ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA BIOMEDICÍNSKÉHO INŽENÝRSTVÍ

Katedra informačních a komunikačních technologií v lékařství

Segmentace ultrazvukových obrazů aterosklerotických plátů s využitím neuronových sítí

Segmentation of ultrasound images containing atherosclerotic plaques using neural networks

Bakalářská práce

Studijní program:	Informatika a kybernetika ve zdravotnictví
Studijní obor:	Biomedicínská informatika
Autor práce:	David Pilný
Vedoucí práce:	Ing. Michal Reimer

Kladno 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Pilný**	Jméno:	**David**	Osobní číslo:	**499967**
Fakulta:	**Fakulta biomedicínského inženýrství**				
Garantující katedra:	**Katedra informačních a komunikačních technologií v lékařství**				
Studijní program:	**Informatika a kybernetika ve zdravotnictví**				
Studijní obor:	**Biomedicínská informatika/Informační a komunikační**				

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Segmentace ultrazvukových obrazů aterosklerotických plátů s využitím neuronových sítí

Název bakalářské práce anglicky:

Segmentation of ultrasound images containing atherosclerotic plaques using neural networks

Pokyny pro vypracování:

The aim of the bachelor thesis is the segmentation of ultrasound images of atherosclerotic plaques using neural networks. As part of your bachelor's thesis, learn about currently used deep learning methods and their use for the segmentation of medical images. Subsequently, propose a procedure for segmentation of defined objects in provided images of atherosclerotic plaques. Focus also on the analysis of the provided dataset and suggest possibilities of augmentation of the images to achieve better results. To verify the functionality of both the correctly chosen architecture and the dataset augmentation, perform testing using expertly annotated images. Finally, evaluate the accuracy of the segmentation and possibly suggest options for modifying the model you used in order to increase the accuracy of the segmentation.

Seznam doporučené literatury:

[1] Norouzi, A., Rahim, M. S., Altameem, A., Saba, T., Rad, A. E., Rehman, A., & Uddin, M. (2014). Medical image segmentation methods, algorithms, and applications. *IETE Technical Review*, *31*(3), 199–213.
[2] Wang, R., Lei, T., Cui, R., Zhang, B., Meng, H., & Nandi, A. K. (2022). Medical image segmentation using Deep Learning: A Survey. *IET Image Processing*.
[3] Shorten, C., Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. *J Big Data* **6**, 60 (2019).

Jméno a příjmení vedoucí(ho) bakalářské práce:

Ing. Michal Reimer

Jméno a příjmení konzultanta(ky) bakalářské práce:

doc. Mgr. Radim Krupička, Ph.D.

Datum zadání bakalářské práce: **06.01.2023**
Platnost zadání bakalářské práce: **18.09.2024**

....
doc. Ing. Karel Hána, Ph.D.
podpis vedoucí(ho) katedry

....
prof. MUDr. Jozef Rosina, Ph.D., MBA
podpis děkana(ky)

Prohlášení

Prohlašuji, že jsem práci s názvem *Segmentace ultrazvukových obrazů aterosklerotických plátů s využitím neuronových sítí* vypracoval samostatně a použil k tomu úplný výčet citací použitých pramenů, které uvádím v seznamu přiloženém k práci. Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Kladně 18.5.2023

David Pilný

Poděkování

Rád bych poděkoval svojí mamince, díky které jsem mohl strávit celý týden psaním bakalářské práce v malebné vesničce Žichovice na prahu Šumavy, kde jsem měl klid a přístup k přírodě pro čerpání můzy, a která mě celý týden krmila všelijakým výborným jídlem a zásobovala litry a litry cold brew pro lepší soustředění.

Také bych rád poděkoval mé přítelkyni, která je mi obecně velkou oporou a která se navíc zrovna v týden nejintezivnějšího psaní této práce rozhodla odjet na Ukrajinu a nerozptylovat mě tak svojí bezmeznou krásou.

Na závěr bych chtěl poděkovat především svému vedoucímu panu Ing. Michalu Reimerovi za odborné vedení práce a cenné rady, které mi pomohly tento výplod zkompletovat.

Abstrakt

Segmentace ultrazvukových obrazů aterosklerotických plátů s využitím neuronových sítí

Tato bakalářská práce se zaměřuje na využití konvolučních neuronových sítí pro segmentaci aterosklerotických plátů. Společně s využitím pro autonomní řízení vozidel, detekci objektů a vyhodnocováním medicínských obrazů, tvoří segmentace snímků významnou část oboru počítačového vidění. Práce srovnává vliv na výkon jednotlivých architektur a trénovacích parametrů a zkoumá, jak dobře jsou konvoluční sítě schopné provádět medicínskou segmentaci.

