation Approaches

Due to the fact that the scanned 3D objects often are more than 1000 voxels in the three directions, it is not feasible to segment the images manually, and image processing algorithms are required. The methods for segmenting the images range from simple thresholding to advanced workflows with several steps.

The typical workflow for segmenting micro-CT images of rock samples consists of spatial filtering, noise and artefact removal, thresholding, morphological operations and cluster analysis. Each step does not produce a unique result, and manual interaction and quality control are therefore needed (Andrä et al. 2013). The different steps and why they are performed are explained below in Section 2.3.1 to 2.3.5.

2.3.1 Spatial Filtering

The images often suffer from non-uniform brightness distribution where, for example, the voxels in the centre of the sample are brighter than the voxels at the edges of the sample. This is usually corrected by fitting a brightness profile that characterizes the inhomogeneity. Then the pixel values are adjusted in order to make the profile uniform (Andrä et al. 2013).

2.3.2 Noise and Artifact Removal

The images usually contain noise, and this is typically removed with the non-local means denoising algorithm, which is introduced by Buades et al. (2005). This filter uses the mean of all pixels in the image weighed by how similar they are to the pixel to be smoothed. This is shown to achieve better results and less loss of information than in the local mean methods.

Ring artefacts are caused by shifts in output from individual detectors, which causes the corresponding ray to have anomalous values and the reconstruction to form circular streaks at the greatest overlap of these rays. These rings may be removed in several ways. If the image is not yet reconstructed, vertical lines can simply be removed. On the other hand, if the image is reconstructed, it can be transformed into polar coordinates before vertical lines are removed, and then transformed back to cartesian coordinates. Yet another solution is to detect circles with radii bigger than some threshold value and exclude these circles from the analysis (Andrä et al. 2013; Ketcham et al. 2001).

2.3.3 Thresholding

Now thresholding is performed on the image to reduce the images to two or a few classes. Thresholding means assigning one class to all pixels with intensity lower than a fixed constant and assign all other pixels to a different class. Therefore, in its simplest form, it creates binary images. It can be extended to more classes by using several threshold values.

A range of studies has been conducted on how to best determine the threshold value for the segmentation. The most popular method is Otsu's method, which is described by Otsu (1979). This method selects the threshold that minimizes the intra-class intensity variance, which for two classes is the same as maximizing the inter-class variance. It does this by first creating the histogram of the image and computing the probabilities for each pixel value. Then it loops through all possible threshold values and computes the inter-class variance, and in the end, it chooses the threshold value with the maximum inter-class variance. In Figure 2.3 there is an example of an image that is segmented with Otsu's method, where the corresponding histogram and threshold value also are visualized. It is straightforward to extend the algorithm to multiple classes, and the procedure is explained by Otsu (1979).

The Otsu method has some limitations. For example, it does not yield satisfactory performance when one of the classes is much smaller than the other, the image contains noise, or different classes have similar pixel values but different textures. Because of these limitations, thresholding is often just one of many steps in a segmentation workflow.

2.3.4 Morphological Operations

Morphological operations are a set of non-linear image processing operations that process images based on the shape or morphology of features in an image. These operations rely only on the relative ordering of pixel values and not on the actual values. They are therefore well-suited operations to apply on binary images. The basic operations are dilation and erosion. Dilation works like a local maximum

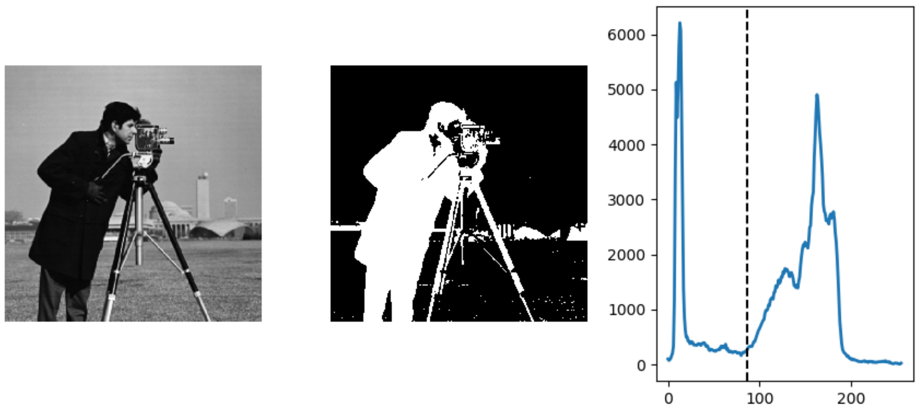


Figure 2.3: Otsu's method. From left to right: Original image, segmented image, and histogram of the original image with the threshold value visualized (*3.3.9.7. Otsu thresholding — Scipy lecture notes* n.d.).

filter, where it iterates over every pixel, uses a surrounding kernel, and assigns the maximum value of the kernel to the target pixel. Erosion is the opposite of dilation and acts as a local minimum filter. In other words, dilation expands features and erosion shrinks or removes them. By combining these operations in different ways, other operations are formed. Examples are opening, which is an erosion followed by a dilation, and closing, which is the other way around. Opening removes narrow connections, and closing removes small holes in the images (Raid et al. 2014). Visualisations of the operations is found in Figure 2.4. These operations are useful for improving the results from thresholding. Figure 2.3 shows that there are a lot of white dots on the segmented lawn. An opening operation would remove these dots and make a better segmentation.

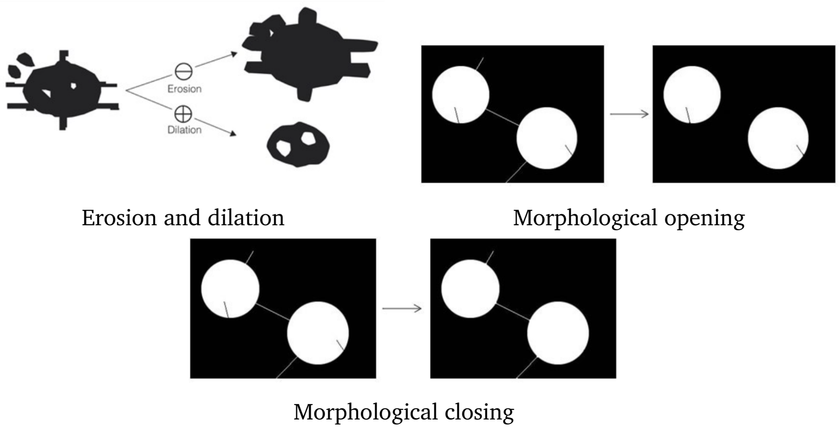


Figure 2.4: Morphological operations (Joram n.d.).

2.3.5 Cluster Analysis

After the previous operation, the image is segmented, but sometimes a cluster analysis is performed to identify and remove isolated pore regions (Andrä et al. 2013). There are multiple ways to do this, but the most common algorithm is Fuzzy c-Means (Bezdek et al. 1984). The details of this algorithm are not covered because it is not used in this study.

2.3.6 Watershed Segmentation

Watershed segmentation is a method that uses image morphology to segment images. The intuition behind the algorithm is that the image is treated like a topographic landscape with ridges and valleys, and the algorithm divides the image into adjacent drainage basins. There exist a drainage basin for each local minimum, and this basin corresponds to all of the pixels whose steepest gradient path ends up in the local minimum (Beucher et al. 1992). Watershed segmentation is further explained in Section 4.2.

2.4 Computer Vision

The purpose of computer vision is to describe the world that we see in one or more pictures and reconstruct its properties. Object detection and classification has been one of the topics that have been studied the most in computer science (Szeliski 2010).

Image classification determines which classes are present in a given image. The task of classifying images can be extended with several levels of sophistication. According to Girshick et al. (2014), the different levels are described as image classification, image classification with localization, semantic segmentation and instance segmentation. Until the rise of convolutional neural networks for this problem in 2012, this was an unsolved problem unless one had just a few classes. Szeliski (2010) claims that no one yet has constructed a system that approaches the performance level of a two-year-old child. Classification with localization is an enhancement of image classification. Here, the localization of the predicted class is specified, commonly with a bounding box. Semantic segmentation is an enhancement where one also precisely segment the boundaries of the different classes, while instance segmentation separates between different instances of the same class. The levels are visualised in Figure 2.5.

Semantic segmentation differs from image classification and image detection by labelling every pixel of the input image. This dense prediction naturally leads to a segmentation, which is what we require in this study.

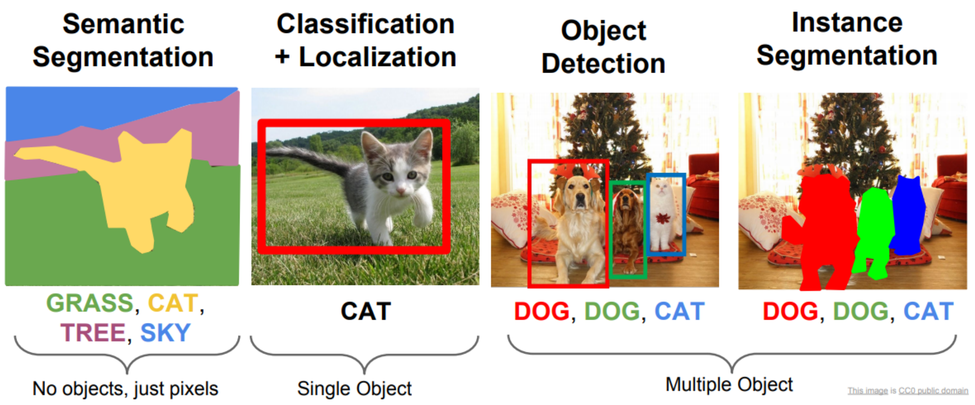


Figure 2.5: Differences between different levels of classification (Li et al. 2017).

2.5 Conventional Machine Learning

According to Jordan et al. (2015), machine learning is focused on two aspects. The first is how to construct computer systems able to learn and improve from experience. The second aspect is determining which laws of statistics, computation and information that govern all learning systems, including humans, computers and organizations. Machine learning can be split into supervised and unsupervised methods.

In supervised learning, all instances are labelled, which means that we have prior knowledge of the output value. The goal of supervised learning is to train a model that, given a sample of data and desired output, is able to approximate the relationship between the input and the output. Common use cases for supervised learning are classification and regression (Jordan et al. 2015).

Unsupervised learning does not have labelled output, and its goal is to infer the structure between data points. Typical use cases are clustering, representation learning and density estimation (Jordan et al. 2015).

2.5.1 Gradient Descent

Gradient descent is an old algorithm that was suggested by Cauchy in 1847. Because of its simplicity, it is often used within machine learning. The objective of gradient descent is to descend a slope and find a minimum, and it is, therefore, an optimization algorithm. It is iterative and starts at an arbitrary location in the search space and moves in steps in the direction with the steepest slope until it reaches a minimum (Bottou 2010).

Gradient descent is not efficient for large datasets with many features because it requires calculating the gradients of all features of all data points for each iteration. An enhancement of gradient descent that mitigates this problem is Stochastic Gradient Descent (SGD). This algorithm stochastically, i.e., randomly, picks one data point or a small batch of data points from the dataset at each iteration and calculates the direction with the steepest slope. This dramatically speeds up the computation time, and due to its simplicity and efficiency, it is widely used (Bottou 2010).

2.5.2 Support Vector Machine

The Support Vector Machine (SVM) was introduced by Cortes and Vapnik (1995). It is a learning machine for binary classification, and its basic idea is that input vectors are non-linearly mapped to very high-dimension feature spaces. The goal of the SVM is to construct a linear decision surface. An optimal hyperplane is described as a linear decision function with the maximal margin between the vectors of the two classes. The margin is visualized by the dark green dotted lines in Figure 2.6. The method creates support vectors that define the margin of the largest separation between the classes, and the decision boundary is set in the middle of the margin, as illustrated by the bright green dotted line in Figure 2.6. To accommodate for potential outliers and errors in the training set, it is common to use a soft margin. This means that the SVM is allowed to make a certain number of mistakes in the training process in order to make the margin as big as possible. After that, the other points may be correctly classified. To use the SVM to segment pictures, one first needs to extract features from the image. A common way to do this is by employing different filters to the image and recording the filter responses. Examples of filters are Gabor filters, edge filters, Gaussian filters, median filters. Additionally, Gray Level Co-occurrence Matrix (GLCM) features may be used to characterize the texture of the images.

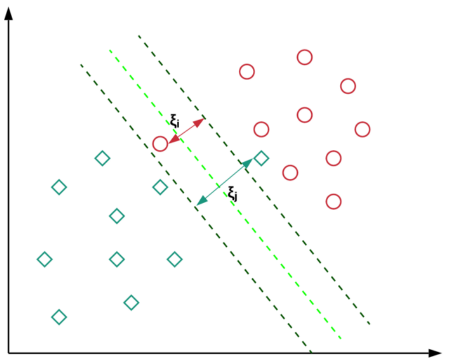


Figure 2.6: Example of decision boundary in Support Vector Machines. The soft margin, drawn with a dark green line, allows for some outliers in order to make the margin wide. The decision boundary is defined to be in the middle of the margin and is drawn with bright green (Misra 2019).

2.6 Deep Learning

Conventional machine learning techniques are limited in their ability to process data in their raw form. In the case of image segmentation, they depend on feature extraction with the use of filters. Deep learning methods use representation learning, which means that they can learn and discover patterns in raw data. The methods transform the input data in a simple but non-linear manner, making deep learning able to see patterns at a higher, more abstract level. This way, the algorithms develop deeper and deeper feature maps, and these maps are learned from the data by using a general-purpose learning procedure (LeCun et al. 2015).

According to LeCun et al. (2015), deep learning has led to significant advances in solving a range of problems. It has beaten records in image recognition, speech recognition, predicting activity of potential drug molecules, natural language understanding and many more. Because the results depend on the data and computational power, it is expected that deep learning will have continued success in the future.

2.6.1 Artificial Neural Networks

Jain et al. (1996) describes Artificial Neural Network (ANN) to be a massively parallel computing system consisting of a large number of simple processors with many interconnections. These networks are inspired by biological neural networks and try to use some "organizational" principles, which are believed to be used in the human brain.

Architecture
ANNs consist of artificial neurons and edges. Each neuron receives input and produces an output, which is sent to other neurons. The input can be features, e.g., from images or outputs from other neurons. The edges connect the neurons to each other and deliver the output from one neuron as an input to a different neuron. Both the neurons and the edges have a weight that represents its importance (Jain et al. 1996).

Usually, the neurons are organized in layers, where neurons in one layer only have connections to the neurons in the next layer. ANNs typically consists of an input layer, multiple hidden layers, and an output layer. Sometimes, each node in a layer is connected to all nodes in the subsequent layer; this is called fully connected layers. Another way to organize the connection is with pooling, where a group of neurons in one layer is connected to one neuron in the next layer. Networks that consist of only pooling connections are called feedforward networks (Jain et al. 1996). A picture of a general artificial neural network with fully connected layers is displayed in Figure 2.7.

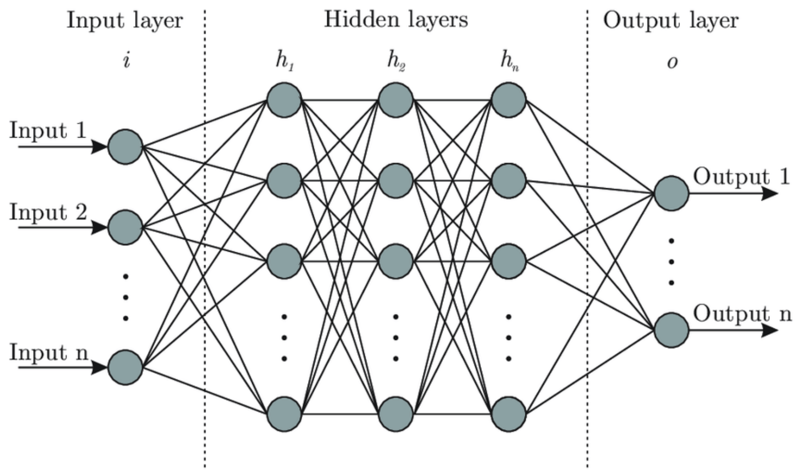


Figure 2.7: Example of an artificial neural network (Bre et al. 2018).

2.6.2 Convolutional Neural Networks

Convolutional Neural Network (CNN) are designed to process data in the form of multiple arrays, e.g. an RGB image is made up of three matrices containing the pixel values in the red, green and blue channel. The essential ideas behind convolutional networks are shared weights, local connections, pooling and the use of many layers (LeCun et al. 2015).

Convolution

Convolution is a linear operation, and a convolution between two functions produces a third function that expresses how the shape of one is modified by the other. The convolutions in CNNs are between kernels and input data or feature maps. The kernel is a small two-dimensional matrix that functions as a filter. In the convolution operation, the kernel is convolved, i.e., moved over the entire input data, and the dot product is calculated and stored in the feature map, which is the output of the convolution operation (Nielsen 2015). The discrete convolution operation is shown in Equation (2.1), where O is a single output in the feature map, I is the input data to the convolution and K is the kernel.

$$O(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.1}$$

In Figure 2.8 we see an example of how the features are calculated.

Architecture

CNNs are typically structured as a series of stages, where the first stages usually consist of convolutional layers and pooling layers. The results from all convolutions are stored in feature maps, and each element in a feature map is connected to a local patch in the previous layer, which is called the receptive field, through

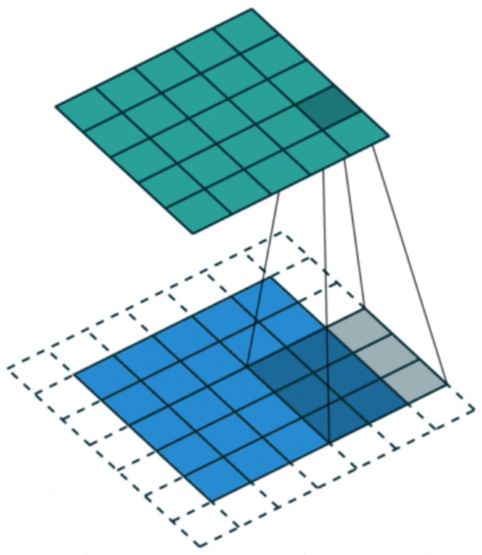


Figure 2.8: Example of convolution with input dimension 5x5, padding of size 1, kernel size 3 and stride 1. The blue pixels on the bottom are the input and the green pixels are the output from the convolution. The shaded pixel on the output (green) represents the output from one convolution operation between the shaded input area and the kernel. When the kernel has convolved over the entire input, the green output is created (Pröve 2017).

a set of weights called a feature bank. A non-linearity, e.g., a Rectified Linear Unit (ReLU), is employed on the result of this locally weighted sum. All of the features in the feature map shares the same filter bank (LeCun et al. 2015). This means that there are far fewer weights in CNNs than in ANNs. There are, of course, different filter banks for different layers in order to capture different kinds of features in the input.

The role of the pooling layers is to merge semantically similar features into one. It does this by coarsening the feature maps. The most common pooling operation is max-pooling. It finds the maximum value in a patch and uses it to represent the patch as a whole (LeCun et al. 2015).

2.6.3 Learning

Learning in deep learning is the task of updating the network architecture and connection weights in order for the network to perform a specific task. To be able to explain this, we first need to understand activation functions and backpropagation.

Activation Functions

Activation functions decide whether a neuron should be activated or not. It does this by calculating the weighted average of the input and adding a bias. Its purpose is to make the output of a neuron non-linear. The activation functions are essential to training a network since it adds non-linearities, enabling complex relationships to be formed. It is a prerequisite for backpropagation since it supplies the gradient and the error needed to update the weights and biases (Nielsen 2015). The two activation functions used in this paper are the Softmax function and ReLU.

To understand the Softmax function, it is easiest first to understand the Sigmoid function. The Sigmoid function is bounded, differentiable and shaped like an "S", as seen in Figure 2.9. Several common Sigmoid functions exist, but in machine learning, the Sigmoid function usually refers to the logistic function. This function maps any real numbers to the range (0,1). A Sigmoid function is usually placed as the last layer in the neural network because it efficiently maps the model's output to a probability score (Nielsen 2015). The logistic function is shown in Figure 2.9.

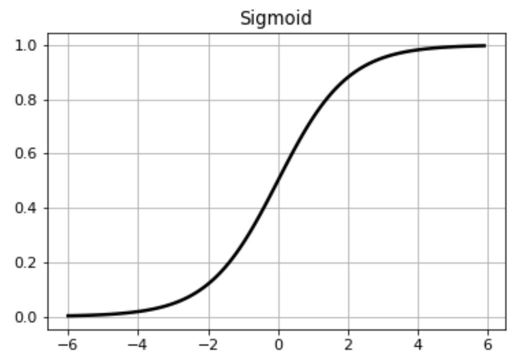


Figure 2.9: Sigmoid function, which here is the same as the logistic function.

Sigmoid functions are usually used for segmenting images with two classes, but in this paper, where there are three classes, the Softmax activation function is used. The Softmax function resembles a multi-class Sigmoid function. A useful property of the Softmax function is that the sum of the outputs equal one, so it outputs a multinomial probability distribution (Nielsen 2015).

ReLU is a piecewise linear function. It will output its input directly if it is positive, and if it is negative, it will output zero. Because of its simplicity and strong performance, it is the most common activation function in neural networks. Its linearity for positive values ensures that it preserves many of the properties that make linear models easy to optimize (Brownlee 2019). The ReLU is displayed in Figure 2.10.

Backpropagation

Backpropagation is the most used algorithm for calculating the gradients in feed-forward neural networks. The gradient is calculated with respect to all of the weights in the network. It is able to do this efficiently by starting at the end of the network and calculating the partial derivatives of the loss function for each weight with the use of the chain rule and iteratively moves towards the start of the

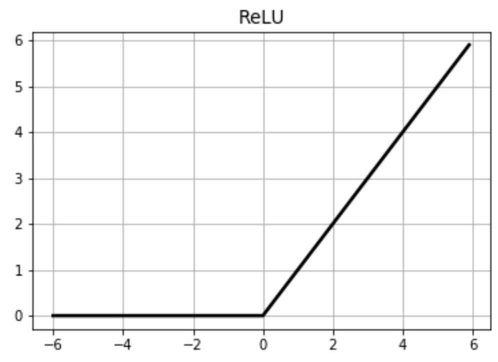


Figure 2.10: ReLu function.

network. This calculation is done for each input-output example (Nielsen 2015).

The training process starts with the network being supplied with a batch of training data. A sample from the batch is propagated through the network based on the connection weights and biases. At the end of the network, an output is created. The output is compared to the ground truth data, and the difference is calculated with a loss function. Then, backpropagation calculates the gradient of this loss function w.r.t. all of the weights in the network. The network repeats this process for all of the samples in the batch. After every sample in the batch is processed, it computes the average gradients for all of the gradients obtained by backpropagation for each weight. These gradients are input to an optimizer, e.g., gradient descent, which decides how the weights should be adjusted in order to minimize the loss function. There is a subtle difference in the learning process for ANNs and CNNs. For ANNs, the weights may be considered the relative importance for each edge in the network. For CNNs, the weights are the elements in the filter banks, i.e., the convolution kernel elements. This means that in CNNs, every layer shares the same weights, and, as a consequence, there are far fewer weights in CNNs (LeCun et al. 2015).

Other essential concepts are hyperparameters, overfitting and underfitting, data splitting, regularization, and batch normalization. These concepts are briefly explained below.

Hyperparameters

A hyperparameter is a parameter that is set prior to the learning process. Examples of hyperparameters are learning rate, which is the size of the corrective steps in the learning process, number of hidden layers in the neural network, and batch size in the learning process (Nielsen 2015).

Overfitting and Underfitting

Overfitting means that the model learns the training data to such an extent that it does not perform well when used on unseen data. It learns the random fluctuations and noise in the training data, and consequently, it will perform poorly when applied to different data (Nielsen 2015).

Underfitting occurs when a model is not trained enough. This means that it will not give a strong performance on either training data or unseen data (Nielsen 2015). Figure 2.11 visualises overfitting and underfitting.

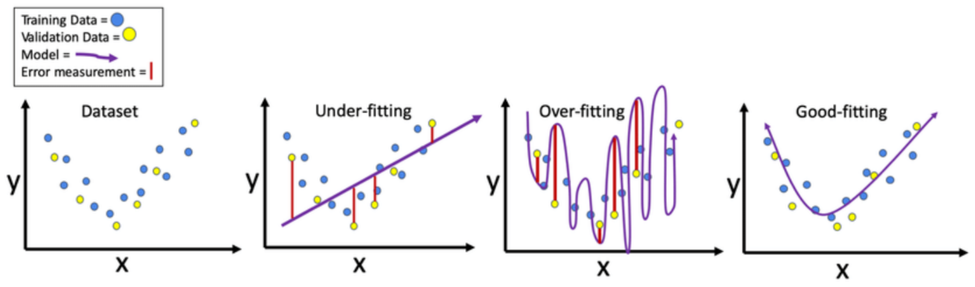


Figure 2.11: Example of underfitting, overfitting and a well-fitted model (Kiourt et al. 2020).

Data Split
In supervised learning, it is common to split the data. One typically split between a training set, validation set and testing set. The purpose of doing this is to detect and avoid overfitting. The training set is used to train the model and update the weights in the model. While the model is training, it regularly is tested on the validation set. This validation set gives an unbiased evaluation of the model while it is training and is used to update the hyperparameters in the model. It is also used for Early Stopping, i.e. to terminate the training if the model starts to overfit. The test set is used only to test the model after the training process is finished to give an unbiased evaluation of the final model (Nielsen 2015).

Regularization
Regularization is a range of techniques that reduce a neural network's complexity during training and therefore prevents overfitting. Examples of regularization are dropout, early stopping, and data augmentation, all used in this paper (Nielsen 2015).

Dropout means that there is a predefined probability that a neuron is turned off during a backpropagation. This will lead to a less complex model because there will be fewer connections in the network, which reduces the probability of overfitting.

Early stopping stops the model's training when the training error keeps going down but the validation error goes up or has stagnated. This is a sign of overfitting, and techniques for early stopping help mitigate this. Figure 2.12 illustrates an example of a graph that plots error vs the number of epochs. It shows that the model is optimally trained when the validation error is at its minimum. The model is underfitted if the number of epochs is lower and overfitted when it is higher.