Přehled současného stavu segmentace postavené na konvolučních sítích, včetně přehledu relevantní literatury a existujících modelů, je prezentován na začátku této práce. Následně bylo navrženo 12 modelů, které se lišily strukturou architektury a zvolenými trénovacími parametry. Tyto modely byly testovány na různých testovacích datasetech (vygenerovaných augumentací původního testovacího datasetu) pro otestování variability modelu. Dodatečně jsou podrobněji zkoumány také metody jako augmentace dat nebo dolaďování (fine-tuning).

Výsledky experimentů ukazují, že konvoluční neuronové sítě jsou schopné při segmentaci aterosklerotických plátů dosáhnout velmi vysoké přesnosti. Nejlépe natrénovaný model, aplikující architekturu U-net a trénovaný na augmentovancýh datech, dosahuje průměrné hodnoty IOU 0,9575 při segmentaci pozadí, 0,8272 při segmentaci plátů, 0,726 při segmentaci průsvitů a 0,8384 při segmentaci artefaktů, na testovacích datech. Tyto poznatky demonstrují efektivitu použití konvolučních sítí při segmentačních úlohách a indikují potenciální rozsah praktického využití.

Tato práce celkově poskytuje podrobnou analýzu využití konvolučních sítí pro segmentaci medicínských obrazů, včetně shrnutí použitých architektur a trénovacích metod. Výsledky ukazují, jak dobře si konvoluční sítě vedou na této úloze a indikují, že konvoluční neuronové sítě mohou dále nacházet využití napříč různými sférami. Zjištěné poznatky navíc dále naznačují možnost aplikace ostatních metod optimalizace, které však v této práci nebyly podrobněji zkoumány.

Klíčová slova

U-net, konvoluční sítě, augmentace dat, ateroskleróza, segmentace

Abstract

Ultrasound atherosclerotic plaque segmentation using neural networks

This bachelor thesis explores the use of convolutional neural networks (CNNs) for atherosclerotic plaque segmentation tasks. With applications in autonomous vehicles, object detection, and medical imaging, image segmentation is a significant area of computer vision. This thesis compares the performance of various architectures and training parameters in order to investigate how well CNNs perform medical segmentation.

An overview of the current state of CNN-based image segmentation, including a review of pertinent literature and existing models, is presented at the outset of the thesis. Subsequently, 12 models were designed which slightly differed in architecture structure and training parameters. These models were then tested on various testing datasets (generated by augmentation of the original dataset) to examine model's variability. Additionally, various training methods like data augmentation and fine-tuning are investigated.

The experiments' findings demonstrate that CNNs are very efficient at segmenting atherosclerotic plaques, achieving high accuracy and performing well on the test set. The model that performs the best averages with IOU of 0.9575 in the background segmentation, 0.8272 in the plaque segmentation, 0.726 in the lumen segmentation, and 0.8384 in the artifacts segmentation on the test sets using a U-Net architecture with data augmentation. These findings show how effective CNNs can be at performing image segmentation tasks and indicate a wide range of practical uses.

Overall, this thesis offers a thorough analysis of CNNs' use for medical image segmentation, including an assessment of various architectures and training methods. The outcomes show how well CNNs perform for this task and indicate that they could find widespread use across a variety of sectors and domains. Also, more methods to further optimize the performance were suggested.

Key words

U-net, convolutional networks, data augmentation, atherosclerosis, segmentation

Acronyms

Acronym	**Meaning**
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
FCN	Fully Convolutional Network
VGG	Visual Geometry Group
ELU	Exponential Linear Unit
NAS	Neural Architecture Search
LDL	Low-density lipoprotein
API	Application Programming Interface
PNG	Portable Network Graphics
ReLU	Rectified Linear Unit

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Convolutional neural networks (CNNs) have revolutionized the field of image recognition, classification and segmentation, especially in the last decade. CNNs have proven to be a useful tool for a variety of applications, from identifying faces in photographs to spotting tumors in medical images.

This thesis examines the capabilities and constraints of convolutional neural networks for the segmentation of atherosclerosis-related medical images. Specifically, I will investigate how various training parameters, data augmentation and architectures affect the performance of convolutional neural network, and how these models can be fine-tuned to improve accuracy.

Along with the empirical analysis, I will also provide a brief overview of the fundamental theory and ideas behind CNNs - convolutional layers, pooling layers, and fully connected layers, as well as more sophisticated methods like transfer learning and data augmentation, will be explained as the basic building blocks of a CNN.

Overall, this thesis aims to provide a thorough description of methods for training CNN for specific tasks and analysis of individual models' performance.

1.1 Motivation

This thesis relates to my previous semestral projects, which were focused on the theory behind CNNs, designing the first convolutional model and creating a web interface for the trained model, so a person with a minimal technical background is able to use the model, too. My motivation for choosing this topic was the desire to understand, how this interesting technology works and what is the "magic" behind it. Since I already gained some knowledge while working on my semestral projects, I decided to continue in my work within the frame of my bachelor thesis and provide more thorough description of the process.

1.2 What is atherosclerosis?

A common condition known as atherosclerosis is characterized by the accumulation of fatty plaques, cholesterol, and other substances on the inner walls of arteries. Over time, this condition can cause the arteries to narrow and harden. The Greek words "athero" (meaning gruel or paste) and "sclerosis" (meaning hardness) are where the word "atherosclerosis" originates. [1, 2]

The damage to the endothelium, the inner lining of the arteries, which is typically the first sign of atherosclerosis, can be brought on by conditions like high blood pressure, smoking, high cholesterol, diabetes, or inflammation. Low-density lipoprotein (LDL) cholesterol and other particles may be able to enter the artery wall and accumulate there when the endothelium is damaged. These substances have the potential to accumulate into a fatty plaque over time, thickening and stiffening the artery wall. [1, 3]

Since atherosclerosis is a progressive condition that frequently takes years to develop, symptoms might not appear until the disease is far along. Tobacco cessation, a healthy diet, regular exercise, controlling blood pressure and cholesterol levels, and taking medications as directed by a doctor are just a few lifestyle changes and medical procedures that can help to slow or reverse the progression of the disease. [3, 4]

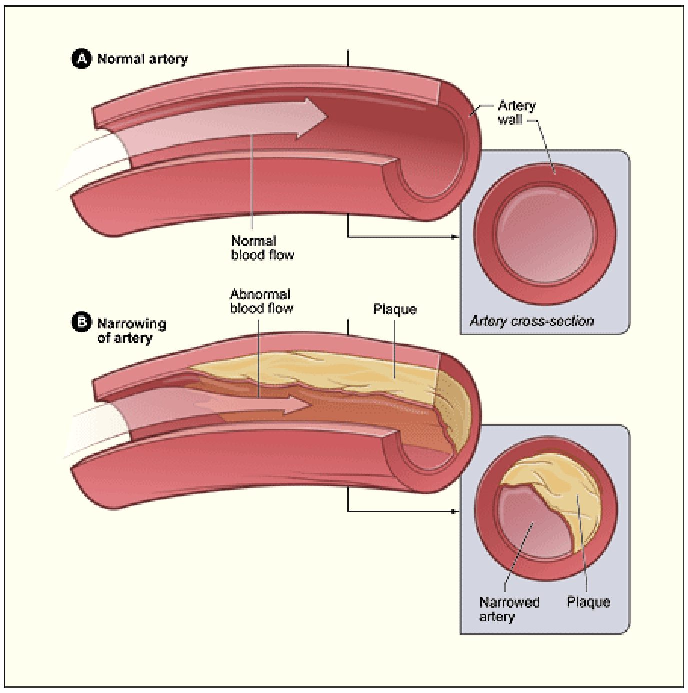


Figure 1.1: Normal artery and an artery with plaque buildup, from [5]

Chapter 2

Objectives

The objective of my thesis is to design optimal methods for atherosclerosis plaque segmentation using convolutional neural networks. The process can be divided into several phases.

2.1 Research

In this part the prime objective is to get familiar with the current state of application of convolutional neural networks in the analysis of medical image data and with methods of desgining such networks.

2.2 Dataset Analysis

Before the training part itself, data must be preprocessed so they are compatible with the network's input layer. Here I will analyse the dataset, design methods of loading the dataset a transform the dataset so it can be "fed" into the network.

2.3 Data Augmentation

To heighten accuracy, a bigger dataset may be needed. It might also happen the provided dataset will be too monolithic and the model will not be prepared for all possible situations that might arise in production. For that reason one of the objectives is to augment the existing data to generate new ones and thus broaden networks "learning material". Augmentation might include image rotation, zooming, shearing, flipping, shifting and other modifications

2.4 Network Design

This part's objective is to select an optimal architecture for the image segmentation and design its layers so the result is the most optimal one. This part can be further divided into research part, where I will gather existing data about various methods, and the practical part, where I will apply different architectures and experiment with them.

2.5 Results' Evaluation

Once the networks output results, they need to be evaluated so the model can be eventually redesigned or the training parameters adjusted. Result evaluation consists from simple "first-glance" analysis to elaborate statistical analysis.