Data augmentation increases the dataset by creating modified copies of the original data. This is important when using small datasets to make the model able to generalize and to avoid overfitting. Different kinds of data augmentation techniques lead to resilience against diverse data and acts as a regularizer. Some examples are described below.

Transformations augmentation - Different kinds of transformations are applied to the input image and the corresponding mask. Examples are flips, shifts, rotations, distortions or a combination of these, making the model invariant to these transformations in the input image.

Color augmentation - Brightness, contrast, hue and saturation are changed to make the model able to generalize. This will improve performance for images from different scanners or with different lighting etc.

Blur augmentation- Blur is added to the input image to make the model resilient to images of poor quality.

Noise augmentation - Noise is added to the input image to make the model resilient to this.

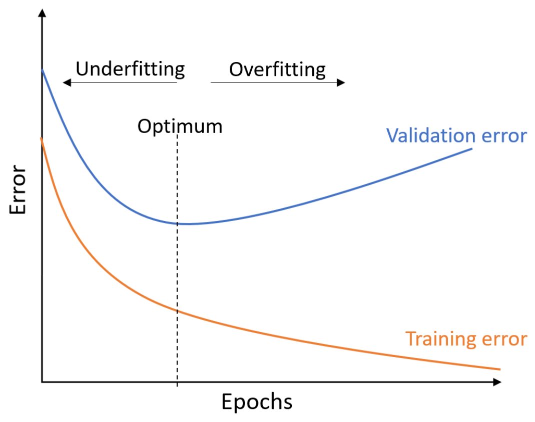


Figure 2.12: Error vs number of epochs.

Batch Normalization

Batch normalization normalizes the output of a layer by subtracting the batch average and divides it by the batch standard deviation. This can be applied to any layer in the network and improves the speed and reliability in neural networks, and have a regularization effect (Santurkar et al. 2018).

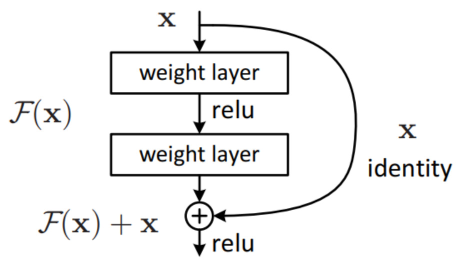

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.13: Residual learning. $F(x)+x$ is realized with a shortcut connection (He et al. 2016).

2.6.4 Residual Learning

Increasing depth and complexity for neural networks have led to increased difficulties in neural network training. One of the main obstacles is the degradation problem; When the network depth increases, the accuracy gets saturated before a rapid decline in accuracy occurs. Unexpectedly, this degradation is not caused by overfitting (He et al. 2016).

He et al. (2016) proposes deep residual learning as a solution to the degradation problem. Instead of letting each stack of layers in the network fit a desired underlying mapping, they fit a residual mapping. Let the desired underlying mapping be denoted by $H(x)$, and the stacked non-linear layers fit the mapping $F(x) := H(x) - x$. The original mapping may then be denoted as $F(x) + x$. The authors of the paper hypothesize that it is easier to optimize the residual mapping than the original, unreferenced mapping (He et al. 2016).

The mapping $F(x) + x$ may be realized by feedforward networks with shortcut connections that skips one or more layers. Different operations can be added in the shortcut connection, but in the paper, they simply use identity mapping, whose outputs are added to the outputs of the stacked layers (He et al. 2016). See Figure 2.13 for an illustration of the architecture of the building block.

Experiments show that networks that implement these building blocks are easier to optimize than equivalent networks without these shortcut connections. They are also able to gain accuracy by increasing the depth in the network and does not suffer from the degradation problem (He et al. 2016).

2.7 Related research

In this section, some related research on segmentation of images of porous media is presented.

Wang, Blunt, et al. (2021) describes techniques and common neural network architectures used in digital core analysis and reviews recent studies to demonstrate all of the tasks where Deep Learning is beneficial. They also shed light on the challenges and limitations of deep learning. The review concludes that CNNs used for segmentation have improved the issues of inherent bias in the segmentation, the high sensitivity of physical modelling, and the increased difficulty in segmenting noisy images. However, an issue that plagues the segmentation CNNs is the lack of available datasets over a broad range of samples. They also emphasize that the results achieved with deep learning in pore-scale imaging are still in the early stages of research.

The authors of the review above also published Wang, Shabaninejad, et al. (2021). They implement and test four CNN architectures for 2D and 3D segmentations in ten configurations. Manually segmented micro-CT images of sandstone are used as ground truth, and they achieve a high voxelwise accuracy above 99%. They use downstream analysis to validate the physical accuracy. These measures show high variance with permeabilities and connectivities orders of magnitude off. They also introduce a new network, which is a combination of a U-Net and ResNet. It is, however, not the same network as the MultiRes U-Net, which is used in this thesis. The hybrid network in this paper uses the original skip connections instead of the Res paths, which is explained in Section 3.5. It also differs in the sense that they operate on authentic data, while this thesis works on synthetic data.

Kjerland (2017) implements a three-dimensional convolutional neural network called DeepMedic, which was originally intended for brain tumor segmentions. They train the network for three different use cases, coronary segmentation, brain tumor segmentations from MRI images and digital rocks segmentations. They segment the digital rocks into three classes; grain, multi-phase and pore.

Andresen et al. (2019) uses three different convolutional neural networks to segment micro-CT images of Bentheimer, Berea and Carbonate sandstone obtained from Petricore. The ground truth segmentations were generated by combining the segmentations produced by five operators, which used a multi-thresholding algorithm. They use one enhancement of U-Net, where the down-sampling convolution blocks are replaced with residual down-sampling blocks. They also use a version of a pyramid network, which resembles U-Net, but there are some key differences: The skip connections are modified with a 1x1 convolution operation, each layer in the upsampling path makes a prediction, and all of the predictions are concatenated in order to make a segmentation. The third model, which they call Hybrid Digital Rock Network, is an enhancement of the first, where they include multiple adjacent channel inputs in order to gain information of the neighbouring pixels in the depth dimension. The last network is able to outperform the DeepMedic network's performance in the work of Kjerland (2017). The authors believe that a 3D approach will be better, but currently, the size of 3D patches are limited by hardware capabilities.

A major caveat of the studies mentioned above is the quality of the ground truth data. They use manually segmented images as ground truths, and these will be affected by the inherent user bias. Therefore, this study differs from the ones mentioned above by using synthetic data, which removes the user bias from the training process. It also differs by segmenting the fluid distribution in the pore space, which according to Wang, Blunt, et al. (2021) has not been done in any published works on deep learning. Among else, this makes it possible to extract contact angles directly from the segmented images.

Chapter 3

Theory

This section will describe the foundation of the sphere pack generation and then dive into the details of the deep learning models used in this thesis.

3.1 Sphere Pack Generation

This thesis uses vials filled with glass beads and fluids as an approximation for porous media. There are several benefits of this. The glass beads are of exact spherical shape and diameter, and the wetting conditions can be modified as desired. The exact spherical shape of the glass beads makes it possible to generate synthetic images of spheres with given diameters, which may be used to either replace or expand existing datasets of micro-CT images of glass beads.

In semantic segmentation, it is a great advantage to use synthetic data. Firstly, the process of creating ground truths to 3D data is exhaustive. Secondly, the results from these manual or semi-automatic annotations are often not precise and are biased by the person that made the annotations. The synthetic images are created the other way around. First, the ground truth is created, and then it is processed to resemble an authentic image. The process of imitating authentic images is, of course, a challenge. However, by using a range of data augmentation techniques, the resulting images could possibly cover image output from a wide range of scanners. In other words, the variety in the images is beneficial to the generalization of the model since different CT scanners give images with different properties. Training on images from just one or a few scanners could make the resulting model unfit for other scanners.

Random arrangements of spheres are useful to model systems for a range of physical and engineering problems. Sphere packs are crucial in determining the macroscopic granular nature of porous materials and liquid flow characteristics (Bezrukov et al. 2002). There exist numerous algorithms for generating sphere packs. Two of them, force-biased sphere packing and the Lubachevsky-Stillinger algorithm, are used in this thesis and presented below.

3.1.1 Force-Biased Sphere Packing

The theory in this section is based on Bezrukov et al. (2002). The force-biased algorithm is part of the family of algorithms called Collective Rearrangement. The algorithm is able to generate packings with a wide range of densities. The algorithm starts with a set of N spheres, $\{b(\mathbf{r}_i, d_i)\}$ with the centers, $\mathbf{r}_i$ uniformly distributed in a parallelepipedal container, and diameters d_i determined by a prescribed distribution function. It is, therefore, common with overlapping in the initial configuration. For convenience, it is assumed that the minimum diameter is 1.

The algorithm iteratively improves the initial configuration. In this iteration, each sphere are assigned two diameters: the inner diameter $d_i^{in} = d_i d^{in}$ and the outer diameter $d_i^{out} = d_i d^{out}$. The value of the d^{in} factor is chosen so that there are no overlaps in the system and exactly two spheres are in contact. The outer diameter factor d^{out} is initially calculated from the equation

$$V_{V,nom} = \frac{\text{volume of } \sum_{i=1}^{N} b(x_i, d_i^{out})}{\text{volume of container}} = \frac{\frac{\pi}{6}(d_{out})^3 \sum_{i=1}^{N} d_i^3}{\text{volume of container}} \tag{3.1}$$

where $V_{V,nom}$ is given as an input parameter.

The algorithm attempts to reduce the number of overlaps with the following two operations:

- push apart overlapping spheres by choosing a new position
- gradually shrink the spheres by reducing d^{out}

The pushing operations usually increase d^{in} whereas shrinking reduces d^{out}. When $d^{in} \geq d^{out}$ the algorithm is stopped. Then a system of non-intersecting spheres is obtained with diameters proportional to the initial diameters, d_i.

In every step of the iteration, the algorithm defines a "repulsion force", F_{ij}, between each pair (i, j) of overlapping spheres:

$$F_{ij} = \rho \mathbf{1}_{ij} p_{ij} \frac{\mathbf{r}_j - \mathbf{r}_i}{\left\|\mathbf{r}_j - \mathbf{r}_i\right\|} \tag{3.2}$$

where p_{ij} is a potential function, ρ is a scaling factor and $\mathbf{1}_{ij}$ is equal to 0 if $b(\mathbf{r}_i, d_i^{out}) \cap b(\mathbf{r}_j, d_j^{out}) = \emptyset$ and equal to 1 otherwise.

The new position for sphere i is given by

$$\mathbf{r}_i := \mathbf{r}_i + \frac{1}{2d_i} \sum^{j \neq i} F_{ij} \tag{3.3}$$

This ensures that large spheres move slower than the small spheres. It is non-trivial to choose a suitable potential function. For the case of equal spheres, as is the case in this thesis, the potential function may be proportional to the volume of intersection of the overlapping spheres i and j, as given by

$$p_{ij} = (\left\|\mathbf{r}_i - \mathbf{r}_j\right\|^2 - (d^{out})^2) \tag{3.4}$$

The shrinking operation in the iteration reduces the outer diameter,

$$d^{out} := d^{out} - (\frac{1}{2})^{\delta} \frac{d^{out}_{st}}{2\tau} \tag{3.5}$$

where δ is the integer part of $-log_{10}(V_{V,nom} - V_{V,act})$, where $V_{V,act}$ is the current volume fraction of the spheres, d^{out}_{st} is the start value of d^{out} given in Equation (3.1), and τ is the contraction rate of the outer diameter. τ controls the execution time of the algorithm and influences the final packing density.

3.1.2 Lubachevsky-Stillinger Sphere Packing

The Lubachevsky-Stillinger algorithm was proposed in Lubachevsky et al. (1990) and all of the information presented in this section stems from that article. It is stated that "its most general version concerns the inquiry of non-overlapping arrangements of D-dimensional spheres, D=1, 2, 3, ..., N confined to a "rectangular" region Ω_D of size $L_1 \times L_2 \times ... \times L_D$".

The events in the algorithm are processed in an event-driven fashion and not in a time-driven fashion. This means that there is no calculation required on maintaining the positions and velocities of the particles between the collisions.

The algorithm starts by placing N points randomly within the unit cell Ω_D. The points are assigned random velocities, whose components are independently distributed at random between -1 and 1. These points will move at their initial velocity along a straight line until a collision happens.

At $t = 0$ the points are infinitesimal, but they begin to grow at a common rate into elastic D-spheres with diameters given by some function $a(t)$. It is required that $a(0) = 0$ and that $a(t)$ is continuous and nondecreasing with $a \rightarrow \infty$ for $t \rightarrow \infty$. Therefore, the probability for collisions increases with time.

The objective is to sample initial configurations and velocities statistically using a common $a(t)$ but generate several starting points. For all starting points, time is allowed to progress until the system jams up, and the collision rate in principle must diverge. The final packing depends on the combination of initial starting points, their initial velocities, and the time-dependent collision diameter $a(t)$. The weights $w(q)$ with which the packings q are sampled still depend on $a(t)$ after averaging over the initial conditions.

Since the diameters of the spheres changes with time, the collision dynamics do not conserve energy, and must be altered. Suppose particle 1 and 2 have velocities $\mathbf{v}_1$ and $\mathbf{v}_2$ respectively before the collision. As is shown in Figure 3.1, these velocities are expressed as a sum of the parallel and transverse velocity related to the line of centers:

$$\mathbf{v}_1 = \mathbf{v}_1^{(p)} + \mathbf{v}_1^{(t)} \tag{3.6}$$

$$\mathbf{v}_2 = \mathbf{v}_2^{(p)} + \mathbf{v}_2^{(t)} \tag{3.7}$$

We thus have that $\mathbf{v}_i^{(p)} \cdot \mathbf{v}_i^{(t)} = 0$ and $\mathbf{v}_i^{(t)} \cdot (\mathbf{r}_2 - \mathbf{r}_1) = 0$, i=1,2. The transverse velocities remains unchanged after a collision, but the parallel components are

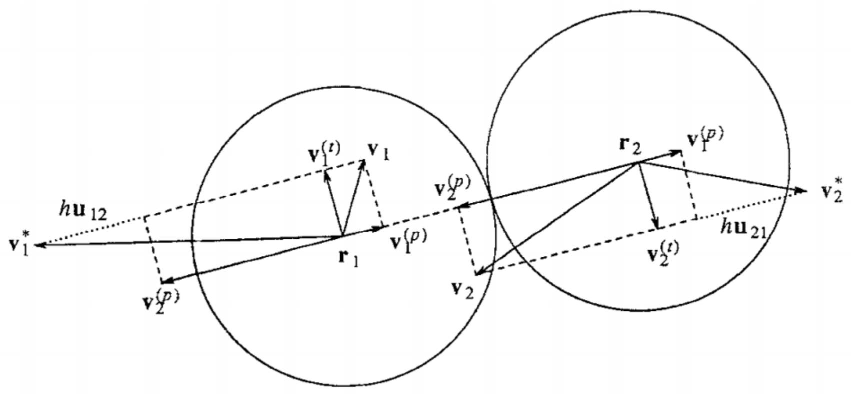


Figure 3.1: Pair collision dynamic for expanding spheres. Velocities are expressed as a sum of parallel and transverse velocities to the line of centers (Lubachevsky et al. 1990).

modified with an additive h. Assume that $\mathbf{v}_1^*$ and $\mathbf{v}_2^*$ are the velocities right after the collision, we have that

$$\mathbf{v}_1^* = [\mathbf{v}_2^{(p)} + h\mathbf{u}_{12}] + \mathbf{v}_1^{(t)} \tag{3.8}$$

$$\mathbf{v}_2^* = [\mathbf{v}_1^{(p)} + h\mathbf{u}_{21}] + \mathbf{v}_2^{(t)} \tag{3.9}$$

where $\mathbf{u}_{12}$ is the unit vector:

$$\mathbf{u}_{12} = (\mathbf{r}_1 - \mathbf{r}_2)/|\mathbf{r}_1 - \mathbf{r}_2| = -\mathbf{u}_{21} \tag{3.10}$$

If $2h$ exceeds the diameter growth rate $a'(t_c)$, collisions occur at discrete isolated times.

The difference in kinetic energies after and before a collision is proportional to

$$\frac{1}{2}(|\mathbf{v}_1^*|^2 + |\mathbf{v}_2^*|^2 - |\mathbf{v}_1|^2 - |\mathbf{v}_2|^2) = h(\mathbf{v}_1^{(p)} - \mathbf{v}_2^{(p)}) \cdot \mathbf{u}_{21} + h^2 \tag{3.11}$$

A requirement for collisions is

$$(\mathbf{v}_1^{(p)} - \mathbf{v}_2^{(p)}) \cdot \mathbf{u}_{21} > 0 \tag{3.12}$$

That means that the difference in kinetic energy shown in Equation (3.11) is strictly positive since $h > 0$. Therefore, the total kinetic energy in the system increases with each collision.

Lets assume a constant diameter growth rate

$$a(t) = a_0 t \qquad (a_o > 0) \tag{3.13}$$

This means that jamming will occur in a finite time. The sampling weights $w(q)$ for the jammed disk packings depend on the choice of a_0. Typically, jamming occurs in an irregular structure if a_0 is large compared to the mean initial particle

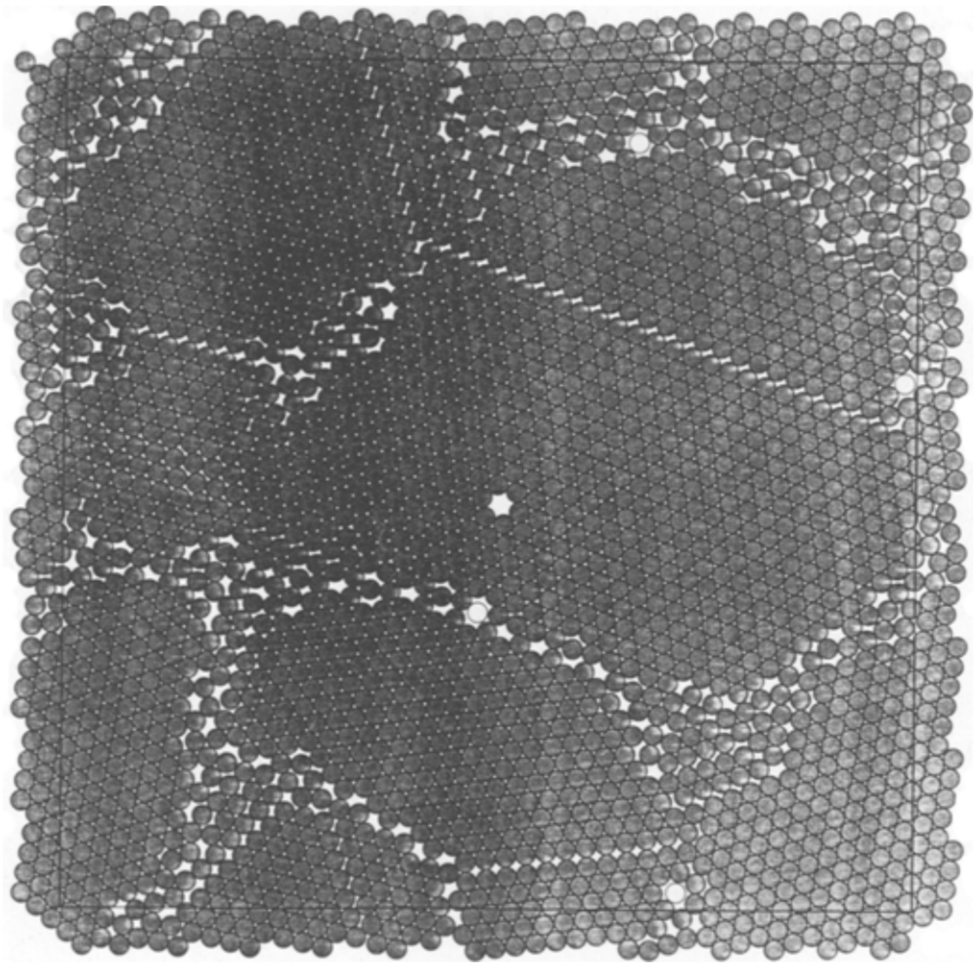


Figure 3.2: Illustration of a packing with rattlers, monovacancies and shear fractures. The rattlers are the non-shaded disks, a monovacancy may be found in the center of the image, and the shear fractures are the visible linear shears in the packing (Lubachevsky et al. 1990).

speed. For small a_0, the extended collision dynamics makes the system rearrange into a nearly crystalline packing. Therefore, the sampling weights will depend on a_0 in a way that makes the mean covered area fraction a monotonically increasing function of a_0.

The Lubachevsky-Stillinger algorithm produces random disk/sphere packings with characteristics unlikely to emerge from the standard sequential construction procedures. These characteristics include monovacancies, rattlers, and linear shear fractures. Rattlers are trapped, but unjammed grains. The linear shear fractures preserve bond orientational order, but disrupt translational order within the crystalline grains. All these characteristics are illustrated in Figure 3.2.

3.2 Morphological Invasion

To simulate the primary drainage process, a pore-morphology-based simulation may be used.

The following explanation of the process is from Berg et al. (2020), and the original procedure is from Hazlett (1995) and modified in Hilpert et al. (2001).

In this section, perfect wetting is assumed, with $cos(\Theta) = 1$. It is, however, possible to extend the procedure to other contact angles, as will be presented in Section 4.4.4. The method also assumes that the two principal curvatures of the fluid-fluid surface are equal and not infinite. This leads to errors in the Laplace pressure which especially affects the wetting phase pendular rings between spherical grains. However, the efficiency and accuracy for most of the drainage process outweigh the errors.

The method is based on morphological erosion and dilation, which are explained in Section 2.3.4. Initially, the pore space is filled with fluid 1, which is the wetting phase. One side of the porous media is connected to a reservoir of fluid 2. In this thesis, fluid 1 is water, and fluid 2 is oil.

For a subset of the pore space, $X \subset \Omega$, $C(X)$ is defined as the part of X that is connected to the side-plane connected to the fluid 2 reservoir.

The corresponding curvature r_c from a pressure difference Δp is given by the Young-Laplace equation $r_c(\Delta p) = 2/(\sigma \Delta p)$. The portion of the pore space filled with oil at pressure Δp is given by

$$C(\{\Omega \ominus S[r_c(\Delta p)]\} \oplus S[r_c(\Delta p)]), \tag{3.14}$$

where $S(r)$ is a sphere with radius r, $\ominus$ is erosion and $\oplus$ is dilation. $r_p(x)$ is the smallest value r so that $x \in C(y \,|\, d_t(y) < r)$.

To find the fluid distribution at different pressures, the distance transform d_t is calculated, which gives the Euclidean distance between any point $x \in V$ and the closest point in the solid phase $s \in V \setminus \Omega$.

$$d_t(x) = min\{\|x - s\| \,|\, s \in V \setminus Q\} \tag{3.15}$$

Then, the critical path radius $r_p(x)$ is calculated for a path from the inlet to point x. That means the minimal distance d_t along the path with the largest minimal distance:

$$r_p(x) = max\{min\{d_t(y) \,|\, y \in S\} \,|\, S \in \mathbb{S}\}, \tag{3.16}$$

where $\mathbb{S}$ is the set of paths from x to the inlet. This equation is solved for r_p with a modified version of Dijkstra's algorithm.

Now there are two spheres spanned by the value of the distance transform and the critical radii. A point x is assigned to the largest sphere covering it as

$$\begin{aligned} c_{dt}(x) &= max\{d_t(y) \,|\, \|x - y\| < d_t(y)\}, \\ c_{rp}(x) &= max\{r_p(y) \,|\, \|x - y\| < r_p(y)\}. \end{aligned} \tag{3.17}$$

$c_{dt}(x)$ and $c_{rp}(x)$ can be related to the morphological operations:

$$\begin{aligned} \{x \,|\, c_{dt}(x) \leq r\} &= [\Omega \ominus S(r)] \oplus S(r) = X(r), \\ \{x \,|\, c_{rp}(x) \leq r\} &= C[X(r)]. \end{aligned} \tag{3.18}$$

Equation (3.18) relates the calculated values for c_{rp} to the morphological description $C[X(r)]$ for fluids at different pressures, as given by Equation (3.14).

We then have that $c_{rp}(x)$ is proportional to the pressure differences when fluid 1 is displaced by fluid 2 at point x. To illustrate the process, consider a simple 2D example. $X(r) = [\Omega \ominus S(r)] \oplus S(r)$ for different radii values r is shown in Figure 3.3a. We have that $X(r) = \{x \,|\, c_{dt}(x) \leq r\}$, where c_{dt} is given by Equation (3.18). For larger subradii r, the subset $X(r) \subset \Omega$ is not connected to the top-side of the figure, which is considered to be the inlet. Therefore, $C[X(r)]$ will be an empty set for larger values of r, which is seen in Figure 3.3b. This simulates the pressure difference needed for invading different parts of the porous medium.

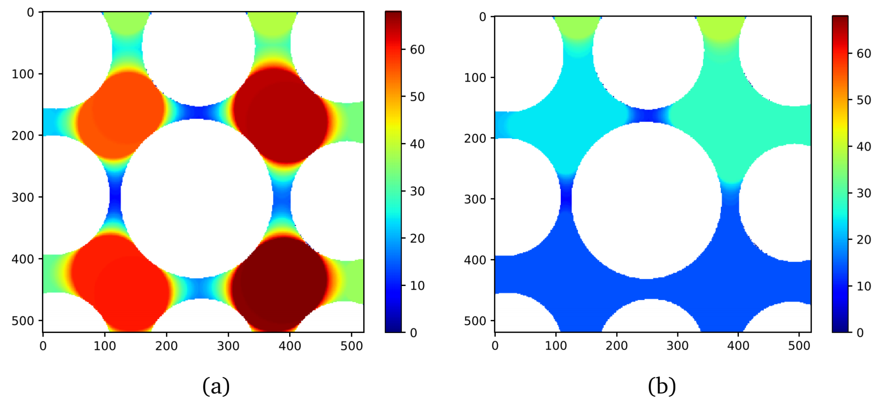


Figure 3.3: (a) - Two-dimensional example. Pore space Ω in colors and solid phase $V \setminus \Omega$ in white. The figure shows $X(r) = [\Omega \ominus S(r)] \oplus S(r)$ for different radii values r, which are shown in the color bar. This is equivalent to c_{dt}. **(b)** - The connected part $C[X(r)]$ of the subsets X(r) shown in the left image. This is equivalent to c_{rp} (Berg et al. 2020).