Chapter 3

Theory Behind Neural Networks

Artificial neural networks proved they can be a useful tool in a plenty of prediction and classification tasks. They were especially effective solving complex problem where traditional algorithms with simpler logic fail. Without artificial neural networks (ANN), there could not have been such progress in area of pattern recognition in healthcare or in protein structure prediction, as we have seen in the recent years. [6, 7]

A neural network is a type of machine learning algorithm that is modeled after the structure and function of the human brain. In this chapter, I will explain necessary basics of how the ANNs work. [6]

3.1 Perceptron

The reason ANNs are called the way they are called, is because they model structure of human neural network. Both networks process an input that leads to a specific output. The human brain is made of dendrite connected neurons, which serve as an input, and axons, that emit some output. The biological neuron transmits the signal only after the signal reach a specific threshold. Similarly , the artificial neural networks are made of connected artificial neurons, sometimes called nodes. [6]

One or more input nodes that receive input signals and one output node that generates output signals make up the fundamental building blocks of a perceptron. The strength of the input signal is determined by the weighted connection, that connects each input node to the output node. The weighted input signals are added together to calculate the perceptron's output, which is then processed through a step or sigmoid activation function to make it nonlinear. [6, 8]

The perceptron algorithm is an example of supervised learning, so in order to learn how to make predictions, it needs labeled training data. In order to reduce the discrepancy between the predicted output and the actual output, the weights of the connections between the input nodes and the output node are changed during training. This process continues until the perceptron reaches a certain level of accuracy on the training data. [6, 8, 10]

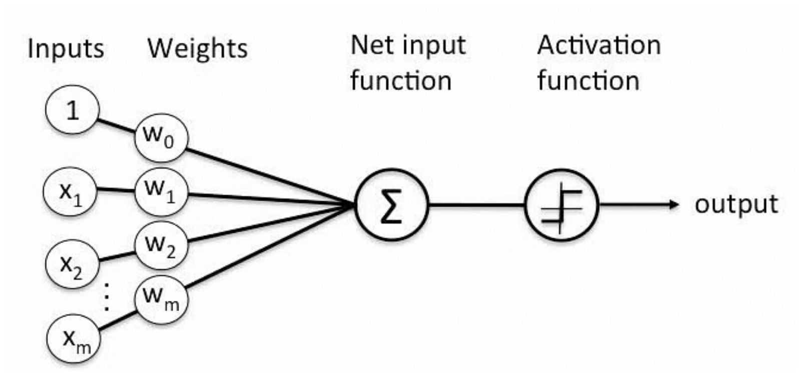


Figure 3.1: Model of node with m weighted inputs, from [9]

When trying to predict which of two classes an input belongs to, perceptrons are frequently used in binary classification problems. Additionally, they can be applied to regression issues where the objective is to forecast a continuous output value. Despite being relatively straightforward compared to other neural network architectures, perceptrons have been used in a number of applications, including image and speech recognition as well as natural language processing. [6, 10]

3.2 Activation function

An activation function is used to affect a neuron's output in a neural network. Based on the inputs it receives, it acts as a kind of "filter" to decide whether the neuron should be activated or not.

Each neuron in a neural network receives input signals from other neurons and bases its output signal on the weighted sum of these input signals. To decide whether a neuron should "fire" or activate and send its signal to the following layer of the network, the activation function is applied to the output signal.

The effectiveness and precision of a neural network can be significantly impacted by the choice of activation function. Activation procedures frequently used include:

ReLU (Rectified Linear Unit) A function that, if the input is positive, returns the value and, if it is negative, returns 0. It has been demonstrated that using this function will enhance the performance of neural networks, and it is frequently used in deep learning models.

Sigmoid A function that converts any value entered into a number between 0 and 1. When the objective is to predict a binary output (such as yes or no), this function is frequently used in binary classification problems.

Softmax A function that converts a list of real numbers into a probability distribution, with the result that the probability sum is 1. When trying to predict the likelihood of each potential class in multiclass classification problems, this function is frequently used.

3.3 ANN architecture

Typical ANN can be divided into input layer, hidden layers and output layer. Data are accepted by the input layer, hidden layer(s) (there is no limit on number of hidden layers) analyzes and process data, and the result is given by the output layer. They are called hidden layers because just like in human vision they covertly process the digitalized object into a form of input signal. For instance, when we see four lines connected as a square, we know immediately it is a square. We do not detect four independent lines that do not have any relationship to one another. Our brain is aware of the information as an input layer and not as hidden layers. By adding hidden layers we increase ability of the network to analyze complex patterns. Therefore, training an ANN with numerous hidden layers is also know as "deep learning". [6]