3.3 Fully Convolutional Network (FCN)

Fully convolutional networks for semantic segmentation was introduced by Long et al. (2015). When it was introduced, it exceeded the state-of-the-art in semantic segmentation. The authors adapted contemporary classification networks into fully convolutional networks, added skip-connections, and performed an upsampling operation that enabled them to predict each pixel in the input image.

Architecture

The network of the original fully convolutional network proposed by Long et al.

(2015) is displayed in Figure 3.4. It starts with two convolutional operations followed by a max pool operation and repeats this pattern with operations four times. The second last operation is a 1x1 convolution that predicts a class for each pixel in the last, coarse feature map. The final operation is an upsampling that upsamples the prediction to the original input resolution. In order to improve the precision of the upsampling, feature maps from previous layers in the network are concatenated to the final convolutional layer, before the final upsampling and prediction. These connections are called skip-connections and refine the spatial precision of the output because the earlier convolutional layers capture finer details in the image than the coarser, later layers. By combining the fine and coarse layers, the model is able to make local predictions that respect the global structure. Long et al. (2015) implements multiple versions of the networks with differing numbers of skip connections. Their performance is visualized in Figure 3.5

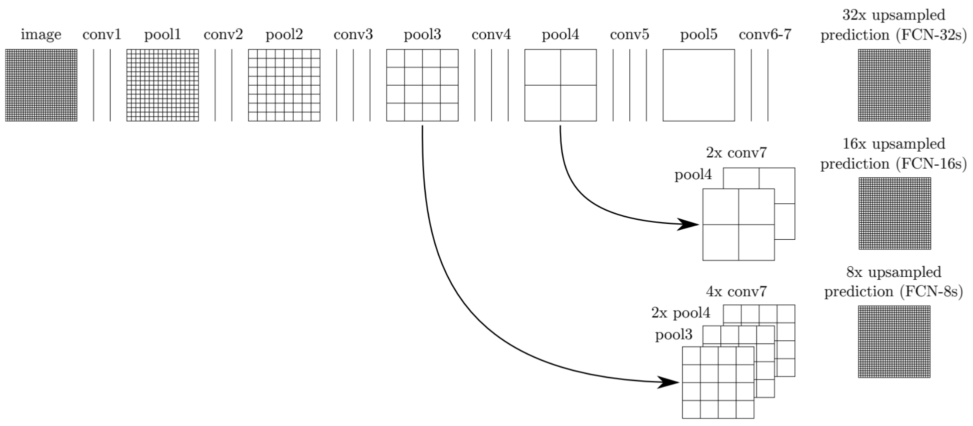


image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6-7
32x upsampled prediction (FCN-32s)
2x conv7
pool4
16x upsampled prediction (FCN-16s)
4x conv7
2x pool4
pool3
8x upsampled prediction (FCN-8s)

Figure 3.4: Fully convolutional network architecture (Long et al. 2015).

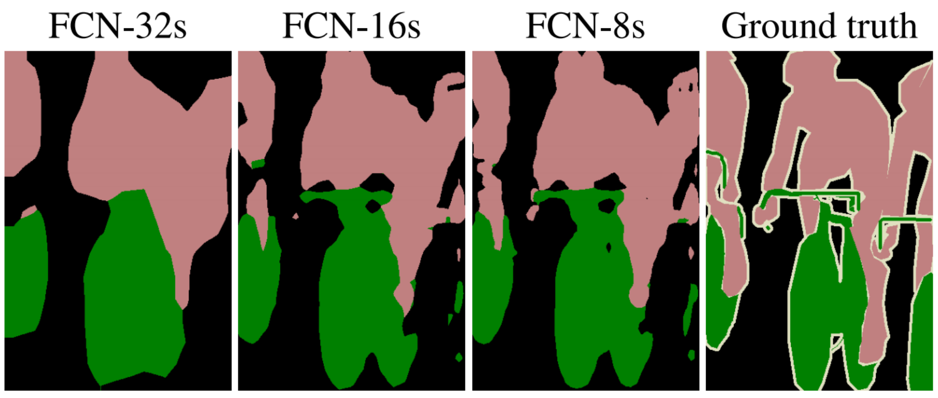


Figure 3.5: Fully convolutional network results (Long et al. 2015).

Upsampling
The upsampling operation is often referred to as deconvolution, but this is not mathematically true. It is rather a transposed convolution, or sometimes as simple as a nearest neighbour interpolation. The transposed convolution may be thought of as padding the pixels that are going to be upsampled with zeros and computing the convolution the same way as described in Section 2.6.2. The padding depends on the size of the kernel, and the number of padding layers is one less than the kernel size. Stride has a different meaning in a transposed convolution than in a regular convolution. In a transposed convolution, stride can be thought of as increasing the distance between the pixels in the input image, which means that increasing stride leads to increasing output dimensions. Figure 3.6 visualizes the transposed convolution with one and two strides. It is important to note that the transposed convolution does not recreate the input to a convolution like a deconvolution. It just recreates the spatial dimensions (Xiang 2017).

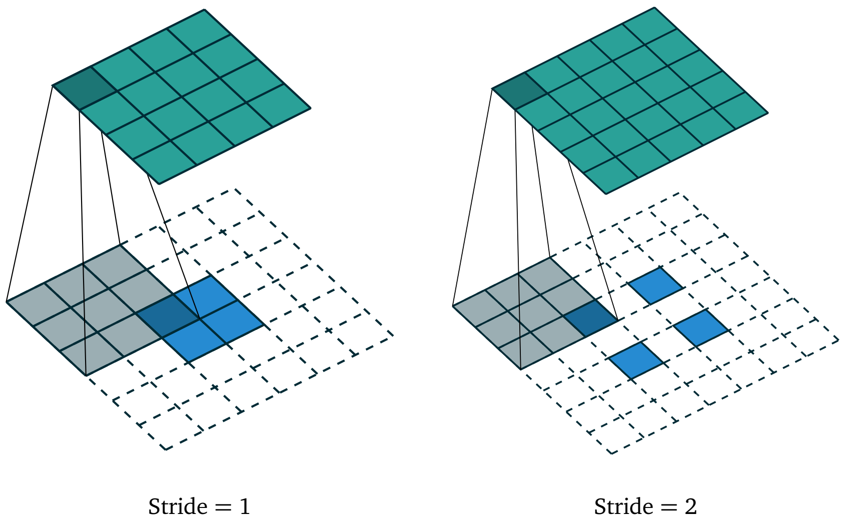


Figure 3.6: Transposed convolution. The input is colored in blue and the transparent pixels surrounding the input is the padding. The green pixels are the output created after the kernel has convolved over the input. The strides in transposed convolution correspond to increasing the distance between the input pixels (Xiang 2017).

3.4 U-Net

U-Net is an enhancement of the Fully Convolutional Network (FCN) and was proposed by Ronneberger et al. (2015).

U-Net extends the FCN with a symmetric, expanding path, where the upsampling operations are used instead of the max pool operations used in the contracting path. An image of the architecture is shown in Figure 3.7.

Contracting Path

The contracting path consists of a fixed pattern of operations. The first layer in the network is the input image. A convolutional operation is performed on the input image. To account for the loss of border pixels after the convolutional operation, the input images are first padded to make the dimension of the output the same as the input image. After every convolution, a Rectified Linear Unit (ReLU)function is applied to the output. Another convolution is performed on the output from the first convolution before a max-pooling operation is performed. This halves the spatial dimension and hence makes the feature map coarser. This order of operations, with two convolutions, each followed by a ReLU, and one max-pooling operation, is repeated until we get to the bottom of the network. The number of filters is doubled for each layer in the network. Together with the coarser spatial resolution, this allows for bigger and more complex features to be captured and learned. The first feature maps are able to capture edges with different orientations and colour. As we move down the network, these feature maps are combined and able to extract basic grids, textures and patterns. Moving even further down, these are combined to be able to extract increasingly complex features, for example, bikes or faces, depending on the content in the training images (Ronneberger et al. 2015).

Expanding Path

The expanding path is symmetric to the contracting path, and together they form the shape of a U, hence the name U-Net. The pooling operation in the contracting path is replaced by an upsampling operation. The upsampling is done with the transposed convolution, which is explained in Section 3.3. The feature maps in the contracting path are fused with the feature maps in the expanding path with skip-connections. This is simply done by concatenating the feature maps, i.e., stack them on top of each other, and then convolution is performed with these concatenated feature maps as input. As we move up the expanding path, the spatial dimensions of the feature maps increase, but the number of feature maps is halved for each layer. These skip connections combine the spatial information from the contracting path with the semantic information from the contracting path, increasing the precision of the prediction in the boundary area between the segmented classes. At the final layer, a 1x1 convolution with a Softmax activation function maps the feature maps to the desired number of classes (Ronneberger et al. 2015).

Expanding To 3D

It is straightforward to extend the network to 3D. One simply replaces all of its 2D operations with the corresponding 3D operations, and add an extra dimension for the kernel and strides.

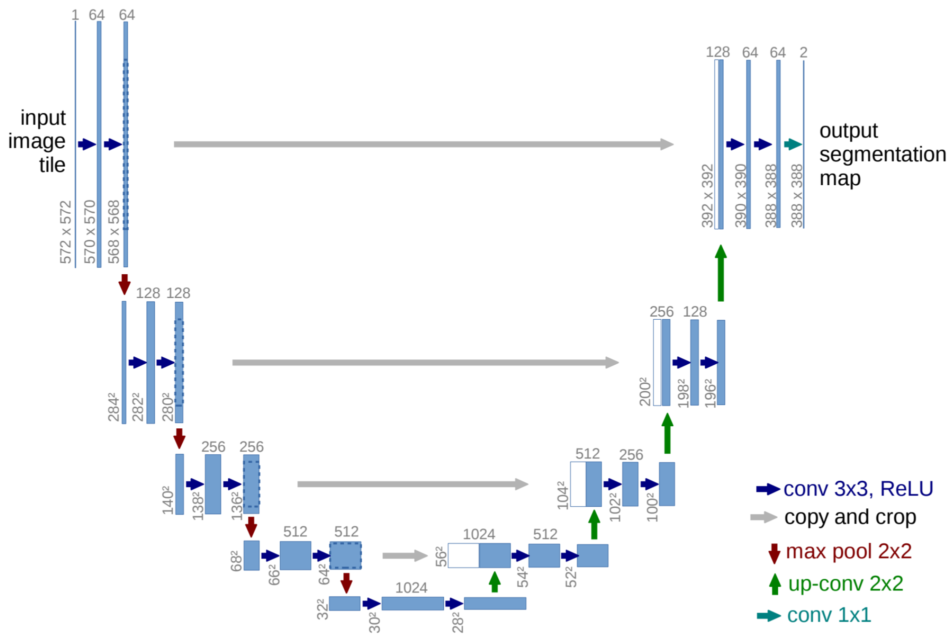


Figure 3.7: Original U-Net architecture (Ronneberger et al. 2015).

3.5 MultiRes U-Net

MultiRes U-Net is a modified version of the original U-Net and was first proposed by Ibtehaz et al. (2020). In their paper, they show that there is a slight improvement in performance on ideal images, but remarkable gains have been attained for challenging images.

The recent advancements in the field of deep computer vision led Ibtehaz et al. (2020) to try to improve the U-Net. A particular issue that the authors wanted to address was to make the network more robust to segment objects at different scales. They were inspired by the Inception architecture (Szegedy et al. 2015) that introduced Inception blocks that used convolutional layers of varying kernel sizes in parallel to inspect the objects from different scales. The results from these convolutional layers were combined and passed on to deeper layers in the network.

In each layer in the U-Net, a sequence of two 3x3 convolutions is used, and Szegedy et al. (2015) shows that this operation resembles a 5x5 convolutional operation. Therefore, Ibtehaz et al. (2020) modifies the U-Net with a multi-resolutional analysis capability by running a 3x3 and 7x7 in parallel to the 5x5 convolution. This is shown in Figure 3.8a.

The use of convolutional operations in parallel severely increases the memory requirement. This is solved by factorizing the bigger 5x5 and 7x7 convolutions using a sequence of 3x3 convolutional blocks that approximate the 5x5 and 7x7 convolutional operations. This is shown in Figure 3.8b. The output from the three

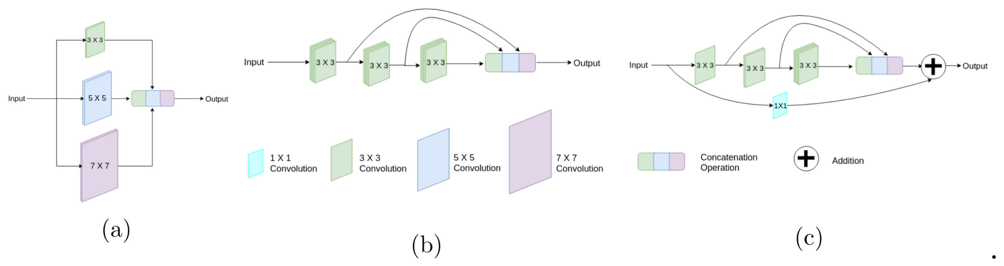


Figure 3.8: a - Inception block. Convolutional layers with various kernel sizes can inspect objects from different scales. **b** - The inception block may be factorized as a series of 3x3 convolutions. **c** - MultiRes block. Because of memory constraints, the numbers of filters in the convolutional operations is gradually increased. A residual connection is added for efficient training (Ibtehaz et al. 2020).

convolutional blocks are concatenated to extract the spatial feature from different scales (Ibtehaz et al. 2020).

The factorization of the convolutional operations greatly reduces the memory requirement, but the network is still demanding. If two convolutional operations are present in succession, the number of filters in the first one has a quadratic effect on the memory. Therefore, the number of filters is gradually increased in the convolutional operations. There is also added residual connections, as explained in Section 2.6.4, with a 1x1 convolution, which leads to increased efficiency in training. This is called a MultiRes Block, and is shown in Figure 3.8c (Ibtehaz et al. 2020).

Ibtehaz et al. (2020) claim that a flaw of the skip connections in the U-Net might be that there is a semantic gap between the two features that are merged in the skip connection. For example, the first skip connection bridges the encoder before the first pooling with the decoder after the last deconvolution operation. The features from the encoder are supposed to be lower level features as they are computed in the earlier layers of the network. On the other hand, the decoder features are supposed to be of a much higher level because they are computed at the deep layers in the network. The authors claim that "the fusion of these two arguably incompatible sets of features could cause some discrepancy throughout the learning thereby adversely affecting the prediction procedure" (Ibtehaz et al. 2020).

To alleviate the disparity between the contracting and expanding path, they propose to incorporate some convolutional layers along with the skip connections. The hypothesis is that convolutional operations should balance the semantic gap. To make the learning more efficient, residual connections are added to the skip connections. Therefore, instead of simply passing the features from the encoder to the decoder, the features are passed through a chain of convolutional layers with residual connections and then concatenated with the decoder features. These new skip connections, or ’shortcut paths’, are called ’Res paths’, and are shown in Figure 3.9

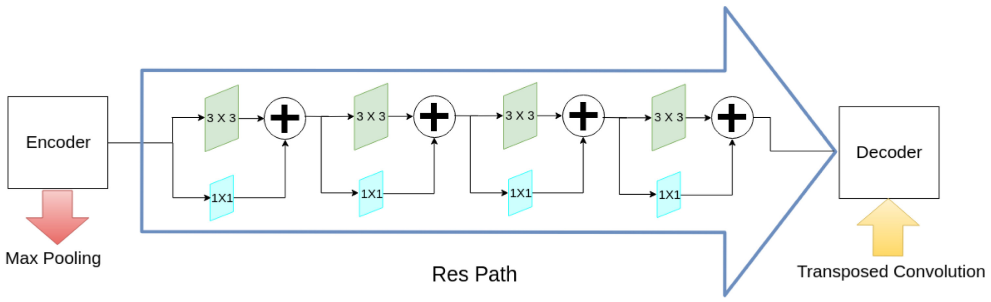


Figure 3.9: Res path. The skip connections from the U-Net is enhanced with a series of convolutions with residual connections. The convolutions are meant to bridge the semantic gap between the encoder and decoder in the network (Ibtehaz et al. 2020).

Architecture

The MultiRes U-Net replaces the two convolutional operations in the U-Net with the proposed MultiRes block. For each block, the parameter W is assigned to control the number of filters of the convolutional layers inside the block. The value of W is computed as

$$W = \alpha \times U \tag{3.19}$$

where U is the number of filters in the corresponding layer of the U-Net and α is a scalar coefficient. This allows for a convenient way to control the number of parameters and keep them comparable to U-Net. A default value of $\alpha = 1.67$ is set to keep the number of parameters slightly below the number in the U-Net (Ibtehaz et al. 2020).

As mentioned, it is beneficial to assign increasing numbers of filters in the successive convolutional operations in the MultiRes block, and therefore it is assigned $\lfloor \frac{W}{6} \rfloor$, $\lfloor \frac{W}{3} \rfloor$ and $\lfloor \frac{W}{2} \rfloor$ to the three successive convolutional operations, respectively. As in the U-Net, after each pooling or deconvolution operation, the value of W gets doubled or halved, respectively (Ibtehaz et al. 2020).

The skip connections in the U-Net are replaced by the proposed Res paths. Since the semantic gap between the encoder and decoder are expected to decrease as we move down the network, the number of convolutional operations in the Res paths are gradually reduced from 4 to 1 along the four Res paths. In order to account for the number of feature maps in the encoder-decoder, there are used 32, 64, 128 and 256 filters in the four Res paths respectively (Ibtehaz et al. 2020).

All of the convolutional layers, except for the final output layer, are activated with the ReLU activation function. The final is originally activated with a Sigmoid function, but due to that this paper studies a multiclass segmentation, a Softmax function is used instead. A diagram of the MultiRes U-Net is found in Figure 3.10.

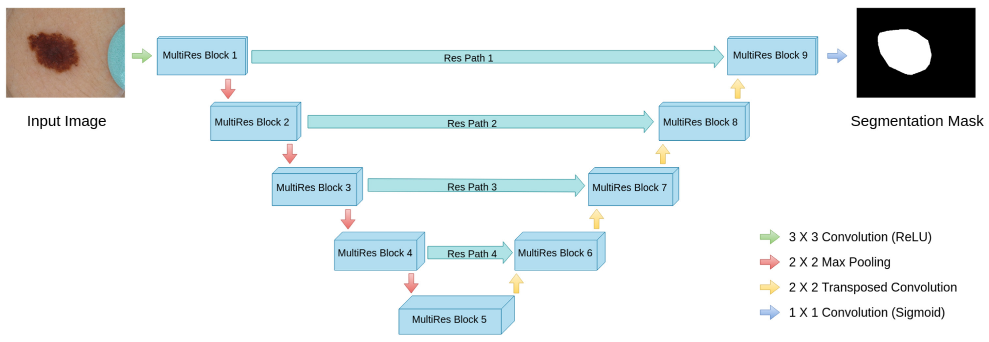


Figure 3.10: Diagram of the MultiRes U-Net architecture (Ibtehaz et al. 2020).

3.6 Deep Learning Training

Loss Function

The loss function that is minimized in all of the deep learning models in this thesis is the categorical cross-entropy. The variation used is often called Softmax Loss, since it is a Softmax activation followed by a cross-entropy loss. The Softmax activation is defined as

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \tag{3.20}$$

where s_i is the output score and C is the number of classes. The cross-entropy loss is defined as

$$h(s,t) = -\sum_i^C t_i log(f(s)_i \tag{3.21}$$

where t_i is the ground truth. Since the ground truth is one-hot, and only one element, $t_i = t_p$, in the ground truth vector is true, and hence not zero, we find that

$$h(s) = -log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right) \tag{3.22}$$

where s_p is the score of the true class. The gradient of the loss function with respect to the output layer is

$$\frac{\partial}{\partial s_p}\left(-log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)\right) = \frac{e^{s_p}}{\sum_j^C e^{s_j}} - 1 \tag{3.23}$$

for the true class. For the negative classes, it is

$$\frac{\partial}{\partial s_p}\left(-log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)\right) = \frac{e^{s_n}}{\sum_j^C e^{s_j}} \tag{3.24}$$

The derivations above are retrieved from Gómez (2018).

Backpropagation, which is explained in Section 2.6.3, is then used to find the gradient with respect to the weights in the previous layers. Then an optimizer is employed to adjust the weights in order to minimize the loss function.

Adaptive Moment Estimation (Adam)

The optimizer used in the implementations is Adam. This is an extension of Stochastic Gradient Descent (SGD), which is explained in Section 2.5.1, and is designed specifically for training neural networks. Adam computes individual adaptive learning rates for the different weights from estimates of the first and second moments of the gradients. This improves performance on problems with sparse gradients and noisy inputs. Empirical results show that Adam performs better than other stochastic optimization methods (Kingma et al. 2014). The pseudocode is shown in Algorithm 1.

Algorithm 1: Adam pseudocode

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0,1)$ : Exponential decay rates for the moment
  estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
   $m_0 \leftarrow 0$ (Initialize 1st moment vector)
   $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
   $t_0 \leftarrow 0$ (Initialize timestep)
   while $\theta_t$ not converged do
      $t \leftarrow t+1$
      $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$ (Get gradient w.r.t. stochastic objective at
        timestep $t$)
      $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$ (Update biased first moment
        estimate)
      $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$ (Update biased second raw moment
        estimate)
      $\hat{m}_t \leftarrow m_t/(1-\beta_1^{t)}$ (Compute bias-corrected first moment estimate)
      $\hat{v}_t \leftarrow v_t/(1-\beta_2^t)$ (Compute bias-corrected second raw moment
        estimate)
      $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
   end
   return $\theta_t$ (Resulting parameters)
```

In the algorithm, all vector operations are element-wise and a good default setting is $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The pseudocode is retrieved from Kingma et al. (2014).

3.7 Support Vector Machine

I have also implemented a Stochastic Gradient Descent (SGD) (Section 2.5.1) classifier that minimizes the hinge loss. This is the same as a linear SVM (Section 2.5.2).

3.7.1 Hinge Loss

The hinge loss is a convex loss function and is defined as

$$l(y) = max(0, 1 - t \cdot y) \tag{3.25}$$

where y is the prediction and t is the target for the prediction. The loss function is visualized in Figure 3.11. The dotted line is drawn at x=1. A positive distance from the boundary means that an instance is correctly classified, and a negative means it is incorrectly classified. Positive distances above one are not penalized, but positive distances below one are penalized more the smaller the distance is. Wrongly classified instances are penalized more the bigger the distance from the boundary is. Because the correctly classified instances with small distances to the boundary are penalized, the loss function tries to make a margin, with width = 1 on either side of the boundary, i.e., the hinge loss aims to maximize the decision boundary between the two classes that are going to be classified. Depending on the regularization term, which is explained below, the margin will allow for some outliers to be incorrectly classified (Crammer et al. 2001). The margins are seen in Figure 2.6.

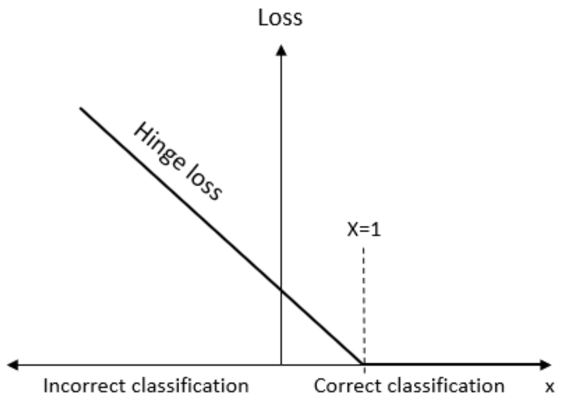


Figure 3.11: Hinge loss. The x-axis is the distance from the boundary and the y-axis is the loss. Correctly classified points near the boundary are penalised. Wrongly classified points are classified more the further they are from the boundary. The dotted line is drawn at x=1.

3.7.2 L2 Regularization

To avoid overfitting, L2 regularization is used. It means that a term consisting of the sum of the squared feature weights, scaled by a scalar λ is added to the loss function:

$$L2 = \lambda \sum_{i=1}^{N} w_i^2 \tag{3.26}$$

When minimizing the loss function, the term forces the weights to be small, i.e., the model complexity is also minimized. A small lambda will make the model prone to overfitting, and a big lambda will lead to an increased risk of underfitting. Hence, a small lambda will choose a narrower margin than a big lambda (Cortes, Mohri, et al. 2012).

By introducing weights to the hinge loss and adding the regularizer, the loss function becomes

$$f(y,w) = \sum_{i=1}^{N} max(0, t_i w^T y_i) + \lambda w^2 \tag{3.27}$$

3.7.3 One versus All

Because the version of hinge loss defined above only works for binary classification, the one versus all strategy is used to expand to multiclass classification. This means that N classifiers need to be trained when there are N classes, i.e., one classifier for each class. Each classifier is only able to distinguish between one class and all other classes. This means that the input to the classification is passed to all of the classifiers. Then, the classifier with the highest probability score determines which class that is assigned to the input.

Chapter 4

Methodology

In this chapter, the methodology for the project is described. Section 4.1 explains the materials used for testing the models, and Section 4.2 describes workflows of traditional segmentation, which is used to create ground truth training images. Section 4.3 presents the process of manually annotating ground truth training images and the steps required to make them ready for use in a deep learning approach, while Section 4.4 describes the process of making synthetic training data. Section 4.5 explains the implementation of the Support Vector Machine, and Section 4.6 presents the implementation and training of the deep learning models. Section 4.7 presents the evaluation metrics used to compare the models and Section 4.8 lists the software and hardware used in this project.

All of the scripts marked in *italics* are hyperlinks and points to the Github repository that accompanies this thesis. If a printed copy of the thesis is used, the scripts may be found by navigating to:
`https://github.com/aavikdal/FluidFlowSegmentation`.

4.1 Materials

The materials used in this study are micro-CT images of two glass vials containing glass beads, oil and water. One of the vials contains oil-wet glass beads, and the other contains intermediate-wet glass beads. The vial is filled with water and oil and is scanned in a micro-CT scanner.

4.1.1 Glass Beads

The glass beads are produced by Sigmund Lindner GmnH [1] and are polished high precision soda lime glass beads. They have a radius of 1.0 mm +/- 0.02 mm. The beads are packed in a glass vial with 10mm inner diameter, 12mm outer diameter and height of 100mm, which is called a DURAN Culture Tube, with DIN thread

[1]`https://www.sigmund-lindner.com/en/`

and screw cap from PBT. The space between the glass beads is filled with water and oil.

The wetting of the glass beads is altered by being soaked in a solution of Hydrocarbon Soluble Siliconizing Fluid. The solution is diluted by heptane to 1 volume percent for an oil-wet result and 0.05 volume percent for intermediate-wet. The fluid is manufactured by Thermo Scientific TM [2]. The beads are soaked in the solution for 5 minutes before they are washed with heptane and methanol. Then they are dried in the oven at $80°C$ for 60 minutes.

When the solution is applied to glass, the unhydrolyzed chlorines in the solution react with surface silanols to form a hydrophobic and tightly bonded film over the entire surface.

The glass beads are wetted by brine in a glass dish and placed into a vial filled with brine. Then, a syringe filled with oil, with a needle as long as the vial's length, is inserted till the bottom of the vial, and oil is injected slowly. The oil will move up and push the brine out of the vial. Once some of the oil reaches the top, the injection is stopped, and the needle is pulled out. The vial is then sealed and put upside down, and is left for a couple of hours before it is scanned.

4.1.2 Micro-CT Scanning

Petricore AS scanned the pack of beads in the mCT lab [3] in a HeliScan micro-CT scanner [4] from ThermoFisher. It scans in a helical manner, making it able to scan the entire object in one scan. This makes the images less prone to artefacts than traditional multi-scan with stitching. The scanner has autofocus, drift correction and delivers a high signal to noise ratio. It has a spatial resolution of 800 nm and is able to scan samples with diameters up to 240 mm. The process of scanning the bead pack took approximately 10 hours. After scanning, a reconstruction software developed by ThermoFischer reconstructed the image with an iterative approach.

The centre 512x512 pixels of 50 slices of the image are cropped in order to remove the cross-section of the glass vial that the glass beads are packed in. An example of such an image slice is shown in the top-left image in Figure 4.2.

4.2 Traditional segmentation

This section describes the process of using traditional segmentation approaches for creating ground truth training images from the micro-CT images introduced in the previous section.

Multiple workflows are tested, and to simplify the process of selecting the best method, they are first tested on images with two classes, water and glass beads, and then the most promising candidates are extended to three classes. The

[2]https://www.thermofisher.com

[3]http://www.petricore.com/en/laboratories/trondheim

[4]https://www.thermofisher.com/no/en/home/electron-microscopy/products/microct/heliscan-microct.html

details for watershed segmentation and Otsu thresholding are explained below. Other workflows tested are K-means clustering and Gaussian mixture models, but the details of the implementation of these are not shown. However, results from these methods are seen in Figure 4.3.

Watershed Segmentation
Watershed segmentation is the first method used to create ground truths. The watershed segmentation starts with an Otsu thresholding operation.

As explained in Section 2.3.3 Otsu thresholding is an easy and straightforward way to segment images. It is not resilient to noise, so the images need to be denoised with non-local means (2.3.2) before the Otsu threshold is calculated and the image segmented. Then, a morphological opening operation (2.3.4) is performed to remove narrow connections and small white holes.

The watershed segmentation (2.3.6) needs to have markers defined to instantiate the segmentation. In practice, this means to define pixels that are certain to correspond to a class for all classes. First, the pixels that are sure to be the background (water) are defined. It is done by using several iterations of morphological dilation, which expands the glass beads, and effectively moves the boundary between the glass beads and the water into the water. This removes the boundary pixels from the background and is seen in the left image in Figure 4.1. This ensures that what is left in the background class is water only. The distance transform is used to identify the pixels that are certain to be in the foreground (glass beads). It changes the pixel values of the foreground (glass beads) to correspond to the distance to the closest water-bead boundary for each pixel. This means that the pixels with the highest values are the pixels furthest from the boundaries. By selecting the top 10 percent of the pixels, they are certain to be in the foreground. This is shown in the middle image in Figure 4.1.

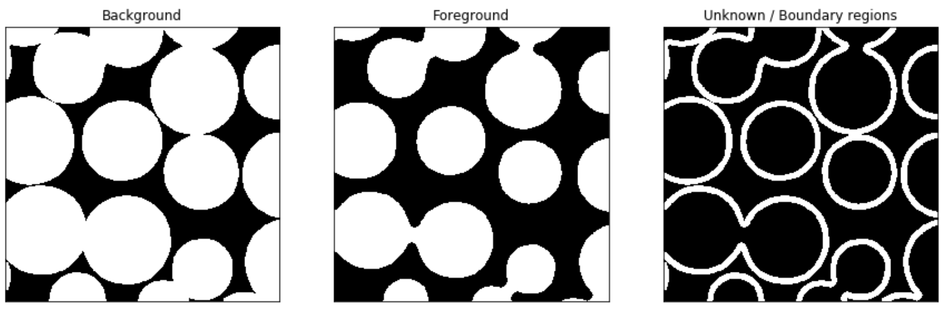


Figure 4.1: The pixel regions used as markers in the watershed segmentation. From left to right: The certain background pixels in black, the certain foreground pixels in white, and the boundary region in white.

By subtracting the foreground from the background, the unknown pixels are identified, i.e., the boundary pixels. This region is visualized in the right picture in Figure 4.1. Now we know which classes are located on either side of the boundaries. The watershed segmentation is then applied to the unknown pixels. The result from the watershed segmentation is seen in Figure 4.2. We see that the edges have sawtooth shapes, which we know the glass beads do not have. In the same figure, we can see that the results from the Otsu thresholding are better, and therefore the Otsu segmentation is expanded to three classes, rather than using the watershed segmentation.

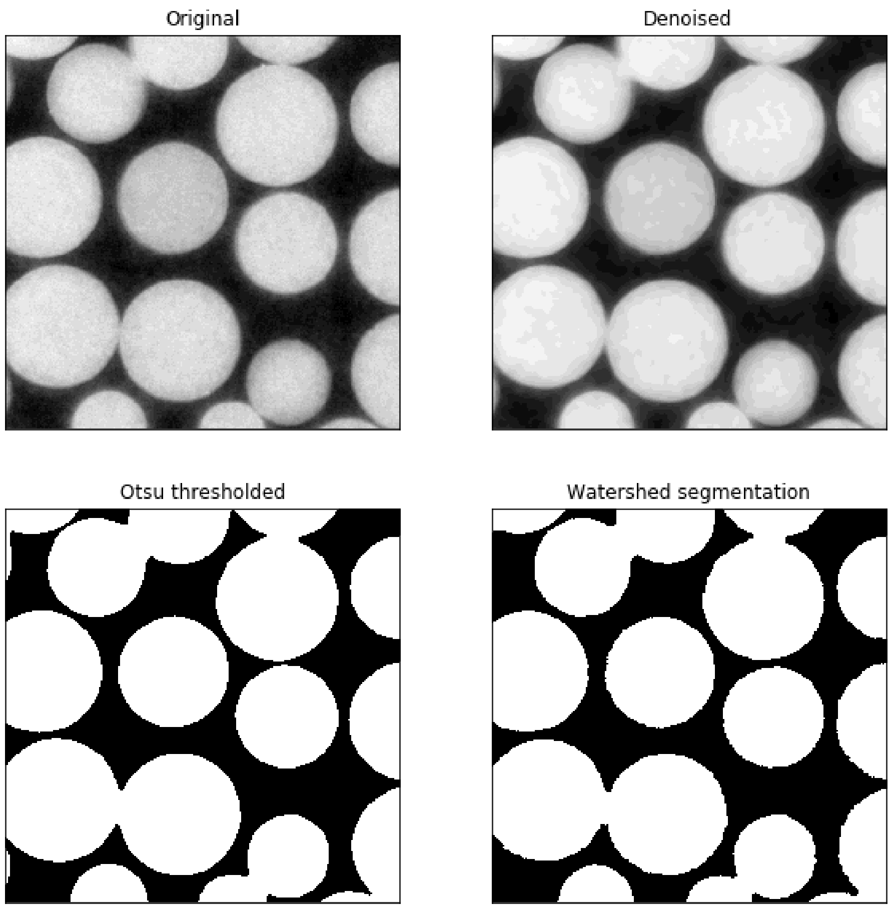


Figure 4.2: Result from Otsu thresholding and watershed segmentation

Otsu multiclass thresholding

Because of the promising results from the Otsu thresholding, it is expanded to be able to segment all three classes. The results are seen in Figure 4.4. The shapes and edges of the glass beads are good, but the segmentation is struggling with the edges between oil and water, and predicts a thin sheet of glass in between the phases, which obviously is wrong. It is understandable that it does this since the brightness value of glass is in between the values for water and oil. Therefore, Otsu thresholding is not used to create ground truths. However, because it is a simple

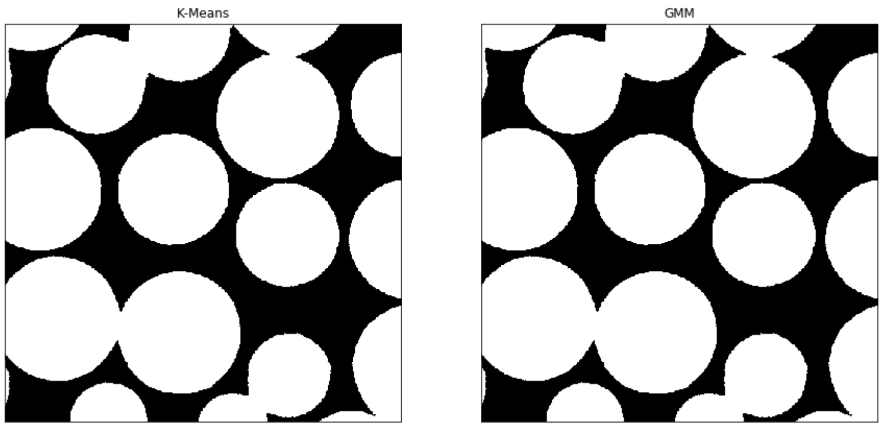


Figure 4.3: K-means clustering and Gaussian mixture model segmentation

and commonly used algorithm, its results are compared to the results obtained from the deep learning approaches. Its implementation is found in *otsu.py*. The input images are denoised with non-local means before two thresholds are calculated. If the distance between the two thresholds is below a predefined margin, the image is considered to consist of two classes, and a regular Otsu thresholding is performed. If the distance is above the margin, the two thresholds are used to segment the image, which gives a segmentation with three classes. The margin is set to be 5 for the results presented later.

The watershed segmentation will not suffer from falsely predicting solids between the oil and water. However, it will still suffer from sawtooth edges and is not suitable for being used as ground truth images. The Gaussian mixture models and k-means clustering give good results for two classes, as is seen in Figure 4.3. However, they also struggle with the boundary between oil and water for three classes.

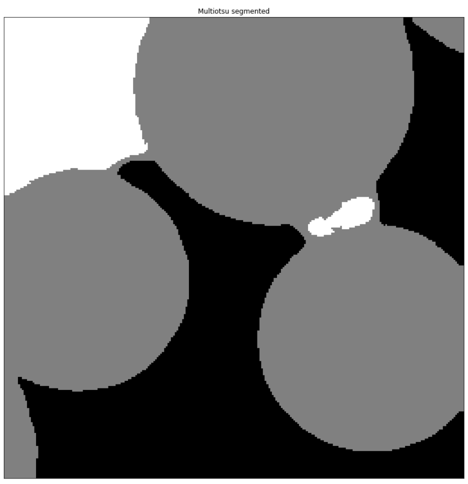


Figure 4.4: Otsu multiclass thresholding

4.3 Manual annotation

Because the traditional methods are not able to give good enough segmentations to be used as ground truth images in a machine learning approach, the images are instead annotated manually. This is done in a web application available on `apeer.com`, which is a free platform funded by ZEISS. They host an annotator application that allows for pixel-level annotation, making it possible to segment the images accurately. With this tool, better results are achieved than with the efforts above. An example of an original image and the corresponding annotated mask is shown in Figure 4.5. 50 512x512 images are annotated, and each of them is split into nine 256x256 images. This means that the second half in one image corresponds to the first half in the subsequent image, as shown in Figure 4.6. This gives redundancy in the pixel values, but not in the semantic meaning. For example, pixels in the rightmost, top half-circle in Figure 4.6 is the same as the pixels in the top, middle circle in the second image, but the semantic meaning is different. In the first image, the pixels are part of a half-circle, whereas they are part of a full circle in the second image. It is, therefore, a useful data augmentation to split images this way.

4.3.1 Data Augmentation

Next, data augmentation is used to increase the size of the dataset and add diversity to it. The different augmentation techniques used are noise, rotation, vertical flip, horizontal flip, grid distortion, blur and downscaling, which makes it a total of $50 \cdot 9 \cdot (1 + 7) = 3600$ images. The implementation of the data augmentation is called *ImageAugmentation.py* and is found in the Github repository. The augmentations done for one image is seen in Figure 4.7.

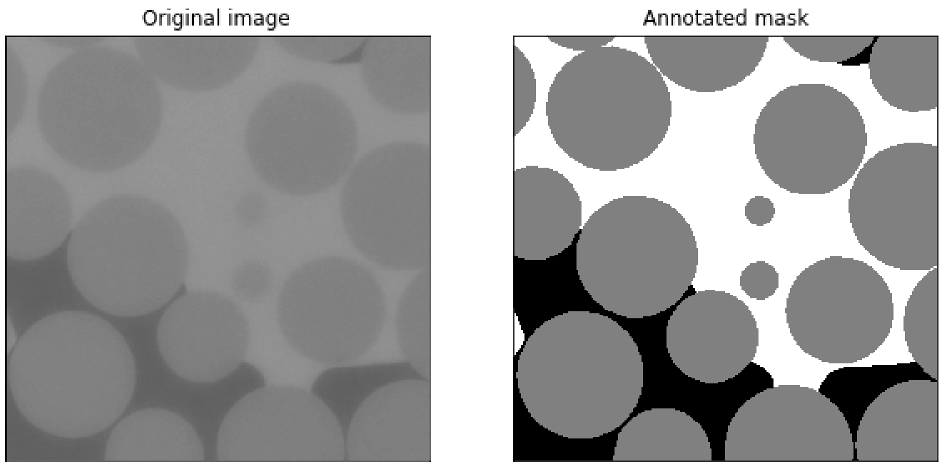


Figure 4.5: Original image and corresponding annotated mask

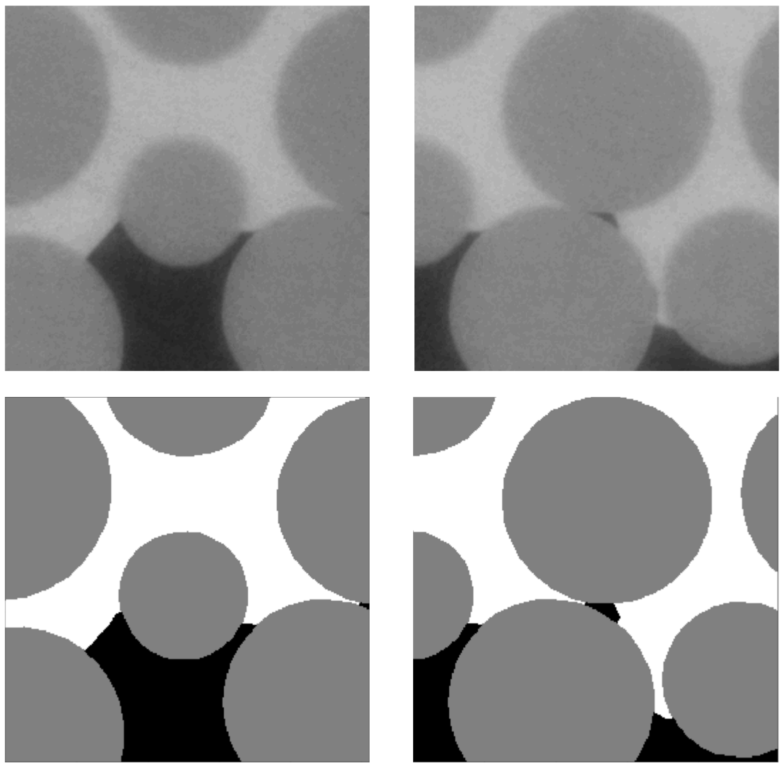


Figure 4.6: Two subsequent images and their corresponding masks. The right half of the first image is the left half of the second image.

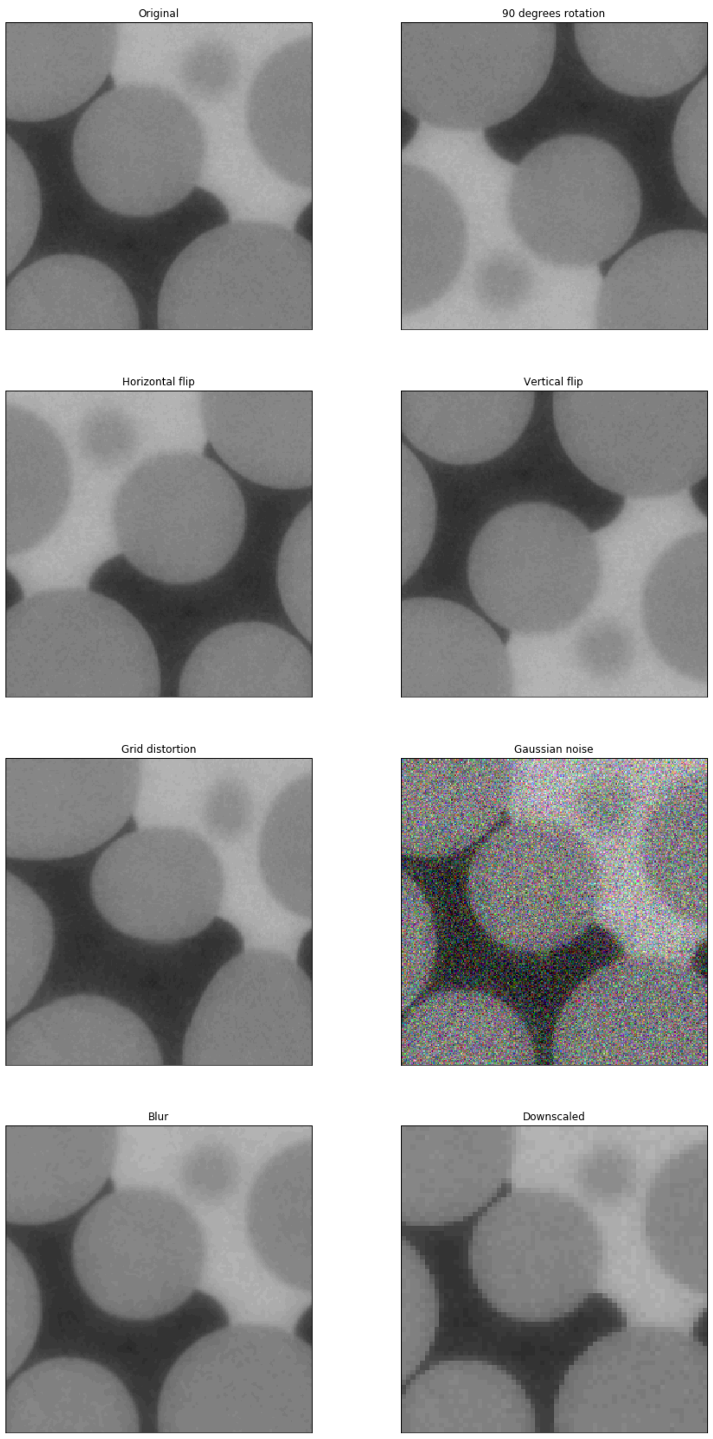


Figure 4.7: Data augmentation transformations applied to one image

4.4 Synthetic Data Creation

As an effort to eliminate user bias, synthetic training data is generated. The data creation consists of sphere pack generation (4.4.1), morphological invasion (4.4.3) and preprocessing (4.4.5) Additionally, methods for expanding the variety of porosity (4.4.2) and contact angles (4.4.4) are included.

4.4.1 Sphere Pack Generation

The packing generation program is developed by Baranau et al. (2014) and available on Github, and is used to generate a monodisperse packing with 1000 particles.

First, a force-biased algorithm, as explained in Section 3.1.1 is used to generate a dense pack from an initial Poisson sphere pack. Then this packing is the input to the Lubachevsky-Stillinger algorithm, explained in Section 3.1.2. Both algorithms need to be used because the Lubachevsky-Stillinger algorithm will only work correctly when the initial packing has a relatively large density (0.4-0.6), otherwise, collisions with far particles will be missed because they will not be included as possible neighbours. The following lines are used to generate the packing.