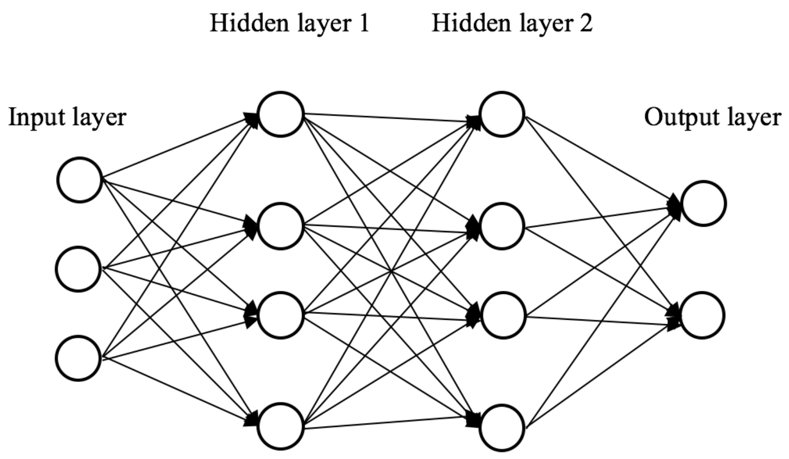


Figure 3.2: ANN with 4 layers, from [11]

3.3.1 Dense layers

A dense layer, also referred to as a fully connected layer, is a type of layer in a neural network where every neuron in the layer is connected to every neuron in the layer before it. As a result, each neuron in the layer receives a weighted sum of its predecessor's neurons' outputs as input. [12]

Using an optimization algorithm like gradient descent, the weights in a dense layer are learned during training in order to reduce the discrepancy between the predicted output and the actual output. [12]

3.4 Convolution

Many disciplines, including deep learning, image processing, and signal processing, use the mathematical operation of convolution. It entails applying a sliding window, referred to as a kernel or filter, to a signal or image, multiplying the window's values by the corresponding values in the signal or image, and adding the results to create a new signal or image. [13]

Convolution is a technique used in signal processing to filter a signal by removing noise or unwanted frequencies. A filter kernel that emphasizes or de-emphasizes particular frequencies is convolved with the input signal to create a new, filtered signal. [13, 14]

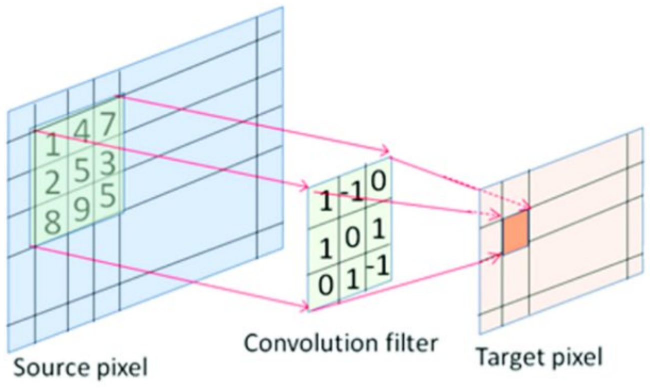


Figure 3.3: Depiction of the convolution layer, from [15]

Convolution is also used in image processing to extract features from an image. A new pixel value is created in the output image at each location where the filter kernel is moved across the image by summing the product of the kernel values and the corresponding image pixel values. To identify various features in the image, such as edges, corners, and textures, this procedure can be repeated with various kernels. [13, 14]

As for example, let us say we have 2 convolutional kernels:

$$(1)\quad \begin{pmatrix} 0.25 & 0.5 & 0.25 \\ 0.0 & 0.0 & 0.0 \\ -0.25 & -0.5 & -0.25 \end{pmatrix} \qquad (2)\quad \begin{pmatrix} 0.25 & 0.0 & -0.25 \\ 0.5 & 0.0 & -0.5 \\ 0.25 & 0.0 & -0.2 \end{pmatrix}$$

When we apply these two kernels on an image, we get these results:

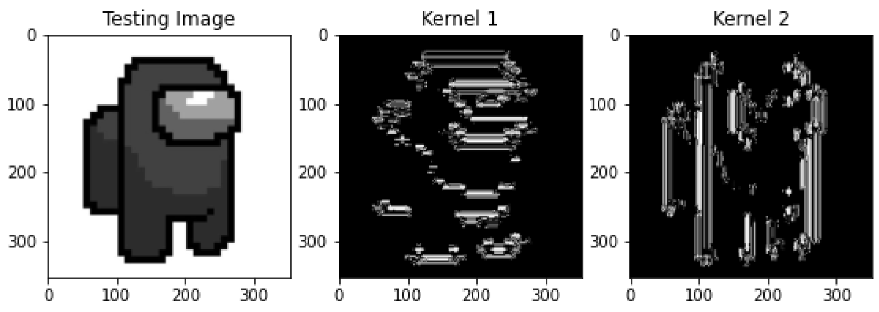


Figure 3.4: Feature extraction with different convolutional kernels

As shown in the figure above, by applying different kernels, different feature can be extracted. In this example, horizontal edges were extracted by kernel 1, vertical edges were extracted by kernel 2. [1]

[1]Testing image taken from DeviantArt - https://www.deviantart.com/nimowerytheking

3.5 Convolutional Neural Networks

Convolution in CNNs entails applying a filter or kernel to an input image or feature map to create a fresh feature map as the result. A single output value is produced for each location in the output feature map by sliding the filter over the input, multiplying each value in the filter by its corresponding value in the input, and summing the results. The size of the input, the size of the filter, and the stride of the filter all affect the size of the output feature map. [13]

By identifying patterns and edges, convolutional operations in CNNs are used to extract features from input data, such as images. Backpropagation is a technique used to teach the weights of the filters so that the network can learn to recognize particular features in the input data. This makes CNNs very good at tasks involving working with image data, such as object detection and image recognition. [13]

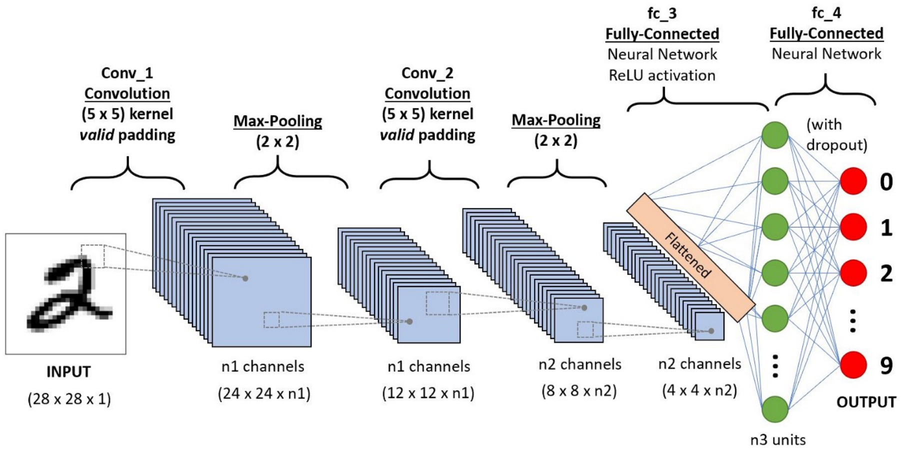


Figure 3.5: CNN for number classification, from [16]

3.5.1 Pooling layer

The convolved feature's spatial size is decreased thanks to the Pooling layer. A significant reduction in the dimensions of the data has resulted in a decrease in the amount of computing power needed to process it. Typically, a Pooling Layer is applied following a Convolutional Layer. The main objective of this layer is to reduce the size of the convolved feature map in order to save on computational costs, which is achieved by reduction of connections between layers and working independently on each feature. Reducing the connections between layers and working independently on each feature map achieves this. Depending on the mechanism used, there are various types of pooling operations. [17]

The Max Pooling feature map is used to determine the largest element. Using average pooling, the elements within an image segment of a specific size are averaged. The total sum of the elements in the predefined section is calculated using sum pooling. Connecting the Convolutional Layer and the FC Layer is frequently done using the Pooling Layer. [17]

3.5.2 Dropout

A dropout layer is used in CNNs, in which a few neurons are removed from the neural network during the training phase, resulting in a smaller model, to avoid overfitting. [2] The dropout layer acts as a mask, eliminating some neurons' contributions to the subsequent layer while maintaining the functionality of all other neurons. If we apply a dropout layer to the input vector, some of its characteristics are eliminated; however, if we apply it to a hidden layer, some hidden neurons are eliminated. [17, 18]

Because they avoid overfitting on the training data, dropout layers are crucial in the training of CNNs. If they are absent, the first set of training samples has a disproportionately large impact on learning. As a result, traits that only show in subsequent samples or batches would not be learned. [18]

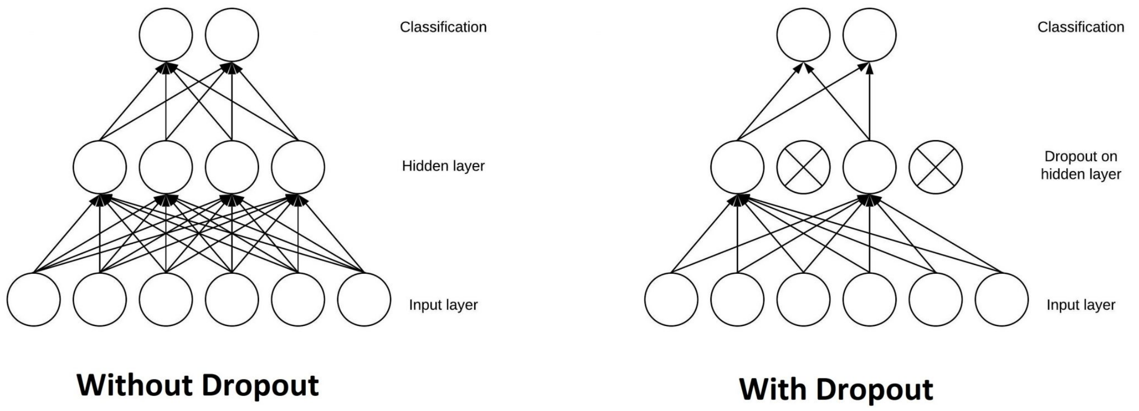


Figure 3.6: Neurons with and without dropout, from [19]