```
PackingGeneration.exe -fba | tee log_fba.txt
PackingGeneration.exe -ls | tee log_ls.txt
```

After execution of the algorithms, the diameters of the particles in the final sphere pack is rescaled with a MATLAB script that is included in the Packing Generation Github repository. The scripts for scaling the diameters and saving the particle coordinates and diameters to CSV files is found in the Github repository, where they are called *ReadPackingScript.m* and *readFinishedPacking.m*, respectively.

With the script *loadInitialPackingData.py*, the sphere pack is transformed from spatial coordinates of the centers of the spheres to a 3-dimensional matrix. The optimal resolution for the preceding use is hard to determine because it will always be a compromise between field of view and resolution. A higher resolution leads to more details and precision in the image, but the field of view will be reduced due to the memory constraints of the hardware. Having a sufficient field of view is essential because most deep learning models are trained to detect features, and if the field of view is too small, some features will be too big to be represented in the images and will hence not be extracted by the models. As the models are tested on authentic micro-CT images, the resolution of the synthetic images is set to approximately the same resolution, where the diameter of the sphere is 160 voxels.

The matrix consists of integers of values 0 and 255, where 255 represents solids, and 0 represents pore space. The matrix is cropped to a dimension of 1024x1024x1024, which corresponds to 8 GB. This matrix is again cropped into eight 512x512x512 matrices. Figure 4.8 shows a slice of the 1024 grid before and after the packing procedure. All of the spheres are identical, and the slices are cross-sections of 3D grids.

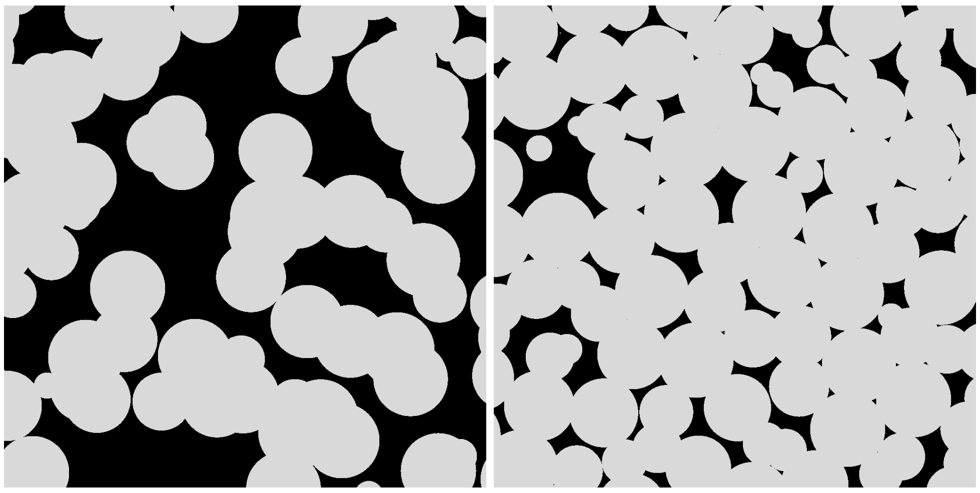

(a) Slice of 1024 grid before packing **(b)** Slice of 1024 grid after packing

Figure 4.8: Initial sphere packing and jammed sphere packing

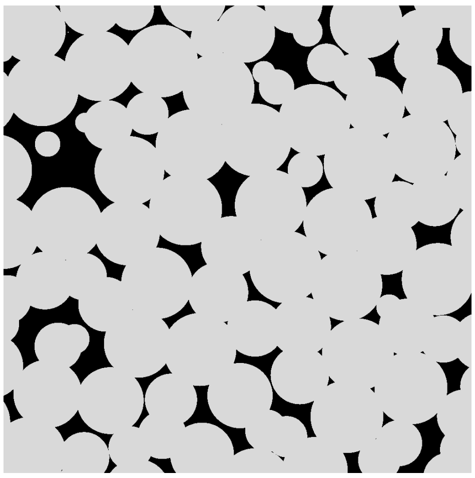
4.4.2 Increase Porosity

Some initial results on the sphere pack described above shows that the created models struggle when the images contain large pores. This is because the sphere packing in the training dataset is more densely packed than the authentic micro-CT images that the models are trained on. It is, therefore, evident that the dataset needs more variety in porosity. This is solved by adding the initial sphere pack, that was created before the packing script was executed, to the training dataset.

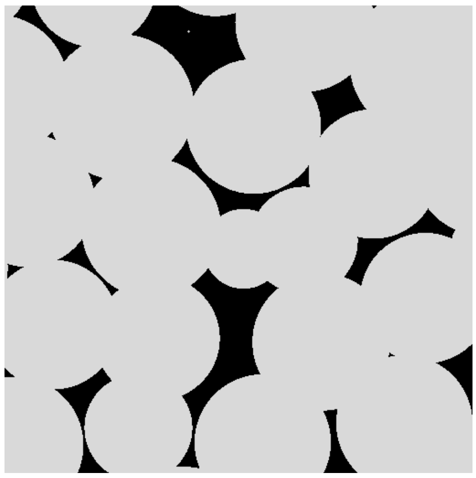
These two porosities represent the extreme cases of porosity, and the models should therefore be able to provide decent segmentations for all of the porosities in this range. However, ideally, more sphere packs with different porosities should be added.

4.4.3 Morphological Invasion

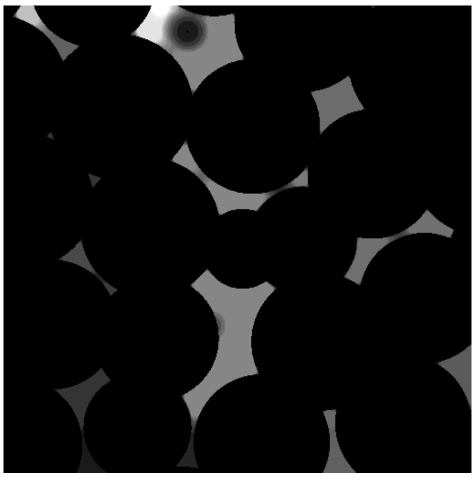
A script that performs morphological invasion is used to invade each 512^3 matrix with oil. The script is described in Section 3.2. It was developed in the paper Berg et al. (2020) and is found in the Github repository, where it is called *morphologicalInvasion.py*. The resulting grid from the script consists of a single brightness value for solids and a continuous range of brightness values that represent the fluid distribution in the pore space of the grid. By using different cut-off values to binarize the continuous fluid distribution, different oil-water distributions are obtained. Values below the cut-off are set to water, and values above the cut-off



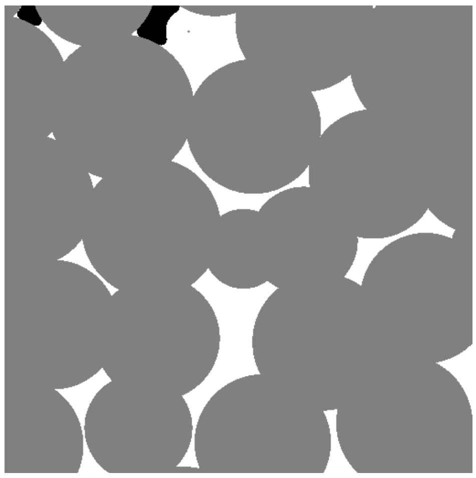
(a) 1024 grid.Binary



(b) 512 grid. Binary.



(c) 512 grid after invasion. Continuous.



(d) 512 grid after cutoff. Ternary.

Figure 4.9: Morphological invasion of binary grid

are set to oil. Thus, the images are converted to ternary images, i.e., a single brightness value for oil, water, and solids.

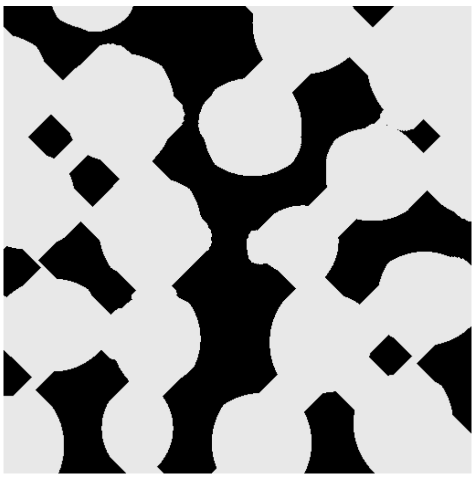
The procedure is illustrated in Figure 4.9. Figure 4.9a shows a slice of the 1024^3 grid and Figure 4.9b shows a slice of one cropped 512^3 grid. Figure 4.9c shows the result from the morphological invasion. The black disks are solid particles, and the pore space has continuous brightness values. By binarizing the pore space with different cut-off values, different oil-water distributions are obtained. Figure 4.9d shows a slice of the resulting ternary grid after applying a cut-off value of 6.

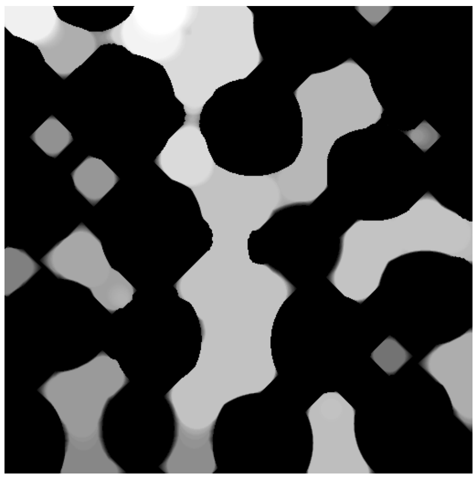
4.4.4 Change contact angle

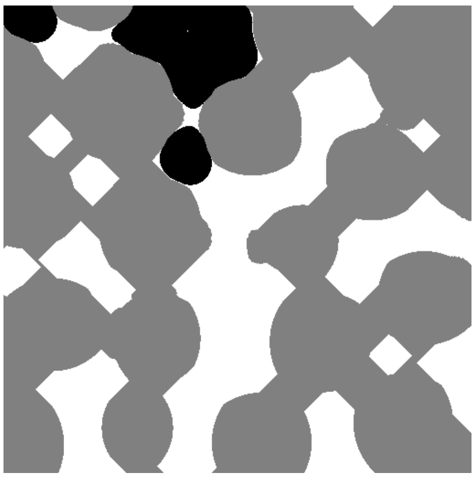
The initial results shows that the models struggle to segment the contact angles right when being used on authentic micro-CT images. This is expected because the images generated straight from the morphological invasion contain water-wet solids, and the images that the models are tested on are oil-wet or intermediate-wet solids.

It is, therefore, necessary to add sphere packs with other contact angles. This is done by modifying one binary 512^3 grid, i.e. a grid with just solids and pore space, with 20 iterations of morphological erosion, which is explained in Section 2.3.4. Because the solids have value of 255 and the pore space 0, the erosion causes the spheres to shrink in size. Morphological invasion is performed on the resulting eroded grid before the spheres are resized to their original size. Because of the increased pore space during the invasion, the oil will flow beyond the original boundaries between the spheres and the pore space, and when the spheres are resized to their original size, the contact angle will be changed. Figure 4.10a shows a slice of the eroded grid and Figure 4.10b shows the eroded slice after the morphological invasion is performed. Figure 4.10c shows the ternary ground truth image, which is created by binarizing the pore space by using a cut-off value, and Figure 4.10d shows the slice when the spheres are resized to their original size.

By comparing Figure 4.10 and Figure 4.12, it is easy to see that the contact angle is modified. In Figure 4.12 the solids are perfect water-wet, and in Figure 4.10a, the contact angle may be characterized as intermediate-wet.


(a) Eroded grid. Binary.


(b) Invaded, eroded grid. Continuous.


(c) Eroded, invaded grid after cutoff. Ternary.

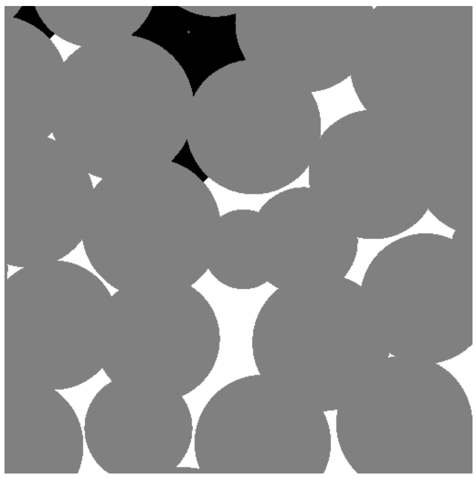

(d) Expanded spheres. Ternary.

Figure 4.10: Contact angle modification by erosion.

4.4.5 Preprocess Into Synthetic Micro-CT Images

The images from the morphological invasion are ternary images and do not resemble authentic micro-CT images. The process of imitating micro-CT images is done by first changing the brightness values for oil, solids, and water to 51, 61 and 67, respectively, which are the average values for the three classes in the authentic micro-CT images used in the testing. Then noise and blur are applied to the images. Noise is applied with different kernel sizes, and the blur is Gaussian. This processing is quite simple, and one expects more artefacts in an actual micro-CT image, but these artefacts will vary from scanner to scanner and operator to operator. The aim is to use data augmentation in the training process to add more artefacts and variance to the dataset in order to make the images both general and specific enough to be used on all kinds of micro-CT images. An example of an authentic micro-CT image and one created synthetically by the method explained is seen in Figure 4.11. The implementation of the preprocessing is called *preProcessImages.py* .

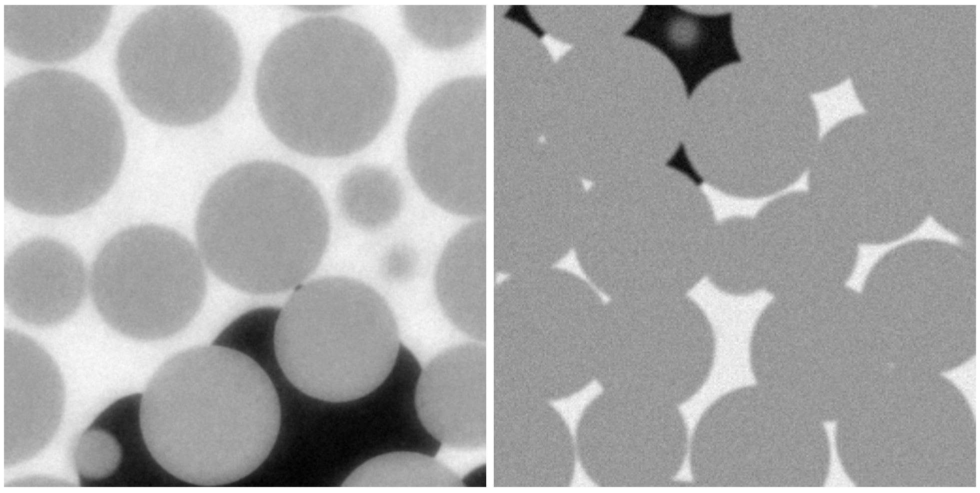

(a) Authentic micro-CT image **(b)** Preprocessed synthetic image

Figure 4.11: Comparison of authentic micro-CT image and preprocessed synthetic image

Both the ternary images from the morphological invasion, which is used as ground truth images, and the preprocessed images, are cropped into 256x256x16 images for the 3D models and 256x256x1 for the 2D models.

Ideally, it would be beneficial to use a larger image depth than 16 pixels. This is, however, not feasible due to hardware constraints. The width and height are chosen to be 256 pixels in order to make the comparison with the 2D models straight-forward.

The reason for experimenting with 3D patches is that the 2D models struggle to segment sphere caps near the end of the solid particles. A hypothesis is that including the adjacent slices in the depth dimension should improve this issue.

There will, of course, exist 3D patches where the ends of the solids will be in either the foremost or rearmost slice in the patch, and the adjacent pixels that would help predict the difficult pixels will be in the neighbouring patch. They will, however, not occur often. An alternative that would remove this problem may be to consider all of the slices in the patch but only segment the middle slice in each segmentation.

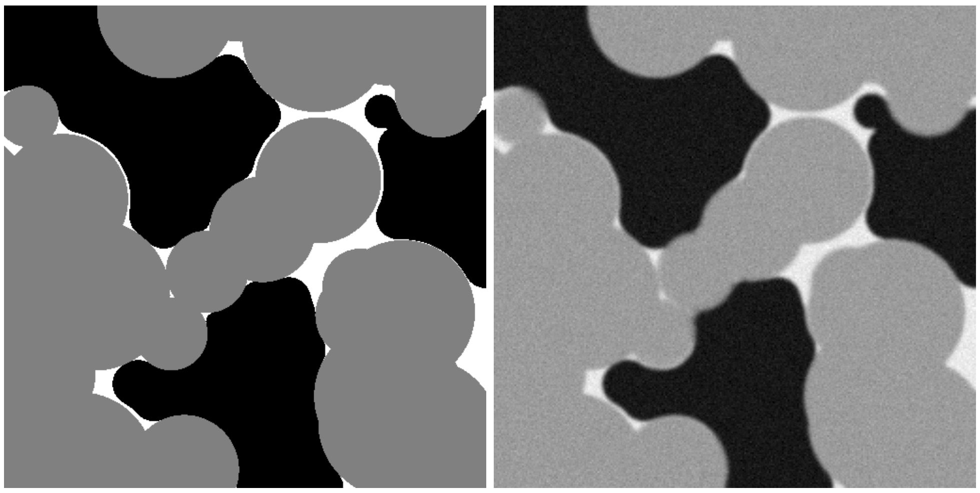

(a) Mask **(b)** Image

Figure 4.12: Mask and image of invaded initial sphere packing.

4.5 Conventional Machine Learning

A conventional machine learning approach is implemented and compared against the deep learning approaches.

4.5.1 Feature Extraction

For the convolutional neural networks, the features of the images are extracted automatically in the convolutional layers. This is not the case for traditional machine learning algorithms, so the features need to be extracted manually. This is done by applying a range of filters to the images and recording the responses in a table, where each row contains the filter responses for one pixel. Each row in this table is an input entry in the model training. There are applied approximately 40 filters to the images, including multiple Gabor filters, Gaussian filters, median filters and multiple edge filters. The implementation is called *feature_extraction.py*.

4.5.2 Support Vector Machine

It was originally planned to implement the Random Forest algorithm, but because of its inability to do iterative learning, which exhausts the computer's memory, the Support Vector Machine, as explained in Section 3.7, is used instead. The algorithm is called *svm.py*. The dataset is imbalanced because there are far more pixels corresponding to glass beads than to oil. The default configuration gives poor results, and class weights need to be applied to the model. They adjust the importance each pixel of each class has on the training process. The theoretical correct class weights are calculated from the total number of pixels corresponding to each class and applied to the model. However, better results are obtained by setting the weights with trial and error. The SVM is trained on the manually annotated images.

4.6 Deep Learning

In this section, the implementation and training of the deep learning models are presented. One model is trained on the annotated images, and three models are trained on the synthetic images. Because they were implemented at different times, the methodology differs slightly. The terms used to describe the methodology and implementation may be found in Section 2.6.2 in the Background chapter.

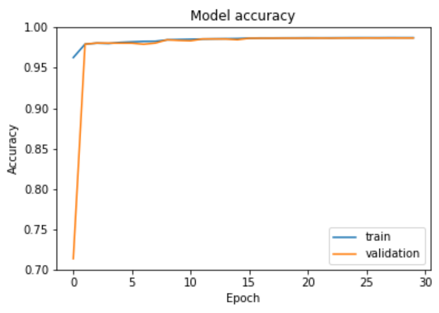
4.6.1 U-Net Trained on Manual Annotations

In this section, the implementation and training of the U-Net trained on the manually annotated images are described. The foundation of U-Net is presented in Section 3.4.

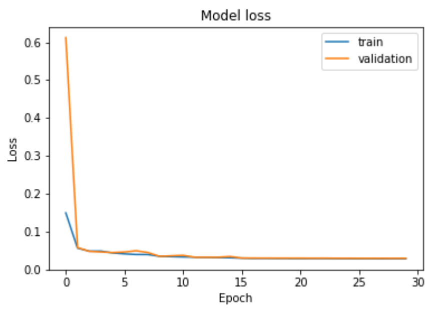
Implementation
U-Net is implemented using Tensorflow Keras, and the implementation is called *UNet.py*. There are some differences to the original architecture proposed by Ronneberger et al. (2015). Here, padding is used to keep the spatial dimensions of the feature maps throughout the layers. After each convolutional layer, a batch normalization is performed to normalize the outputs from the convolution and improve the network's reliability. An input image size of 256x256 is used, and at the bottom of the network, the size of the feature maps is downsampled to 8x8. The number of feature channels ranges from 16 to 128. The dimensions in the model, i.e., input image size, number of feature channels and downsampling rate, are changed in order to adapt the training process to work well on the GPU, listed in Section 4.8.

Training
The model is trained with a batch size of 8 and a train-validation-test split of approximately 0.7 - 0.15 - 0.15. Some additional preprocessing is done before the images are input to U-Net. The implementation of the preprocessing is called *prepare_data_unet.py* and the training process is implemented in *train_unet.py*. The Adam optimizer (3.6) is used to optimize the categorical cross-entropy loss (3.6)

function. The number of epochs is set to 30, but the Early Stopping callback stops the training after 29 epochs since the validation loss has stagnated. The total training time is approximately 20 minutes. The training and validation accuracy and losses for each of the epochs is seen in Figure 4.13. We see that the accuracy for both the training and validation set is increasing slowly until it passes 25 epochs, where it starts to stagnate.



Accuracy plot



Loss plot

Figure 4.13: Accuracy and loss plot

4.6.2 Models for Synthetic Images

In this section, the implementation and training of the models trained on synthetic images are described. Because the models trained on annotated images and the models trained on synthetic images originally were two separate projects, the implementation and training of the models differ.

Three deep learning models are trained on synthetic images, 2D U-Net, 3D U-Net, and a 2D MultiRes U-Net.

2D U-Net

The 2D U-Net is a slightly modified version of the original U-Net proposed in Ronneberger et al. (2015). The implementation is called *UNet2DDropout.py* and is briefly explained below.

Contracting path:

The input layer receives 256x256x1 images, and the network starts with 16 feature channels. It halves the image dimensions and doubles the number of feature channels for each layer until it reaches the bottom layer with an image dimension of 16 and 256 feature channels. Each layer in the contracting path consists of a 3x3 convolution followed by a batch normalization, a ReLU activation, a dropout, another 3x3 convolution, a batch normalization, a ReLU activation function, and a max pool operation with strides equal to two, which reduces the spatial dimension with a factor of two.

Expanding path:

The U-Net is symmetric and therefore doubles the image dimensions and halves the feature channel for each layer as it moves up the expanding path until it reaches the input dimensions of 256x256. The layers differ slightly from the layers in the contracting path. Instead of ending with a max pool, the layers start with a 2x2 transposed convolution (3.3) with strides equal to two. This means that the input to the layer is upsampled and doubles in size. Because it is a convolution, the training will modify how upsamples are done to achieve the best results. This also means that there is one more convolution per layer in the expanding path than in the contracting path. Also, there are skip connections between the layers in the contracting and expanding path. These are made by concatenating the results from the corresponding layer in the contracting path with the results from the layer below in the expanding path.

The final operation of the network is a 1x1 convolution with three feature channels, where each of the three channels correspond to a class. The Softmax activation function is employed on the result of the convolution, and the result is a 256x256x3 matrix. Each element in the matrix is the probability that the pixel in the same x-y position in the input image belongs to the class determined by the z-dimension in the resulting matrix. E.g., the pixel in position [0,0,0] is the probability that pixel [0,0] in the input image belongs to class number 0. A pixel-wise segmentation is made by using *numpy.argmax(prediction, axis=-1)*, which selects the highest probability for each position in the x-y plane and returns the z-index, which is the predicted class.

3D U-Net

The 3D U-Net is similar to the 2D version, but there are some differences. All of the operations in the 2D U-Net are replaced with the corresponding 3D operations. Other than that, the only difference is that the max pooling operation in the z-dimension of the 3D image is only performed in the top two layers due to the shallow depth of the input images, which are 16 pixels deep. The implementation is also available in the Github repository and is called *UNet3DDropout.py*.

2D MultiRes U-Net

The implementation of the 2D MultiRes U-Net is called *multiResUNet2D.py*. The network is introduced by Ibtehaz et al. (2020), and the implementation is retrieved from their Github repository. The convolutional layers in the original U-Net are here replaced with Res Blocks, and the skip connections are replaced with Res Paths, as explained in Section 3.5. Other than that, it follows the same design with contracting and expanding paths that are tied together. To extend the original architecture to multiclass segmentation, the Sigmoid function at the end of the network is replaced with a Softmax function.

As is seen in Table 4.1, the MultiRes U-Net has significantly more trainable parameters than the other two models. This is because the feature channels in the MultiRes U-Net ranges from 32 to 512, and from 16 to 256 in the other U-Nets. Also, the convolutional operations in the Res Paths lead to more trainable

2D U-Net	2D MultiRes U-Net	3D U-Net
1 943 795	7 238 086	2 701 171

Table 4.1: Total trainable parameters

parameters. Despite the difference in the number of trainable parameters, the training times do not differ much and range from 24 to 30 hours.

Model Training

Because both 2D and 3D networks are trained, there are implemented two training pipelines. The implementation of the 2D version is called *train2DNN.py* and the 3D version is called *train3DNN.py*.

All the models are trained from scratch with the Adam optimizer (3.6) which minimizes the categorical cross-entropy (3.6).

The test data is manually separated from the training data before the training ta takes place by reserving one of the eight 512^3 cubes from the jammed sphere pack and selecting approximately 10% of the images from the eroded grid and the initial grid. 20% of the remaining training data is set to be validation data. Online data augmentation is performed on the training data. This means that every time the pipeline fetches a new image and mask, it is a 50% probability that a data transformation is performed on the image and mask. The augmentation scripts for the 2D and 3D augmentation are called *augmentation2D.py* and *augmentation.py* and are further explained in Section 4.6.2.

The 3D model is trained with a batch size of 4, which is the largest batch size that fit in the GPU memory, and the 2D model is trained with a batch size of 32. The number of epochs is set to 100, but the EarlyStopping callback is used with a monitor on the validation loss and 15 epochs patience, which means that the training will stop if there is no reduction in the validation loss in the last 15 epochs. The categorical cross-entropy loss function is minimized with the Adam optimizer, and a callback is used to save checkpoints of the best models. After the training is finished, the training statistics is saved to a json-file.

Data augmentation