[2]when a model performs well on training data but not on new data

Chapter 4

Current State

4.1 History of Medical Segmentation Using CNNs

In order to connect with current CNN architectures for segmentation, it is necessary to establish a historical connection with image classification. Many innovations in image classification came first, and only then were they applied to segmentation. Because a CNN for voxel-wise classification can be used directly for image classification and vice versa, this is possible. Voxel-wise classification is a technique in image analysis that involves assigning a label or class to each voxel, or 3D pixel, in an image. This method is frequently applied in medical imaging, where the objective is frequently to recognize and separate various bodily structures or tissues. [20, 21]

Voxel-wise classification was explicitly used in the first CNN-based segmentation technique. A more effective implementation, which obtained the effects of pooling using sparse convolutions, was introduced in 2014. The voxel-wise learning paradigm could now support output patch sizes greater than 1, without altering anything. One drawback of this implementation was the increased requirement for memory. This computational advancement was not possible if the explicit voxel-wise implementation consumed all memory. Contrary to parallel observations in image classification, such as the emergence of deeper VGG-nets, it was actually only possible to reap the computational benefits by shrinking the CNN (e.g., reducing the depth, width, or RF). [21, 22, 23]

Fully-convolutional networks (FCNs), as they are known, saved the day in 2015. They kept the original pooling implementation in conjunction with a fresh and teachable upsampling operation, i.e., transposed convolution, as opposed to using sparse convolutions. The FCN-32s, the simplest variant, learned a 32-times upsampling directly from the outcome prediction. By first upsampling the most coarse prediction to and combining it with the prediction on an intermediate resolution, this was further improved. Only then was the original image resolution (i.e., the FCN-16s and FCN-8s) upsampled. As a result, skip connections were added as a side effect in segmentation, much like the development of residual connections in image classification discussed above. [21, 24]

From the FCN, it took only a small step to reach U-Net and DeepMedic, the flag ships of medical imaging. Instead of upsampling (intermediate) predictions, (intermediate) feature maps are now upsampled directly and followed by a number of dense layers in both U-Net and DeepMedic. [21, 25, 26]

4.2 U-Net and V-Net

DeconvNet, a completely symmetric architecture with deconvolution and unpooling layers, was proposed in 2015. It was used to compare the performance of integrating with Exponential Linear Unit (ELU) and Maxout in order to solve the "dying ReLU" problem when segmenting the cervical muscle using ultrasound. [27, 28, 29]

In the same year, Olaf Ronneberger, Philipp Fischer, and Thomas Brox proposed U-net with cropped feature maps concatenated from encoder to decoder for localizing each pixel and increased channel number in decoder for propagation of context feature in higher resolution layers. Ran Zhou, Aaron Fenster, Yujiao Xia, and others proposed a dynamic convolution neural network for media-adventitia and lumen-intima segmentation in ultrasound, by introducing short connections into the encoder part of the U-net, to prevent overfitting and dynamically fine tune in specific test task. Slice-by-slice nipple segmentation and localization on breast ultrasound was proposed by Zhemin Zhuang and his team as Grouped-Resaunet (GRA U-net). An architecture called GRA U-net is based on the U-net and uses attention gates to concentrate on pertinent input areas, group convolution for computational efficiency, and residual blocks to solve vanishing gradient problems. For the segmentation of coronary arteries in intravascular ultrasound images, Sekeun Kim, Yeonggul Jang, Byunghwan Jeon, Youngtaek Hong, Hackjoon Shim, and Hyukjae Chang proposed a network with the multi-scale input and hybrid multi-label loss function. On the basis of the U-net network's structure and using a dataset of nerve ultrasound images, Neural Architecture Search (NAS) was applied to a semantic segmentation network. [27, 30, 31]

U-net was first applied to 3D architecture for volumetric segmentation with 3D convolution following its success in 2D medical image segmentation. The same year, V-net integration with 3D convolution and residual blocks was proposed for volumetric segmentation. Then, a multi-directional deeply supervised V-net was presented, and it was used for ultrasound prostate segmentation. The network uses stage-wise hybrid loss function to shorten convergence time, predicts different resolution segmentation from each decoder part stage, and fuses segmentation using multi-directional contour refinement processing. [27, 32]

4.3 Data Augmentation

Large-scale datasets, which are often scarce in medical segmentation tasks, are highly sought-after for the efficient use of deep learning techniques. In cases of insufficiently large dataset, the existing dataset's size can be effectively increased by subjecting the original data to transformations like flipping, rotation, translation, and deformation. This is referred to as data augmentation and is frequently used in machine learning. By adding random variations to the original data, data augmentation increases the size of training examples and decreases overfitting. The data augmentation has been cited as being extremely helpful in numerous studies. [33]

Chapter 5

Methodology

5.1 Tech Stack

Code is written in Python programming language. Besides Python, MatLab was considered as an option for the training, however, after careful consideration, Python was chosen as a better solution for several reasons:

Python is open source MatLab is paid software package, which can be used for free for academic purposes only (that might also vary depending on the university). For interoperability and usefulness of gained knowledge and skills, Python appears to be the better option.