Data augmentation is used to increase the size of the dataset and add diversity. The augmentation of the synthetic data is implemented to be used on the fly, which eliminates the need to save the augmented data and keep it in memory. This is different from the annotated images, where the augmentations were performed and saved on disk before the training process. For each image that is fetched by the pipeline, there is a predetermined probability, here 0.5, that data augmentation is performed. The augmentation includes, with uniform probability, flips, rotations, grid distortions, noise, blur, resizing, gamma adjustments, contrast adjustments, brightness adjustments, downscaling and cropping, and are implemented with the albumentations and 3D-volumentations python packages. Not all the augmentation techniques are available for both 2D and 3D. Therefore, the augmentation is slightly different.

Testing
A testing framework is implemented to semi-automate the reporting steps done to compare the results from the models. The pipeline includes a prediction test on the test set, which is a portion of the data that the models have not seen before. From the predictions, the accuracy score, intersection over union, f1 score, precision score and recall are calculated and written to a text file. In addition, the confusion matrix, the loss plot and the accuracy plot from the model training are generated and saved to disk. The testing script is called *testModel.py* .

The models are also tested qualitatively on the test set, allowing weaknesses to be identified and solutions to be proposed.

4.6.3 Test on Authentic Micro-CT images

In the end, all of the models implemented are tested on the authentic micro-CT images. Because there are no objective ground truths to these images, the segmentations are just evaluated qualitatively.

4.7 Evaluation Metrics

Evaluation metrics are used to measure the performance of models and compare them. There exist several methods for doing this, and some common ones are listed below. To understand them, we need to understand true positives, true negatives, false positives and false negatives. True positives are when the model correctly predicts the positive class, and true negatives are when the model correctly predicts the negative class. False positives are when the model incorrectly predicts the positive class, and false negatives are when the model incorrectly predicts the negative class.

Accuracy is the simplest and most common metric. It is the number of correctly classified predictions divided by all predictions (Eq. 4.1).

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Precision is true positives divided by the true positives and false positives, shown in Equation (4.2). A low precision means that there are many false positives.

$$precision = \frac{TP}{TP + FP} \tag{4.2}$$

Recall is true positives divided by true positives and false negatives (Eq. 4.3). A low recall means many false negatives.

$$recall = \frac{TP}{TP + FN} \tag{4.3}$$

F1 score is simply the harmonic mean of precision and recall.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4.4}$$

Intersection over Union (IoU) is the area of overlap divided by the area of union. It is visualized in Figure 4.14. I.e., it measures the number of pixels in common between the ground truth and the prediction divided by the total number of pixels present across both masks. This metric is the most common in semantic segmentation.

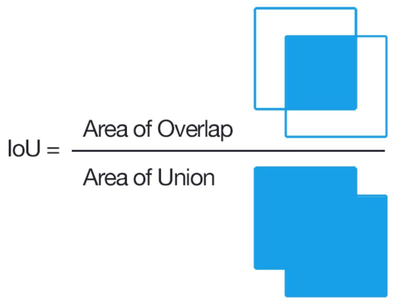


Figure 4.14: Intersection over Union (Rosebrock 2016).

4.8 Hardware and Software

The hardware and software used in this project are:

Hardware
Multicom Kunshan Laptop

- Intel Core i7-6820HK CPU @ 2.70 GHz
- 32 GB RAM
- Nvidia Geforce GTX 1070 Mobile 8 GB 2048 @ 1.44 - 1.65 GPU

Software
Sphere pack generation - PackingGeneration program available at `https://github.com/VasiliBaranov/packing-generation` and created in Baranau et al. (2014).
Programming languages - Python and MATLAB ++
Deep learning implementation - Tensorflow Keras in python

Python IDE - Spyder

Chapter 5

Results and Discussion

In this chapter, the results from the models' performance are presented. Section 5.1 presents the models' quantitative results, Section 5.2 presents the models' qualitative results on the test set and Section 5.3 presents the models' performance on authentic micro-CT images. In order to simplify the discussion, the dimension of the model will only be mentioned if it is 3D, so all models mentioned without dimensions are meant to be 2D.

5.1 Quantitative results on test set

In this section, the models are evaluated based on their performance on the test set. The test set is disjunct from the training data but consists of the same type of data. Approximately 10% of the training data was moved from the training folder to the test folder before any training took place. The test data for the 2D models and 3D models are slightly different due to the different image size.

Metric	**2D U-Net**	**2D MultiRes U-Net**	**3D U-Net**
Accuracy	0.9943	0.9939	0.9944
IoU	0.9887	0.9879	0.9888

Table 5.1: Model evaluation

As seen in Table 5.1, all of the models perform well on the test set, with accuracies above 99.3% and intersection over union above 98.7%. The 3D U-Net has the highest accuracy with 0.9944, whereas the MultiRes U-Net has the lowest accuracy with 0.9939, meaning that the difference between the best and worst is $0.0004 = 0.04\%$. The difference in IoU is $0.9888 - 0.9879 = 0.0009 = 0.09\%$.

Table 5.2 shows that the U-Net struggles with a low recall for oil and water, which are below 97%. A low recall means many false negatives, i.e., that the models predict there to be solid when the true label is oil or water. The confusion matrix confirms this in Figure 5.1, which shows that in approximately 3% of the times the true label is either oil or water, the model predicts there to be

Class	2D U-Net	2D MultiRes U-Net	3D U-Net
Oil	0.9698	0.9649	0.9915
Solids	0.9983	0.9996	0.9967
Water	0.9662	0.9469	0.9645

Table 5.2: Recall

solid. However, when the true label is solid, it predicts solid 100% of the time. It, therefore, seems that the model predicts the solid spheres to be a bit too big. From Table 5.2 it is evident that also the MultiRes U-Net struggles with low recall for oil and water. In the confusion matrix (Figure 5.3), it is shown that the model predicts there to be solid in 5.2% of the cases where the correct label is water. For oil, the number is 3.4%. The 3D U-Net has a high recall for both oil and solids and is only struggling with water, where the recall is 96.45%. The problem is that the model predicts there to be solid when there is water.

A possible reason for the low recall scores is class imbalance. An examination of the training data revealed that it consists of 8.3% oil, 85.7% solid and 6% water. The categorical cross-entropy loss function does not take class imbalance into account. Because there are approximately ten times more solid pixels than oil pixels, it is more critical for the model to predict the solids correctly to attain a good accuracy. A loss function with class weights might help to attain more balanced results.

Class	2D U-Net	2D MultiRes U-Net	3D U-Net
Oil	0.9908	0.9970	0.9907
Solids	0.9954	0.9936	0.9968
Water	0.9831	0.9942	0.9640

Table 5.3: Precision

From Table 5.3, both the U-Net and the MultiRes U-Net have high precision scores for all classes. However, the 3D U-Net has a low precision score for water. A low precision score means false positives, i.e. that the model predicts there to be water when the true label is not water. From the confusion matrix (Figure 5.5), we see that in 0.25% of the cases where the true label is solid, the model predicts water. This might seem like a low value, but keep in mind that there are many more solid pixels than water pixels, and although this is a low percentage of the solids, it will be a more significant fraction of the water. This may indicate that it is harder to correctly segment the boundaries between solids and water than between solids and oil. From the methodology section, we have that the mean brightness values of oil, solids, and water are 51, 61 and 67, respectively. The brightness difference between water and solids is smaller than the difference between solid and oil. This might be why it is harder to correctly segment the boundaries between solids and water than between solids and oil.

Class	2D U-Net	2D MultiRes U-Net	3D U-Net
Oil	0.9802	0.9807	0.9911
Solids	0.9968	0.9966	0.9968
Water	0.9746	0.9699	0.9642

Table 5.4: F1 score

The f1 scores are reported in Table 5.4. Because it is the harmonic mean of precision and recall, it gives a better measure of the incorrectly classified pixels than the Accuracy metric in imbalanced datasets. The harmonic mean penalizes extreme values and therefore prefers a balanced ratio of precision and recall. As expected, all models have high f1 scores for the solids and lower f1 scores for oil and water. The average f1 scores for the three models are 0.9839, 0.9824 and 0.9840 for the U-Net, MultiRes U-Net and the 3D U-Net, respectively. Also, in this metric the 3D U-Net is performing best and the MultiRes U-Net is performing worst.

However, it is important to note that these numbers only describe how well the models are able to segment the synthetic images. A high score on these metrics is not worth anything if the models fail to segment the authentic micro-CT images. Additionally, good results are expected since most pixels should be easy to segment, and only the boundary sections should be challenging. Therefore, we must not take for granted that these models make good segmentations because of their high evaluation metrics. We also need to evaluate the segmented images qualitatively.

5.1.1 2D U-Net

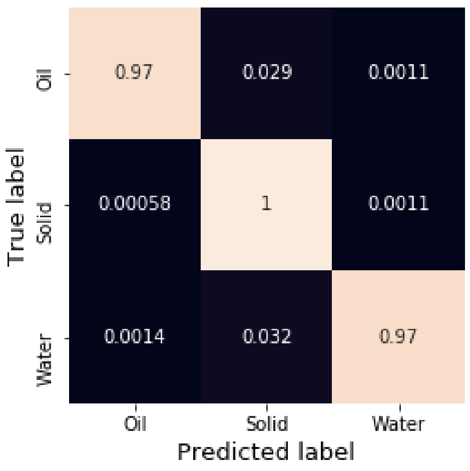
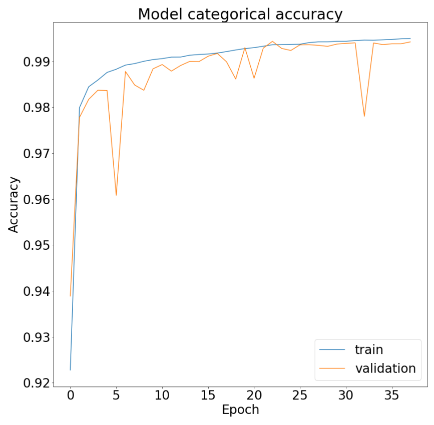
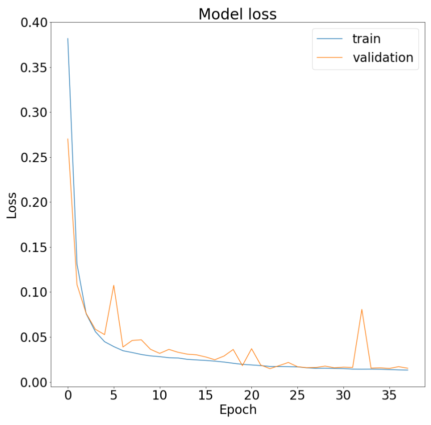


Figure 5.1: 2D U-Net Confusion Matrix



(a) Accuracy Plot



(b) Loss Plot

Figure 5.2: Accuracy and loss plot for 2D U-Net

The U-Net quickly achieves a training accuracy over 99%, which slowly increases epoch after epoch. The validation accuracy follows the same trend as the training accuracy but has a few dips on the way. The loss drops quickly for both the training and validation due to the exponential decay.

The loss has a few spikes that correspond with the dips in the accuracy. These spikes are normal when using versions of mini-batch SGD on datasets with a high variance because the weights of the network are updated from the average of gradients in a small subset of the total data. Therefore, the optimizer performs frequent updates with high variance that cause the objective function to fluctuate. In other words, the current local minima does not generalize as well as the local minima at the end of the previous epoch. These fluctuations have both positive and negative consequences. It enables the optimizer to jump to a new and potential better local minima, but it also complicates the convergence to the global minimum.

Due to that the validation loss does not go lower than it does in epoch 23, the EarlyStopping callback stops the training after 28 epochs since a 15 epochs patience is used. However, the trend of the validation accuracy is positive, and the model might achieve better results after more training.

5.1.2 2D MultiRes U-Net

The MultiRes U-Net also quickly achieves a training accuracy over 99%, but the validation accuracy does not follow as closely to the training accuracy as for the regular U-Net. The validation accuracy is volatile, but we can see that it has a positive trend and that the volatility decreases with time. The fact that the validation accuracy has an increasing trend suggests that it is not overfitting. The model achieves its lowest validation loss in epoch 19, and the training is therefore ter-

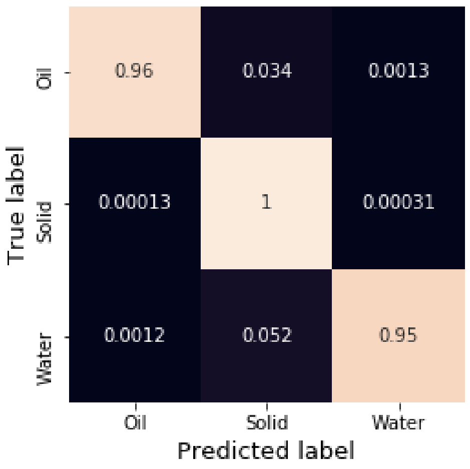
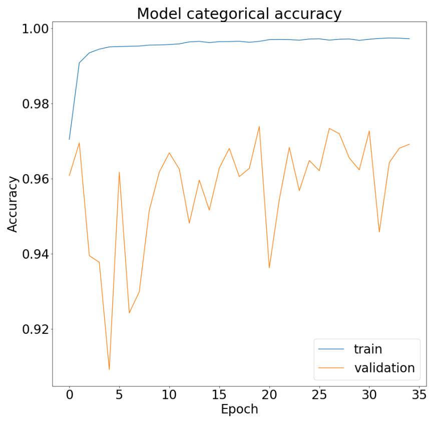
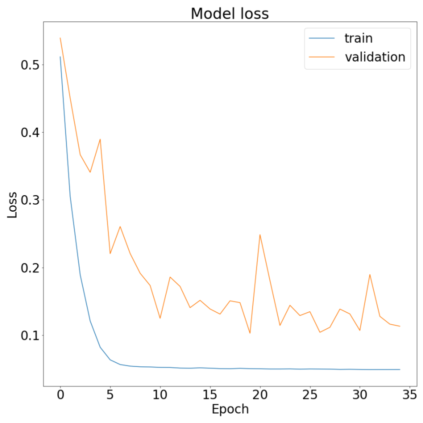


Figure 5.3: 2D MultiRes U-Net confusion matrix



(a) Accuracy Plot



(b) Loss Plot

Figure 5.4: Accuracy and loss plot for 2D MultiRes U-Net

minated after 34 epochs.

5.1.3 3D U-Net

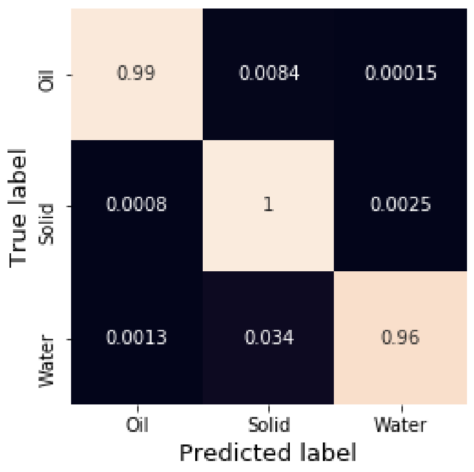
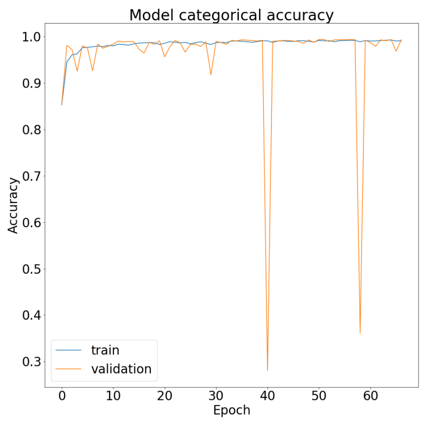
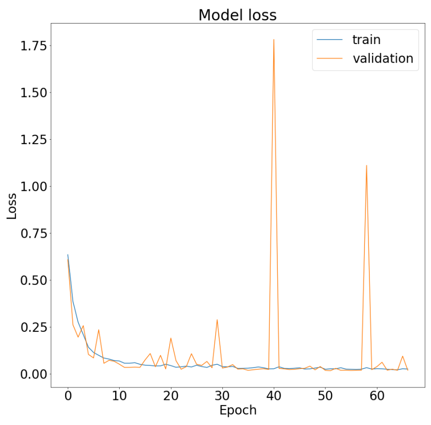


Figure 5.5: 3D U-Net confusion matrix



(a) Accuracy Plot



(b) Loss Plot

Figure 5.6: Accuracy and loss plot for 3D U-Net

The 3D U-Net quickly achieves a high training accuracy. The validation accuracy is volatile, even more than it appears at first, because the y-scale is different from the other training plots because the validation accuracy drops to 30%. After approximately 30 epochs, both the training and validation accuracy are stable, but the training does not end until epoch 67.

The model's accuracy and loss for the U-Net trained on annotated micro-CT images are displayed in Figure 4.13 in the Methodology section. Both the validation accuracy and validation loss are less volatile than for the models trained on synthetic data. It is important to note that for the training of U-Net, only 50 annotated images of size 512^2 form the basis of the data before they were cropped, and the dataset was expanded with data augmentation. For the synthetic data models, 8650 images, also of size 512^2, forms the basis before cropping and data augmentation. Additionally, the data augmentation was set to generate a wider variety in the synthetic images than was the case for the annotated images, and the U-Net implementation used on the annotated data has fewer feature channels than the models trained on synthetic data. The bigger dataset size, the wider variety in data, and an increased number of trainable parameters explain why the synthetic models were harder to train than the U-Net which was trained on annotated images. This is also reflected in the training times for the models. The U-Net trained on annotated images used approximately 20 minutes, while the other models used more than 24 hours. Longer training times is a disadvantage, but the hypothesis is that the models trained on a more extensive dataset with more variety should be able to generalize better. However, the biggest advantage with synthetic images is that there is no operator bias in the segmented ground truths, because they are actual ground truths. The dataset might, however, be biased of the operator that created it, but there should be no problem to add enough variance of all of the relevant parameters, e.g., porosity, contact angles, fluid distribution and grain size, to reduce the bias.

5.2 Performance on Test Images

In this section, the resulting segmented images generated from the different models applied to the test set of synthetic images are visualized. Each figures include the original image, the mask, i.e., ground truth, the segmentation and the difference between the segmentation and the mask.

5.2.1 2D U-Net

Figure 5.7 displays the U-Net's segmentation of a test image. It makes an overall good segmentation, but the result is not perfect. In the small, triangular pore spaces, it smooths the corners a bit, and the small solid circle in the bottom right is not perfectly circular. In the bottom right figure, where the differences are marked in red, some errors are evident. However, these are dispersed along the edges, and there are no spots where the red markings are thick.

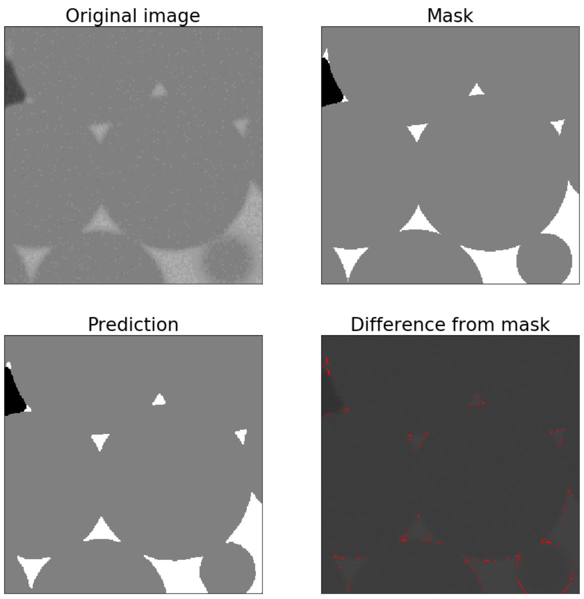


Figure 5.7: 2D U-Net segmentation on image from the test set. The difference between the prediction and mask is visualized in red.

5.2.2 2D MultiRes U-Net

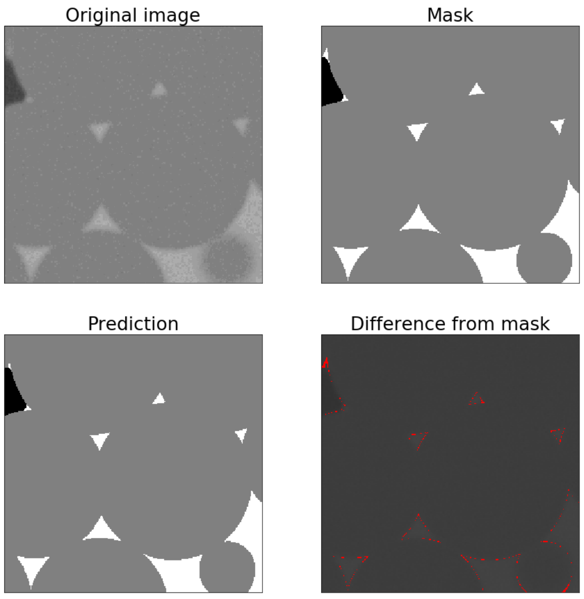


Figure 5.8: 2D MultiRes U-Net segmentation on image from the test set. The difference between the prediction and mask is visualized in red.

Figure 5.8 shows the MultiRes U-Net's segmentation of the same image as the U-Net. The MultiRes U-Net does a better job than the U-Net at making the corners in the tiny pore spaces sharp and the bottom right solid circular. However, from the red markings, we see that it struggles with the small pore space in the upper left corner.

5.2.3 3D U-Net

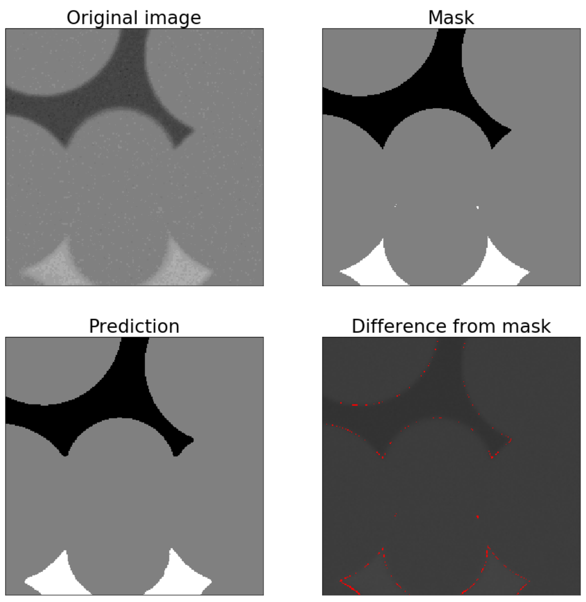


Figure 5.9: 3D U-Net segmentation of image from the test set. The difference between the prediction and mask is visualized in red.

Figure 5.9 shows a segmentation created by the 3D U-Net. The slice displayed in the figure is slice number 8 in the 16-pixel deep segmentation, and it should therefore benefit from depth information in both directions of the slice. It does an overall good job and makes no major errors, but there is still room for improvement. The boundary between some of the classes are fuzzy, and the solid particle in the lower middle is not perfectly circular. This is worrying since it seems that the model has not picked up some key semantics, namely that the boundaries should be sharp and the solid should be perfect circles.

All models make decent segmentations, but none of them is perfect. The two 2D models seem to perform better than the 3D U-Net, which is interesting because the 3D U-Net performed best in the quantitative evaluation of the results.

5.3 Performance on Authentic Micro-CT Images

Section 5.3 shows all of the models' segmentations of both micro-CT images of oil-wet and intermediate-wet glass beads. Both the resulting segmentations of the models implemented in the specialization project and master project are included. Each figure displays the original CT image and the resulting segmented images from Otsu thresholding, a Support Vector Machine, a two-dimensional U-Net trained on annotated images of oil-wet glass beads, a two-dimensional and a three-dimensional U-Net trained on synthetic images and finally a two-dimensional MultiRes U-Net. The contrast of the original CT images is modified for better visibility.

One of the U-Nets is trained on annotated images from the oil-wet micro-CT images and is therefore assumed to generate good segmentations on these images. It is, therefore, interesting to see how the models trained on generic, synthetic data compares to this. Since there are two 2D U-Net models in the results, the discussion is simplified by naming the 2D U-Net trained on annotated data *U-Net I* and the 2D U-Net trained on synthetic data *U-Net II*.

5.3.1 Oil-Wet Glass Beads

Two of the models, the 2D U-Net and the SVM are trained on manually annotated images from the oil-wet glass beads. The other 2D U-Net, the 3D U-Net and the 2D MultiRes U-Net are trained on synthetic images. Figure 5.10 shows the segmentation of micro-CT image of oil-wet glass beads. As expected, the Otsu thresholding is able to segment the rough features of the image but is struggling with the boundaries and predicts there to be a sheet of solid glass between the oil and water. This is because thresholding only considers the brightness values of the pixels. There will be a transition in pixel values from black to white between the oil and water phase, and the pixel values of gray naturally lie between the two and solids will therefore be predicted.

The Support Vector Machine, which uses filters to extract the features of the image, is also able to segment the coarse features of the image. However, due to the small kernel size of the filters, it fails to extract the big scale semantics, and therefore it is some noise along the edges of the classes. Most of these small patches could be removed, with, for example, a morphological closing, but one of the goals for the project is to eliminate the need for manual post-processing of the segmented images. One could implement automatic closing operations, but it might remove small objects, e.g., small oil droplets, which is not desirable in other scenarios.

As expected, the U-Net I generates a more or less flawless segmentation. Nevertheless, one should keep in mind that this model was trained on these exact images. However, a couple of things could be improved: It seems that the contact angles are not perfectly recreated, and there is a small dent in the lower, left glass bead where the intersection of oil, water and solid is.

The U-Net II, which is trained on synthetic data, makes an acceptable segmentation. It is able to capture the main features, but there are a couple of errors. There is a small patch of solid that is being wrongly predicted in the water phase. From the original image, one sees that this area is a bit darker than the rest of the water phase, and that is probably the reason for predicting solid there. It is, however, clear that it has not learned all the relevant semantics of the synthetic images since there does not exist any solids shaped like that in the training set. It also struggles to create the correct contact angles, which is especially visible on the left boundary of the tiny oil droplet.

The 3D U-Net also makes a decent segmentation, but with some obvious errors. The first one being the small oil droplet it wrongly predicts inside the bottom right glass bead. Right next to it, it also makes a small dent in the glass bead, which seems to be because of the brightness values of this area. It also wrongly predict a slight bulge on the glass bead right next to the oil droplet. This is also probably a brightness value issue. Overall, it seems that the model ranks the importance of the brightness values too high and have not adequately learned the semantics of the images.

Visual comparison of the MultiRes U-Net indicates an almost perfect segmentation. The only thing to remark is that it fails to create the correct contact angles and predicts the glass beads to be water-wet. This is probably due to the lack of variety of contact angles in the training dataset. The training set primarily consists of water-wet solids but also contains some intermediate-wet solids. It is therefore understandable that the model struggles with this, since the glass beads in this picture are oil-wet.

In short, the best segmentation in Figure 5.10 is made by U-Net I, but the MultiRes U-Net is not far behind. With more variety of contact angles in the training set, I am convinced that it can make an equal good segmentation.

Figure 5.11 shows the segmentation of an image where we see the contours of the end of a glass bead. In the specialization project, these were shown to be the most challenging images to segment due to the large amount of blur in the area. This blur makes it hard to determine where the boundaries should be, and also, since there is a blur in all three dimensions, if there should be solid there at all. The proposition in the specialization project suggested that a 3D model should alleviate this problem since the information in depth should help segment the ends of the glass beads. The slice displayed in the figure is from the middle of the 3D patch used in the 3D segmentation, and the 3D model should therefore be able to make a proper segmentation.

The Otsu thresholding struggles with the boundaries, and the Support Vector Machine struggles with noise, as in the previous image.

The U-Net I makes an acceptable segmentation but has a couple of flaws. It predicts there to be solid in the problematic area, but the solid is not perfectly circular, as it should be, and it is predicted to be a bit too big. It also has issues segmenting the border of the glass bead on the right side of the image and makes a big dent.

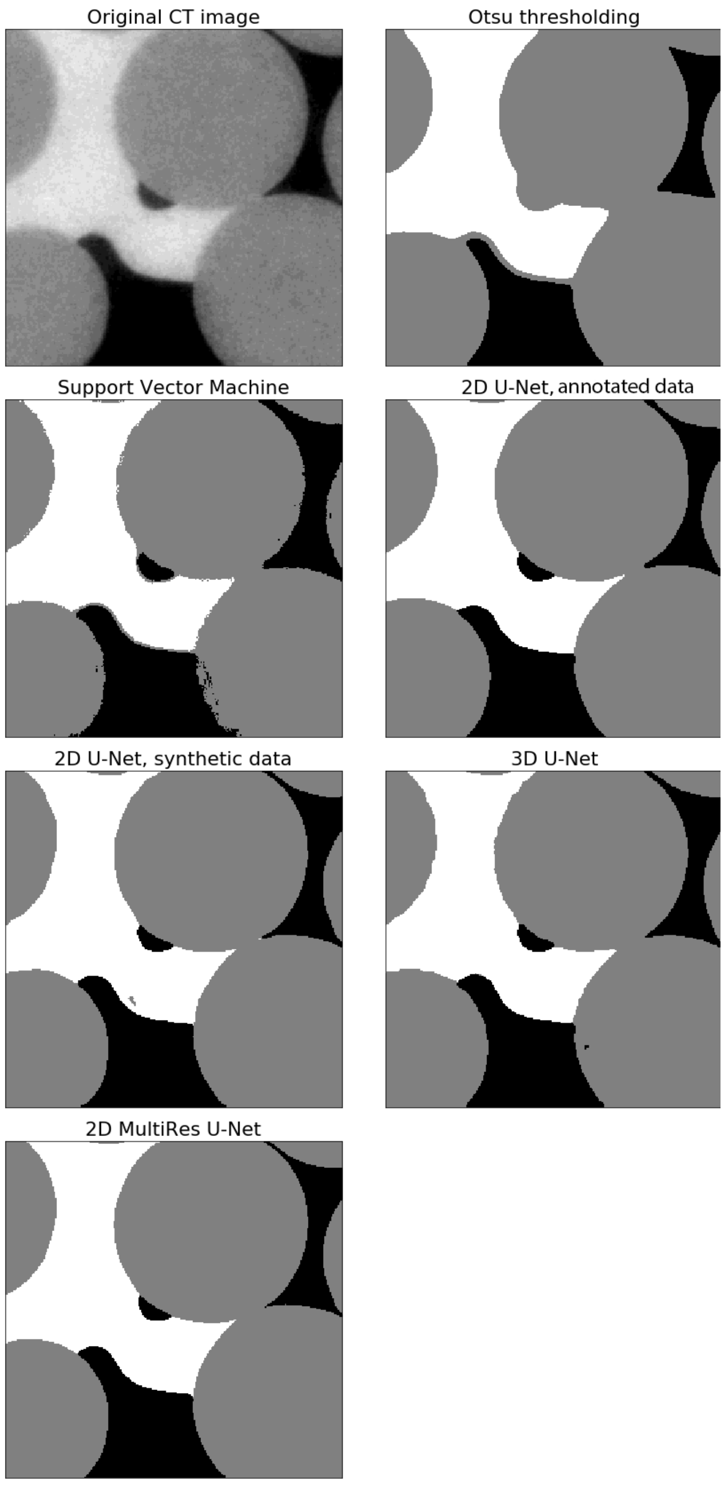


Figure 5.10: Segmentation of oil-wet glass beads. SVM and one of the 2D U-Nets are trained on authentic images from this dataset, while the other 2D U-Net, 2D MultiRes U-Net and the 3D U-Net are trained on synthetic images.

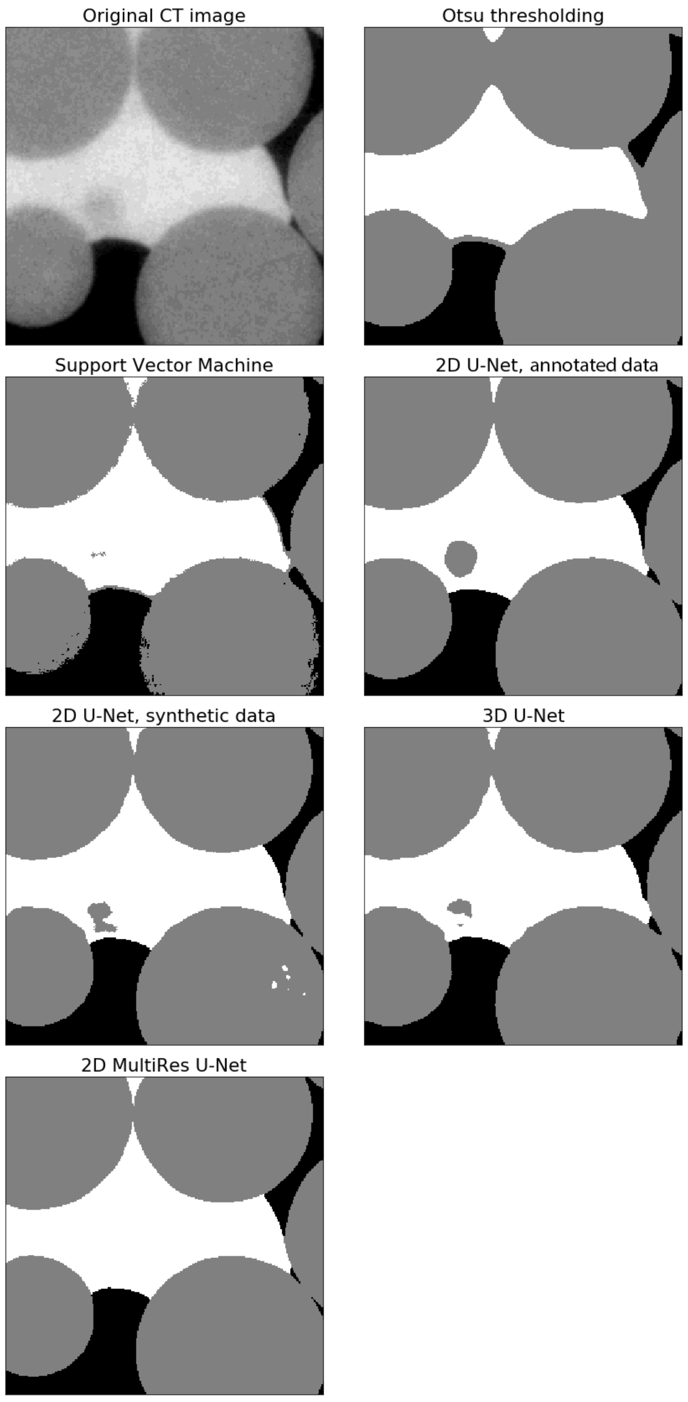


Figure 5.11: Segmentation of oil-wet glass beads.

The U-Net II mispredicts some water droplets inside one of the glass beads in Figure 5.11, which is probably due to a non-uniform brightness distribution inside the glass bead. In the problematic area, it generates an unacceptable segmentation and predicts a shape like an hour-glass. It is yet again evident that the model ranks the local features, such as brightness values and gradients, higher than semantics when generating the segmentation.

The 3D U-Net also struggles with the prediction of the small cross-section and predicts a half circle in the area. This shows that this model also ranks brightness values above the semantics of the image. On top of that, the boundaries are fuzzy, just like in the previous image.

The MultiRes U-Net makes a good segmentation again. It does not predict anything in the trouble area, and this might be true. It is hard to determine if the shade in the image is the glass beads or just the blur surrounding it. The contact angles seem to be acceptable, but it misses a bit of the thin layer of oil on the surface of the rightmost glass bead.

In this figure, the U-Net I and the MultiRes U-Net makes equally good segmentations. It is hard to decide if the small cross-section of the glass bead should be included or not, and they both have some issues with the oil on the right-hand side of the image.

Figure 5.12 shows the segmentation of an image with an imbalanced class distribution. Class imbalance is particularly bad for Otsu thresholding, as is visible in the figure. Due to fewer oil-water boundaries, the SVM seems to do a better job than usual. However, there is still some noise in the segmention and a thin sheet of glass between the oil and water. The U-Net I generates a good segmentation. The only flaw I see is that the curvature of the boundary between oil and water is a bit off. The U-Net II also makes a good segmentation. Also here, the only flaw is the curvature of the oil-water boundary. The 3D U-Net struggles with the boundary between the glass beads and the water. One clearly sees that the glass beads are not perfectly circular. It seems to regard the local brightness information higher than the global semantic information. The MultiRes U-Net makes a good segmentation, but the contact angle is a bit off here as well. The bottom oil-water boundary is segmented water-wet when it should be oil-wet.

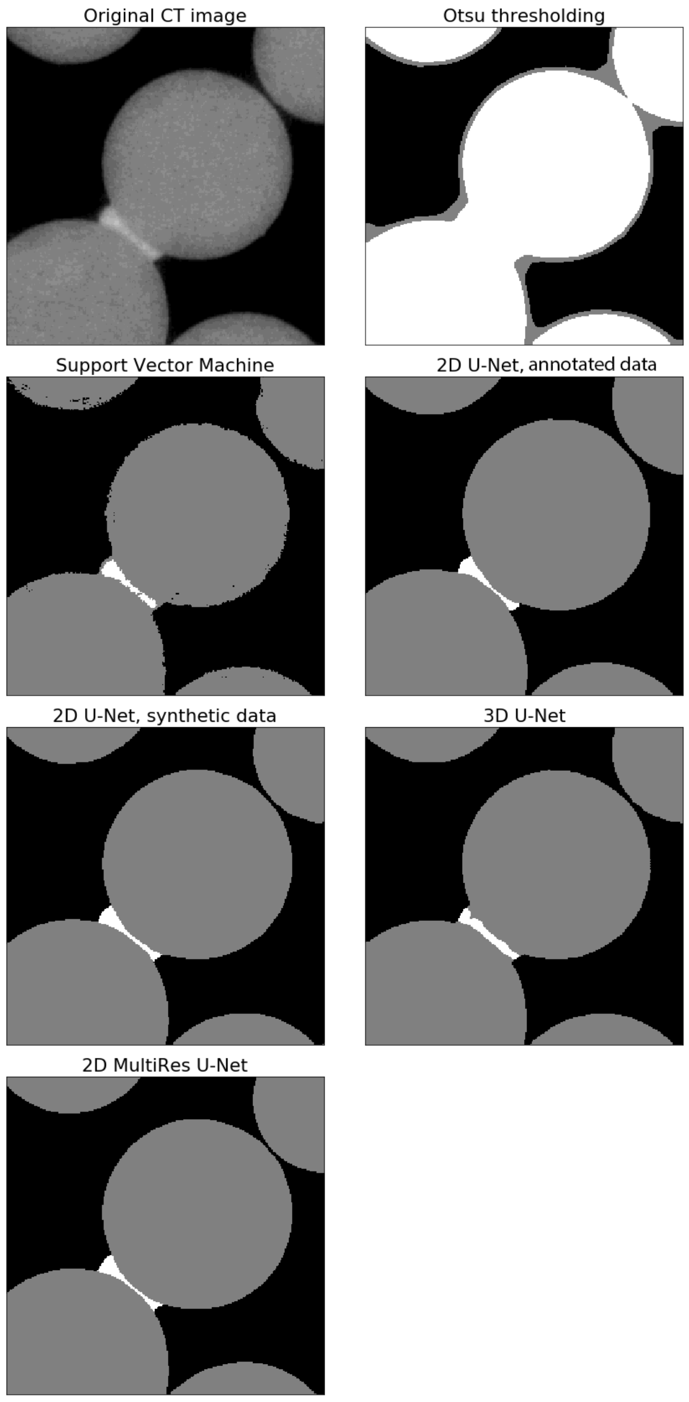


Figure 5.12: Segmentation of oil-wet glass beads.

5.3.2 Intermediate Wet Glass Beads

In this section, the models' performance on intermediate wet glass beads are displayed. This means that the SVM and U-Net I are tested on images that are different from the ones they are trained on. However, the differences are minor, and therefore, it is expected that they generate precise segmentations on these images.

In Figure 5.13 Otsu thresholding makes a decent segmentation but fails to segment the oil in the pore throats between the glass beads. The Support Vector Machine seems to struggle less with noise in this image than in the previous images. However, the boundaries are still a bit fuzzy, and it also struggles with the boundary between oil and water. The U-Net I makes a good segmentation, and it seems that it is able to segment the contact angles correctly. The U-Net II generates a satisfactory segmentation. There is, however, a small spot of water being predicted inside one of the glass beads, and it seems to predict too little oil in the pore throats. The 3D U-Net's segmentation has fuzzy edges and struggles in the pore throats. As usual, the MultiRes U-Net is the best model from the models trained on the synthetic data, but it does seem like it also predicts too little oil in the pore throats and the glass bead in the bottom is not perfectly circular.

In Figure 5.14, the Otsu does a decent job except for the boundaries and the SVM struggles with noise. Both U-Net I and II generates good segmentations but fail to make the small glass bead perfectly circular. The 3D U-Net struggles with very wobbly edges. The MultiRes U-Net makes the best segmentation out of all of the models, but if we are picky, the contact angles are not perfect, and the top, right glass bead has a dent on its right side.

Figure 5.15 contains a micro-CT image with many glass beads and a lot of fluid contacts. Otsu thresholding struggles as usual with the fluid contacts, and the SVM struggles with noisy edges. U-Net I generates a good segmentation, whereas U-Net II fails to make the smallest glass bead perfectly circular, and the contact angles are a bit off. The 3D U-Net creates fuzzy boundaries, but still generates a decent segmentation. The MultiRes U-Net is yet again the best model trained on synthetic data but fails to make the segmentation as precise as U-Net I. It fails to make the smallest glass bead circular, and is way off in segmenting the contact angle on the middle, left glass bead.

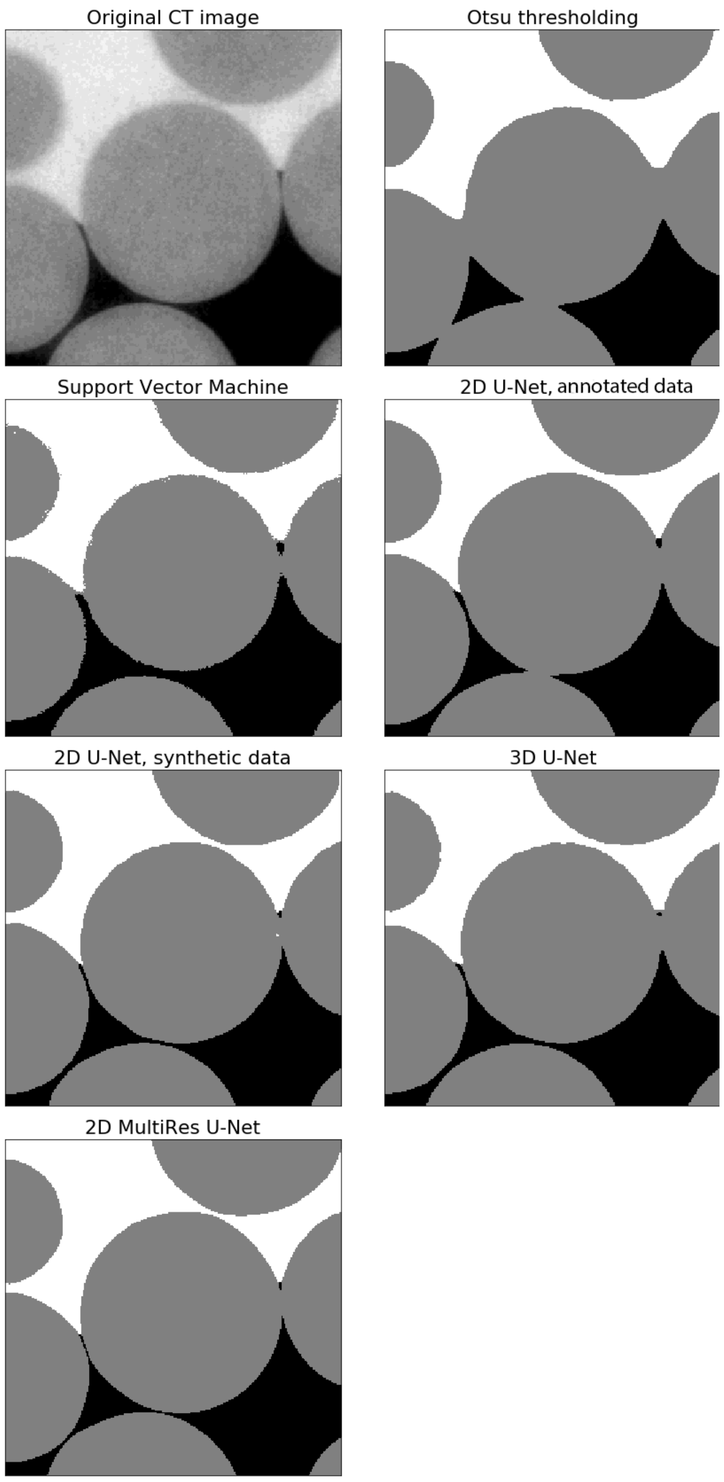


Figure 5.13: Segmentation of intermediate wet glass beads.

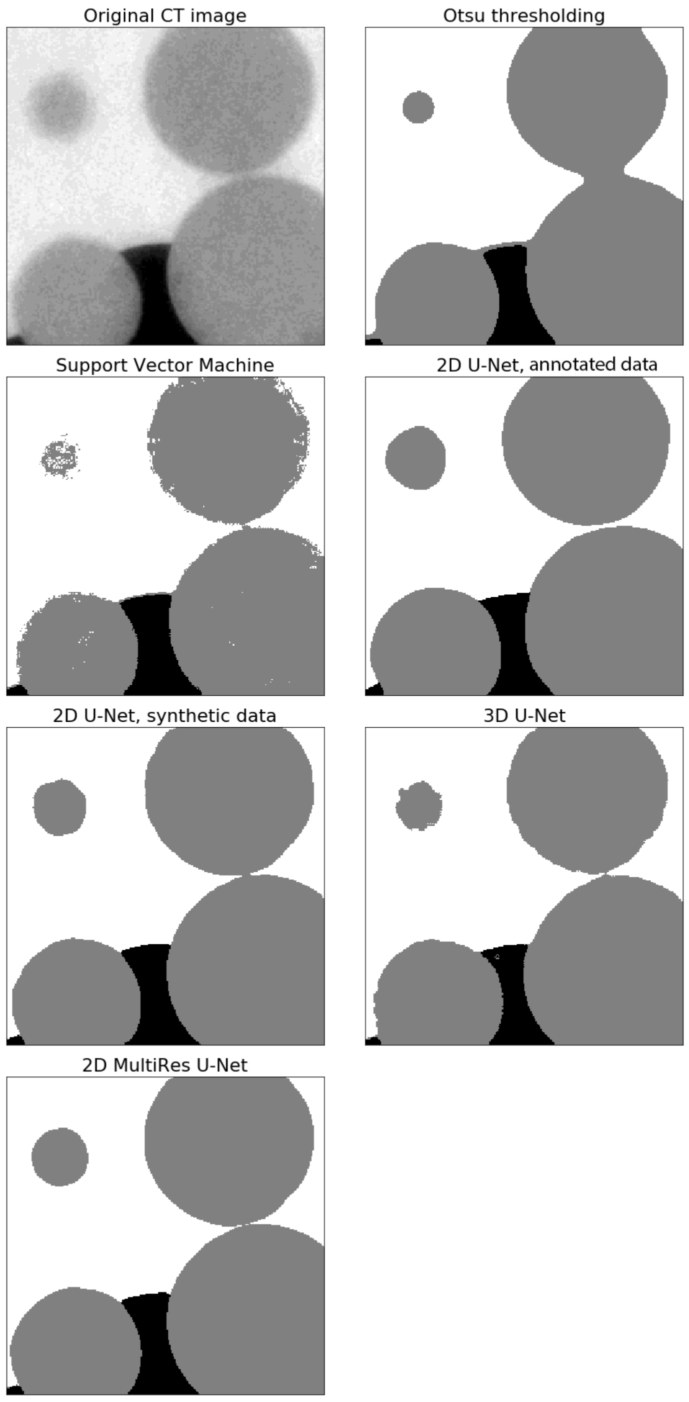


Figure 5.14: Segmentation of intermediate wet glass beads.

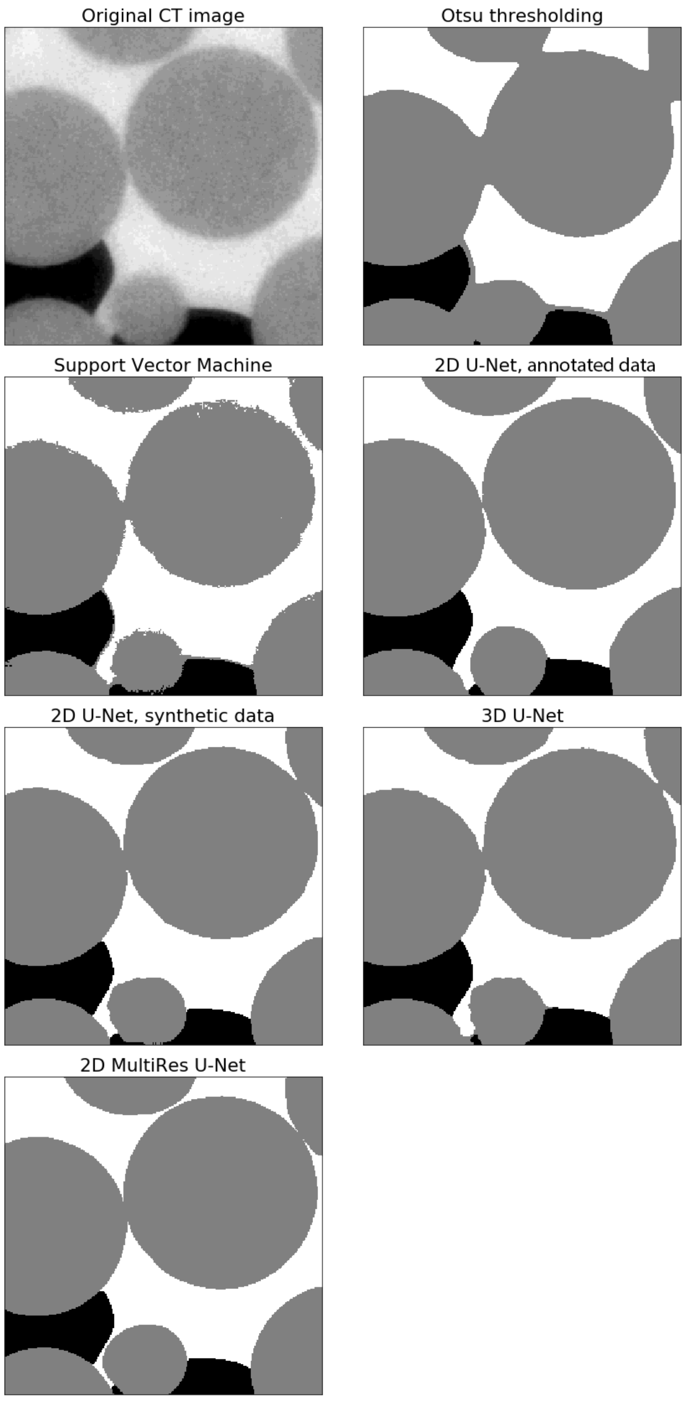


Figure 5.15: Segmentation of intermediate wet glass beads.

In summary, the U-Net I makes the best segmentations overall, but the MultiRes U-Net is not far behind. If the training dataset is extended with more variance in porosity and contact angles, I believe that it will be able to make as good segmentations as U-Net I. This is remarkable, considering that it is trained on a simple synthetic dataset where most of the variance has been created by simple data augmentation.

As expected, the simple Otsu thresholding and the Support Vector Machine fail to create as good segmentations as the deep learning approaches, but they are able to extract the main features. Therefore, it is understandable that they are being used to separate the bulk of the pixels in a multi-step workflow, where the boundaries are being segmented with other approaches, such as watershed segmentation.

U-Net II and 3D U-Net struggle more than anticipated and seem to not have learned the relevant semantics in the data. For example, they often fail to make circular beads, the boundaries are often fuzzy, and small spots are often wrongly predicted in the middle of a different class. They seem to be too dependent on the pixel values and gradients and sometimes unable to see the bigger picture. There could be several reasons for this; the dataset may be too simple and monotonous, and thus makes it unnecessary to be able to generalize in order to achieve good training and validation accuracy. Therefore, it may be enough to learn the local features. It could also be that an optimization of the hyperparameters, e.g. batch size, learning rate and learning schedule, lead to better results.

Another reason for the poor performance of the 3D U-Net could be that it is underfitted, even though the training loss plot does not indicate this. It is trained on the same dataset as the 2D models, but each picture is naturally 16 times bigger than for the 2D models. This means that there are fewer images in the 3D dataset. It, therefore, makes fewer backpropagations than the 2D models, and since the batch size is just eight times smaller than for the 2D models, it only makes half the number of updates on the weights in the network as the 2D models. Therefore, it might need more training data than the 2D models.

It also could be that the models would have performed better if they were trained with a different loss function. As mentioned earlier, the dataset is imbalanced, where there are 8.3% oil, 85.7% solid and 6% water. Therefore, it might be beneficial to train on a loss function that considers the imbalance, for example, weighted categorical cross-entropy or focal loss.

5.4 Areas of Improvement For the Models Trained on Synthetic Images

The images below are included to further investigate the areas where the models trained on synthetic data have room for improvement. The following flaws are not connected to the lack of variety of porosity and contact angles in the training set. These are not a representative selection for the results as a whole, but rather some of the worst segmentations generated.

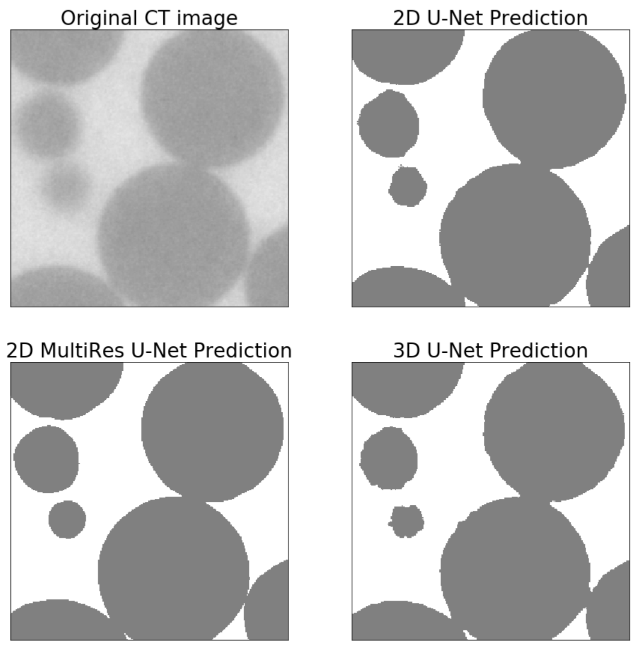


Figure 5.16: Segmentation of one-phase fluid. Water and solid.

Figure 5.16 shows a micro-CT image that only contains glass beads and water. We see that the 2D and 3D U-Net struggles to draw the boundaries between water and solid when the boundaries are blurred and hard to define by pixel values alone. This is often the case at the start or the end of a sphere, i.e., for small cross-sections of the glass beads, where there is noise in the depth dimension of the image that makes it hard to define the boundaries. The MultiRes U-Net, on the other hand, seems to put more emphasis on the semantics and not only on the brightness value and gradient of each pixel, and thus manages to generate a good segmentation. Although it is hard to determine if the boundary is placed in the right place by watching the original image alone, it is at least semantically correct in the sense that the cross-section is circular.

In Figure 5.17, the 3D U-Net struggles when there are brightness non-uniformity in the solids. From the original image, we see that the solids are brighter in the edges than in the center. The pixels in the edges of the solids are mostly predicted correctly. This might be because the predictions of these pixels are affected by the gradient between the white and grey pixels, whereas the pixels a bit closer to the centre are not, and only considers the brightness values. The MultiRes U-Net makes a good segmentation, and the U-Net makes decent a segmentation, but is struggling with the small solid in the top left corner.

In Figure 5.18 we see that the 3D U-Net again struggles with brightness non-uniformity, and predicts water to be in the middle of the solid, and fails to make the

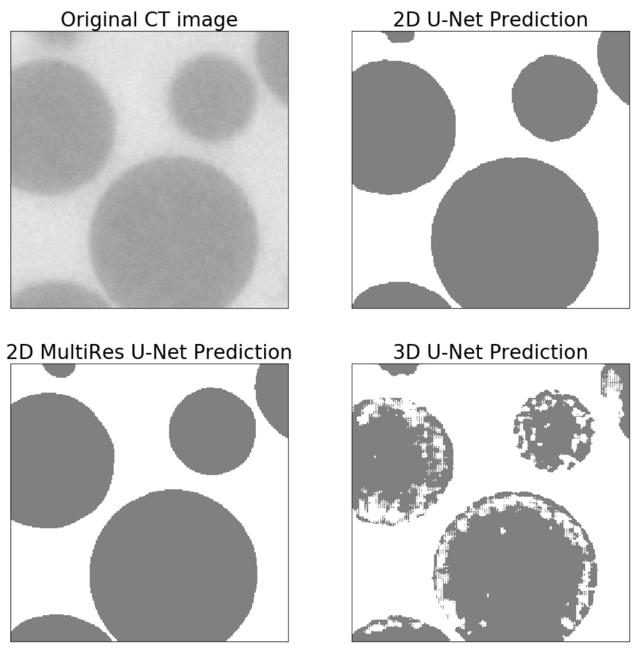


Figure 5.17: Segmentation of one-phase fluid. Water and solid. 3D U-Net struggles..

small solid particle circular. It yet again proves that it has not learned the semantics of the images and is rather using local information, such as brightness values and gradients, to segment the image. The MultiRes U-Net makes an excellent segmentation, and the 2D U-Net makes a good segmentation with the exception of wrongly predicting a small layer of water in the top left corner.

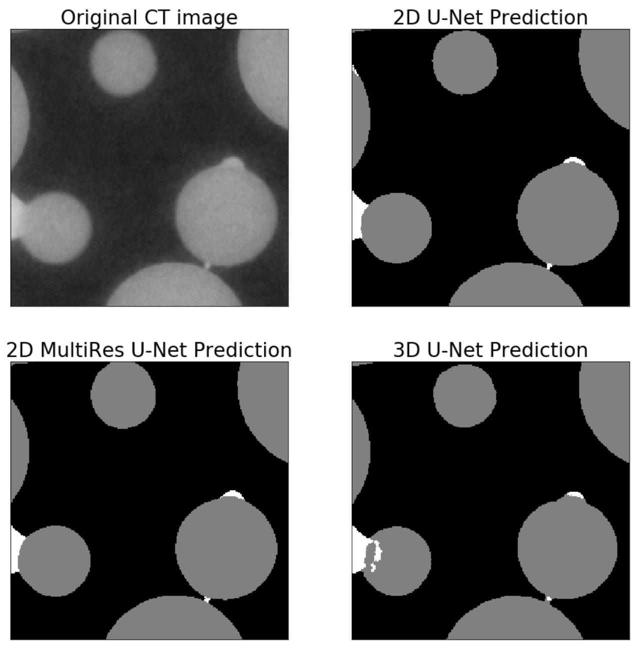


Figure 5.18: Segmentation of two-phase fluid. Oil, water and solid.

5.5 Probability Heatmap

Figure 5.19 shows the probability heatmap for the segmentations done by the four deep learning models on the same image as is segmented in Figure 5.15. The values on the heatmap are the probabilities calculated for each model that the segmented class belongs to that class. I.e., if one pixel on the heatmap has the value 0.8 and the model has predicted the pixel to be solid, the model is 80% certain that the pixel belongs to the solid class.

U-Net I seems to be 100% certain of the segmentation inside all of the glass beads and in the fluid phases, and is only uncertain at the boundaries. The boundaries with uncertainties is quite narrow, which corresponds to the performance we have seen from the model.

U-Net II and the 3D U-Net are also uncertain at the boundaries. The main thing to consider is the high certainties next to low cretainties along the boundary of the small glass bead in the bottom of the image. I.e., they are certain that the spheres should not be circular and that the edges should not be smoothed. Additionally, we see traces of uncertainty in the middle of the glass beads. This suggests again that these networks are considering the local features, such as brightness values and gradients more than the two other models. In other words, the two other models have learned that fluids never occur fluids inside the glass beads, whereas U-NetII and 3D U-Net have not.

The MultiRes U-Net is also 100% sure of the segmentation inside the glass

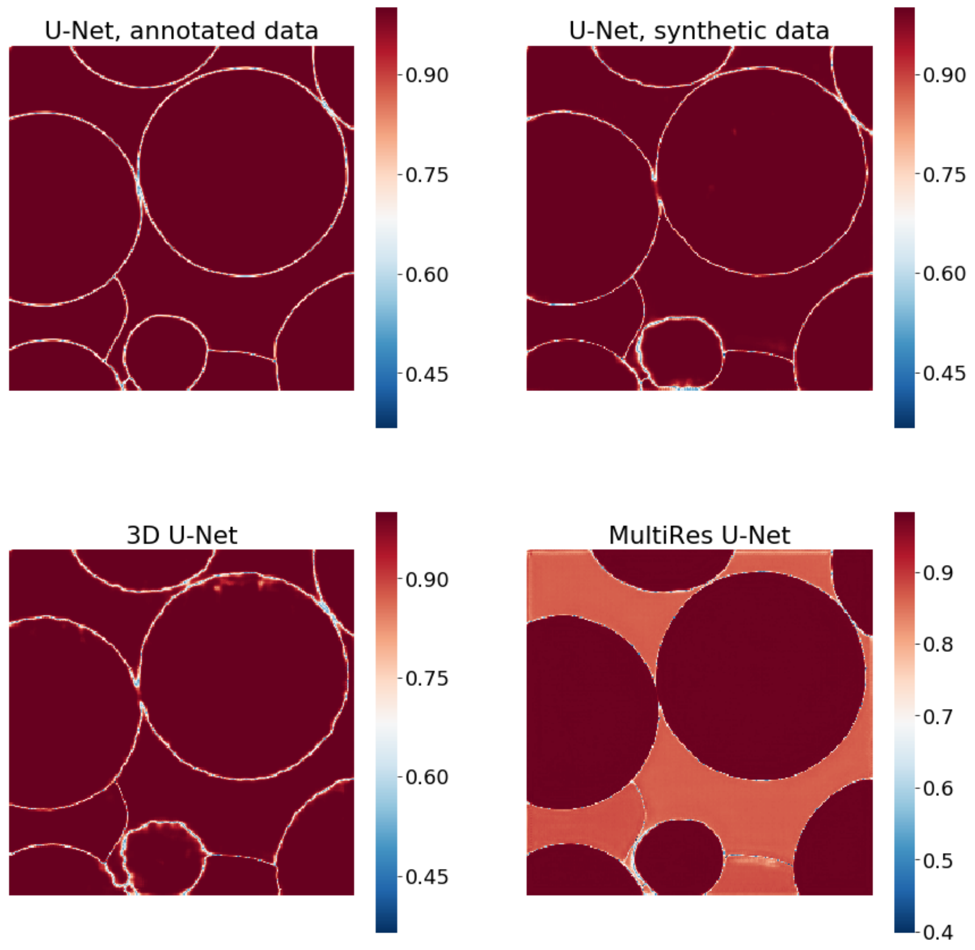


Figure 5.19: Probability heatmap for the segmentations. The heatmaps visualizes the probabilities calculated by the deep learning models in the segmentation. A high value means that the model is certain in its segmentation in the area, and a low value means that the model is uncertain.

beads. Interestingly, it is only about 85% certain of the segmentations in the pore space. However, this does not matter, since 85% is more than enough to make the correct prediction. The uncertainty field along the boundaries are narrower than for all of the other models, which suggest that the model is well familiar with the fact that the edges should be smooth. It does, however, fail to make the bottom glass bead circular, and is 100% certain that it should have a bulge on the left side. It is also quite certain in most of the contact angles, even though they are not correct. This confirms that the reason for failing to recreate the correct contact angles, which especially was evident in the middle left glass bead in the figure, is because of a lack of variety in the dataset. We therefore see that there are both advantages and disadvantages with deep learning models that learn semantics. An advantage is that the model learn important concepts of the images, for example that the edges should be smooth and that there should not be any fluids inside the solids. A disadvantage is that the model perform poorly on data which is not enough represented in the dataset, for example different contact angles.

5.6 Overall Discussion

From the results presented and discussed above, it is evident that the MultiRes U-Net is the best performing model trained on synthetic data when it comes to segmentation of the authentic micro-CT images. Interestingly, it is the worst-performing model on the test set but the best performing on the authentic images. There may be many reasons for this, but the main thing to note is that it is the best generalizing model. The other models may either be overfitted, or depend to heavily on local features instead of the semantics of the images. Potential methods for improving their performance are adding more variety to the data, trying different sized datasets to exclude overfitting, experimenting with different numbers of feature channels, and optimizing the training process by changing the hyperparameters. This includes the batch size, initial learning rate and learning schedule, among others. More methods for improving the results is found in the Further Work section. Because of the long training times for the models trained on synthetic data, ranging from 24 to 30 hours, there was no time to perform many iterations of the training. Approximately five training iterations of each of the three networks were done, where the hyperparameters were changed, and some small modifications to the network were done. The best iteration of each model was selected and tested on the authentic micro-CT images.

From here on, we will only consider the performance of the MultiRes U-Net and U-Net I, since the main focus was not to compare the deep learning models, but simply explore if it is possible to automatically and precisely segment micro-CT images, possibly with the use of synthetic images. Most of the MultiRes U-Net's segmentation flaws may be related to a lack of variety in the training dataset. This was discovered in the initial results, and actions were taken to alleviate this issue. However, the actions were time-consuming since the methods for adding data with higher porosity, and different contact angles both required sphere packs with higher porosities to be the input of the morphological invasion. The increased pore volume naturally led to more calculations in the invasion script. As a consequence, the run time for the morphological invasion increased from approximately 2 hours to more than 24 hours. Because of this, there was only enough time to add one 512^3 grid with high porosity and one 512^3 grid with different wetting conditions. This alleviated the issues, but it still needs to be added more variety to obtain better results. Alternatively, one can remove some of the other sphere packs with the default porosity and wetting conditions so the added data make up a more significant fraction of the dataset.

The quantitative results shows that the MultiRes U-Net struggles with false negatives for oil and water. This means that it frequently predicts solid when the true label is either oil or water. This might be a result from class imbalance and may be fixed by using a loss function that incorporates class weights to account for the imbalance. However, because the training set primarily consists of sphere packs with low porosities, the class imbalance will be reduced when sphere packs with higher porosities are added.

There is a big difference in training time between U-Net I and the MultiRes U-Net. U-Net I finishes the training in approximately 20 minutes, while it for the MultiRes U-Net takes approximately 30 hours. The reason for this is the difference in network complexity and size of the training dataset. In the U-Net I the feature channels range from 16 to 128, while for the MultiRes U-Net it ranges from 32 to 512. Additionally, the added convolutions in the Res paths adds even more complexity to the MultiRes U-Net. The result of this is that while the U-Net I only has 597 603 trainable parameters, the MultiRes U-Net has 7 238 086. An increasing number of feature channels and convolution allows for more complex features to be extracted, but on the other hand, it slows down the training of the models and may make them harder to optimize. A low number of feature channels lead to faster training times, but it may not extract all desired features from the images. However,in most applications, it does not matter if the training takes 20 minutes or 30 hours because it is only necessary to train the model once.

The basis for the training of the U-Net I is 50 images of size 512^2 before cropping and augmentation. For comparison, the training set of synthetic data is comprised of 8650 images before cropping and augmentation. The small dataset consists of manually annotated images, where all of the images come from the same scan. There is not a high variance in, e.g., contact angles, fluid distribution, and brightness values. Therefore, the models trained on this dataset are not expected to produce good segmentations of images with other characteristics when it comes to these parameters. The large, synthetic dataset is created to be more generic, and models trained on it should perform well on a wider range of images than models trained on the small dataset. However, because the models only have been tested on two micro-CT scans from the same scanner, where the only difference was the wetting conditions, the generality of the models have not been adequately tested.

Developing deep learning models may be seen as a balancing act. One wants enough trainable parameters to capture all desired features, but also to balance it against training times and overfitting. Enough data in the dataset makes the model both general and specific enough to perform well on all inputs, but too much data may cause overfitting and unnecessary long training and inference times. An exciting development to keep an eye on in the years to come is the race between image quality versus computing power. When the resolution is doubled in a 3D image, this requires an eightfold increase in computation power not to reduce the run time. It may, therefore, not be the best idea to create too complex networks in order to future proof them. However, even though the MultiRes U-Net may be perceived as complex, if one looks at trainable parameters divided by image size, it has a lower density of trainable parameters than all of the models tested by Wang, Shabaninejad, et al. (2021) except one.

Chapter 6

Conclusion

Traditional segmentation approaches of porous media have indisputable weaknesses. They are both tedious and results in segmentations with operator bias. In this work, these issues are addressed by creating manually annotated and synthetic micro-CT images to obtain training datasets for deep learning approaches. Deep learning makes it possible to automatically segment micro-CT images without the need for pre- or post-processing, and the synthetic dataset has perfect ground truth segmentations, which reduces the operator bias. One deep learning model is trained on the annotated images, and three models are trained on the synthetic images. Finally, they are used to segment authentic micro-CT images, and the results are compared.

All three models trained on the simple, synthetic data are able to make decent segmentations on the authentic micro-CT images of a vial filled with glass beads, oil, and water. However, one of the models, the MultiRes U-Net, is significantly better than the others. It has some issues with segmenting large pore spaces and struggles with the contact angles, but this is likely due to the lack of oil-wet contact angles and large pore spaces in the dataset, and may be mitigated by simply adding images with more variety of contact angles and porosities to the dataset. Despite being trained on simple synthetic images, it is almost able to generate as good segmentations as the U-Net trained on the exact images that the models are tested on.

There are advantages and disadvantages of using both annotated and synthetic images. The models trained on annotated images will be particularly suited to segment images with the same characteristics as the images it is trained on and will learn features that will be hard to recreate in the synthetic images. However, the process of annotating images is tedious, and the model will be affected by the person annotating the images. Models trained on synthetic images eliminate the operator bias, but it may be challenging to capture features tied to specific authentic micro-CT images, which may not be represented in the synthetic dataset.

However, the results obtained answers the research questions in the thesis: It is possible to create a model that may be used to automatically digitize the fluid phases and matrix of micro-CT images precisely. Additionally, it is possible to achieve quality segmentations on authentic micro-CT images with models that are trained on synthetic images.

Chapter 7

Further Work

If I was to continue working on this project or guide someone who wishes to continue this project, these are the next steps I would make.

7.1 Extend Dataset

The biggest problems of the results are connected to lack of variety in the dataset. In order to precisely segment the contact angles, more contact angles must be added to the dataset, because the model is biased as there exists too few in the training set. The same applies for the porosity.

7.2 Optimize Training

Even though it is time consuming for this large dataset, it would be valuable to optimize the training by tuning the hyper-parameters. It could be beneficial to start by tuning the parameters on a subset of the dataset, before eventually fine-tuning on the whole dataset. Both more and less feature channels for all of the models should also be tested, and it would be useful to train on datasets with different size to possibly detect overfitting.

Other loss functions should also be tested. Even though categorical cross-entropy is the go-to loss function for multiclass segmentation, I think that a loss function that incorporate weights to account for class imbalance would be better suited. Suggestions for other loss functions are weighted categorical cross-entropy and categorical focal loss. These are not natively supported by Keras yet, but there exists third-party implementations of them which may be used, or you may implement them yourself.

7.3 Train 3D MultiRes U-Net

There exists a 3D version on the MultiRes U-Net, proposed by Ibtehaz et al. (2020), but due to GPU memory constraints, it was not possible to train this model on the

256x256x16 images. If one has more GPU memory on hand, it would be interesting to see the performance of the 3D version compared to the 2D version.

7.4 Use Deeper 3D Patches

Due to hardware constraints, it was not possible to use larger 3D patches when training the models in this thesis. But if one have more GPU memory, it should be beneficial to use deeper 3D patches. This would allow the model to learn more semantics in the depth dimension, which should benefit the segmentations. Ideally one should be to be able to contain a whole grain in the 3D patch.

7.5 Develop Sophisticated Testing Pipelines

As the model progresses, it would be useful to use a more sophisticated testing methodology on the segmentations of the authentic micro-CT images. This might, however, be complicated, because there are no ground truths to those images. It would, at least, be interesting to quantify the contact angle in both the segmentation and authentic image and compare these. Quantifying the contact angles for the segmentation is straight-forward, and these may be compared to the contact angles obtained using the method proposed by Khanamiri et al. (2020), which generates triangular surfaces using micro-CT images. The triangular surfaces are then smoothed to calculate the contact angles.

7.6 Create More Realistic Dataset

The next step would be to create a more realistic dataset to train the models on. There are several approaches to this.

One approach is to use a process based reconstruction method, as is proposed by Bakke et al. (1997, 2002). These methods reconstruct sandstones by modelling the sandstone-forming geological processes - sedimentation, compaction and diagenesis.

Another approach is to create simple synthetic datasets which encode the information one seeks to extract and letting a cyclic generative adversarial network perform style transfer between the synthetic and raw data - an unsupervised data to content transformation, as is proposed by Ihle et al. (2019).

Bibliography

3.3.9.7. Otsu thresholding — Scipy lecture notes (n.d.). http://scipy-lectures.org/packages/scikit-image/auto_examples/plot_threshold.html. (Accessed on 12/05/2020).

Andrä, Heiko, Nicolas Combaret, Jack Dvorkin, Erik Glatt, Junehee Han, Matthias Kabel, Youngseuk Keehm, Fabian Krzikalla, Minhui Lee, Claudio Madonna, et al. (2013). "Digital rock physics benchmarks—Part I: Imaging and segmentation." In: *Computers & Geosciences* 50, pp. 25–32.

Andresen, Markus and Simen Norby (2019). "Deep Convolutional Encoder-Decoder Networks for Digital Rock Porosity Segmentation." MA thesis. NTNU.

Bakke, Stig and Pål-Eric Øren (June 1997). "3-D Pore-Scale Modelling of Sandstones and Flow Simulations in the Pore Networks." In: *SPE Journal* 2.02, pp. 136–149. ISSN: 1086-055X. DOI: 10.2118/35479-PA. eprint: https://onepetro.org/SJ/article-pdf/2/02/136/2126680/spe-35479-pa.pdf. URL: https://doi.org/10.2118/35479-PA.

— (2002). "Process based reconstruction of sandstones and prediction of transport properties." In: *Transport in porous media* 46.2, pp. 311–343.

Baranau, Vasili and Ulrich Tallarek (2014). "Random-close packing limits for monodisperse and polydisperse hard spheres." In: *Soft Matter* 10 (21), pp. 3826–3841. DOI: 10.1039/C3SM52959B. URL: http://dx.doi.org/10.1039/C3SM52959B.

Berg, Carl Fredrik, Per Arne Slotte, and Hamid Hosseinzade Khanamiri (Sept. 2020). "Geometrically derived efficiency of slow immiscible displacement in porous media." In: *Phys. Rev. E* 102 (3), p. 033113. DOI: 10.1103/PhysRevE.102.033113. URL: https://link.aps.org/doi/10.1103/PhysRevE.102.033113.

Beucher, Serge et al. (1992). "The watershed transformation applied to image segmentation." In: *Scanning microscopy-supplement-*, pp. 299–299.

Bezdek, James C, Robert Ehrlich, and William Full (1984). "FCM: The fuzzy c-means clustering algorithm." In: *Computers & Geosciences* 10.2-3, pp. 191–203.

Bezrukov, Alexander, Monika Bargieł, and Dietrich Stoyan (2002). "Statistical Analysis of Simulated Random Packings of Spheres." In: *Particle & Particle Systems Characterization* 19.2, pp. 111–118. DOI: https://doi.org/10.1002/1521-4117(200205)19:2<111::AID-PPSC111>3.0.CO;2-M. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/1521-4117%28200205%

2919%3A2%3C111%3A%3AAID-PPSC111%3E3.0.CO%3B2-M. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/1521-4117%28200205%2919%3A2%3C111%3A%3AAID-PPSC111%3E3.0.CO%3B2-M.

Bottou, Léon (2010). “Large-scale machine learning with stochastic gradient descent.” In: *Proceedings of COMPSTAT'2010*. Springer, pp. 177–186.

Bre, Facundo, Juan M Gimenez, and Vıctor D Fachinotti (2018). “Prediction of wind pressure coefficients on building surfaces using artificial neural networks.” In: *Energy and Buildings* 158, pp. 1429–1441.

Brownlee, Jason (2019). *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/. (Accessed on 12/07/2020).

Buades, Antoni, Bartomeu Coll, and J-M Morel (2005). “A non-local algorithm for image denoising.” In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 2. IEEE, pp. 60–65.

Cortes, Corinna, Mehryar Mohri, and Afshin Rostamizadeh (2012). “L2 regularization for learning kernels.” In: *arXiv preprint arXiv:1205.2653*.

Cortes, Corinna and Vladimir Vapnik (1995). “Support-vector networks.” In: *Machine learning* 20.3, pp. 273–297.

Crammer, Koby and Yoram Singer (2001). “On the algorithmic implementation of multiclass kernel-based vector machines.” In: *Journal of machine learning research* 2.Dec, pp. 265–292.

Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik (2014). “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.

Gómez, Raúl (2018). *Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names*. https://gombru.github.io/2018/05/23/cross_entropy_loss/. (Accessed on 12/07/2020).

Hazlett, RD (1995). “Simulation of capillary-dominated displacements in microtomographic images of reservoir rocks.” In: *Transport in porous media* 20.1, pp. 21–35.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). “Deep residual learning for image recognition.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hilpert, Markus and Cass T. Miller (2001). “Pore-morphology-based simulation of drainage in totally wetting porous media.” In: *Advances in Water Resources* 24.3. Pore Scale Modeling, pp. 243–255. ISSN: 0309-1708. DOI: https://doi.org/10.1016/S0309-1708(00)00056-7. URL: https://www.sciencedirect.com/science/article/pii/S0309170800000567.

Ibtehaz, Nabil and M Sohel Rahman (2020). “MultiResUNet: Rethinking the U-Net architecture for multimodal biomedical image segmentation.” In: *Neural Networks* 121, pp. 74–87.

Ihle, Stephan J, Andreas M Reichmuth, Sophie Girardin, Hana Han, Flurin Stauffer, Anne Bonnin, Marco Stampanoni, Karthik Pattisapu, János Vörös, and Csaba Forró (2019). "Unsupervised data to content transformation with histogram-matching cycle-consistent generative adversarial networks." In: *Nature Machine Intelligence* 1.10, pp. 461–470.

Jain, Anil K, Jianchang Mao, and K Moidin Mohiuddin (1996). "Artificial neural networks: A tutorial." In: *Computer* 29.3, pp. 31–44.

Joram, Nickson (n.d.). *Morphological Operations in Image Processing | by Nickson Joram | Medium*. `https://medium.com/@himnickson/morphological-operations-in-image-processing-cb8045b98fcc`. (Accessed on 12/05/2020).

Jordan, Michael I and Tom M Mitchell (2015). "Machine learning: Trends, perspectives, and prospects." In: *Science* 349.6245, pp. 255–260.

Ketcham, Richard A and William D Carlson (2001). "Acquisition, optimization and interpretation of X-ray computed tomographic imagery: applications to the geosciences." In: *Computers & Geosciences* 27.4, pp. 381–400.

Khanamiri, Hamid Hosseinzade, Per Arne Slotte, and Carl Fredrik Berg (2020). "Contact Angles in Two-Phase Flow Images." In: *Transport in Porous Media* 135.3, pp. 535–553.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980*.

Kiourt, Chairi, George Pavlidis, and Stella Markantonatou (Apr. 2020). "Deep learning approaches in food recognition." In:

Kjerland, Øyvind (2017). "Segmentation of Coronary Arteries from CT-scans of the heart using Deep Learning." MA thesis. NTNU.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning." In: *nature* 521.7553, pp. 436–444.

Li, Fei-Fei, Justin Johnson, and Serena Yeung (2017). *Lecture 11: Detection and Segmentation*. `http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf`. (Accessed on 12/05/2020).

Long, Jonathan, Evan Shelhamer, and Trevor Darrell (2015). "Fully convolutional networks for semantic segmentation." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440.

Lubachevsky, Boris D and Frank H Stillinger (1990). "Geometric properties of random disk packings." In: *Journal of statistical Physics* 60.5, pp. 561–583.

Misra, Rishabh (2019). *Support Vector Machines — Soft Margin Formulation and Kernel Trick | Towards Data Science*. `https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-and-kernel-trick-4c9729dc8efe`. (Accessed on 12/05/2020).

Nielsen, Michael A (2015). *Neural networks and deep learning*. Vol. 2018. Determination press San Francisco, CA.

Otsu, Nobuyuki (1979). "A threshold selection method from gray-level histograms." In: *IEEE transactions on systems, man, and cybernetics* 9.1, pp. 62–66.

Pröve, Paul-Louis (2017). *An Introduction to different Types of Convolutions in Deep Learning | Towards Data Science*. `https://towardsdatascience.com/`

types-of-convolutions-in-deep-learning-717013397f4d. (Accessed on 12/05/2020).

Raid, AM, WM Khedr, MA El-Dosuky, and Mona Aoud (2014). “Image restoration based on morphological operations.” In: *International Journal of Computer Science, Engineering and Information Technology (IJCSEIT)* 4.3, pp. 9–21.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). “U-net: Convolutional networks for biomedical image segmentation.” In: *International Conference on Medical image computing and computer-assisted intervention*. Springer, pp. 234–241.

Rosebrock, Adrian (2016). *Intersection over Union (IoU) for object detection - PyImageSearch*. https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. (Accessed on 12/11/2020).

Santurkar, Shibani, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry (2018). “How does batch normalization help optimization?” In: *Advances in neural information processing systems* 31, pp. 2483–2493.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2015). “Going deeper with convolutions.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Szeliski, Richard (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media.

Wang, Ying Da, Martin Blunt, Ryan Armstrong, and Peyman Mostaghimi (Feb. 2021). “Deep Learning in Pore Scale Imaging and Modeling.” In: *Earth-Science Reviews* 215. DOI: 10.1016/j.earscirev.2021.103555.

Wang, Ying Da, Mehdi Shabaninejad, Ryan T Armstrong, and Peyman Mostaghimi (2021). “Deep neural networks for improving physical accuracy of 2D and 3D multi-mineral segmentation of rock micro-CT images.” In: *Applied Soft Computing* 104, p. 107185.

Wildenschild, Dorthe and Adrian P. Sheppard (2013). “X-ray imaging and analysis techniques for quantifying pore-scale structure and processes in subsurface porous medium systems.” In: *Advances in Water Resources* 51. 35th Year Anniversary Issue, pp. 217–246. ISSN: 0309-1708. DOI: https://doi.org/10.1016/j.advwatres.2012.07.018. URL: http://www.sciencedirect.com/science/article/pii/S0309170812002060.

Xiang, Mars (2017). *Convolutions: Transposed and Deconvolution Medium*. https://medium.com/@marsxiang/convolutions-transposed-and-deconvolution-6430c358a5b6. (Accessed on 12/05/2020).

Ådne Årevik Vikdal

Multi-Phase Segmentation of Imaged Fluid Distribution in Porous Media Using Deep Learning

NTNU
Norwegian University of
Science and Technology