Larger community As one of the most popular programming languages, Python has a larger community than MatLab, especially concerning machine learning. As a result, Python generally offers more features than MatLab.

Better documentation For its larger developer community Python offers generally better, more extensive documentation.

Despite the fact Python was chosen for the purpose of training, there are some advantages to using MatLab that should be mentioned:

Setup Since MatLab is paid, ready-to-use software, one can be sure everything will work as expected out of the box once the new feature is added to the MatLabs's functionality. On the other hand, while setting up the environment in Python, one can often encounter conflicts with different versions of various libraries, there may be issues with different Python interpreters etc. None of that is expected to happen using MatLab.

Graphical Interface For the purpose of designing the network's architecture, a graphical interface is available, which is, for some developers, more comfortable.

5.1.1 Python frameworks

There are two principal frameworks in Python for the purpose of deep learning: Tensorflow from Google and PyTorch from Meta.

PyTorch is one of the latest deep learning frameworks and was developed by the team at Facebook and open sourced on GitHub in 2017. PyTorch is gaining popularity for its simplicity, ease of use, dynamic computational graph and efficient memory usage. [34]

PyTorch Advantages

- Python-like coding
- Dynamic graph
- Easy and quick editing
- Good documentation and community support
- Open source
- Plenty of projects out there using PyTorch

PyTorch Disadvantages

- Third-party needed for visualization
- API server needed for production

TensorFlow is an open-source deep learning framework created by developers at Google and released in 2015. The official research is published in the paper "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems." TensorFlow is now widely used by companies, startups, and business firms to automate things and develop new systems. It draws its reputation from its distributed training support, scalable production and deployment options, and support for various devices like Android. [34]

Tensorflow Advantages

- Simple built-in high-level API
- Visualizing training with Tensorboard
- Production-ready thanks to TensorFlow serving
- Easy mobile support
- Open source
- Good documentation and community support

Tensorflow Disadvantages

- Static graph
- Debugging method
- Hard to make quick changes

Despite the fact Tensorflow offers in-built visualization tools, it remains to be an irrelevant advantage since I prefer to use Matplotlib library for all the visualizations in my projects (including the objective of this thesis).

When starting with the field of deep learning, I decided to learn to work in Tensorflow because it offers better visualization tools (I thought it would be much more important advantage, however, as described above, my preference became using Matplotlib), has much more pleasant documentation and it is easier to deploy the trained models with it. Because of the skills I have already had in Tensorflow, I decided to use it for the objective of this thesis. There is also a library called Keras, a subset of the Tensorflow framework, which makes working with the framework's features easier and was used for the objective of this thesis together with the Tensorflow framework.

5.1.2 Keras

Keras is an open-source high-level Neural Network library, which is written in Python is capable enough to run on Theano, TensorFlow, or CNTK. It was developed by one of the Google engineers, Francois Chollet. To enable quicker experimentation with deep neural networks, it is made user-friendly, extensible, and modular. It supports both Convolutional and Recurrent Networks separately as well as in combination. It uses the Backend library to resolve low-level computations because it is unable to handle them. As a high-level API wrapper for the low-level API, the backend library enables it to run on TensorFlow, CNTK, or Theano. [35]

By providing high-level building blocks, Keras, a model-level library, aids in the creation of deep learning models. Instead of being handled by Keras itself, all of the low-level computations, including convolutions and products of tensors, rely on a specialized tensor manipulation library that has been carefully optimized to act as a backend engine. As a result, Keras offers the ability to plug in various backend engines, rather than incorporating a single tensor library and carrying out operations specific to that library. [35]

5.1.3 NumPy

For the dataset preprocessing, NumPy library was used extensively, espically for array operations. NumPy is the fundamental package for scientific computing in Python. A multidimensional array object, various derived objects (like masked arrays and matrices), and a variety of routines for quick operations on arrays are provided by this Python library. These operations include discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation, and much more. The ndarray object is the nucleus of the NumPy package. This contains homogeneous n-dimensional arrays of data types, with many operations carried out in compiled code for speed. [36]

5.1.4 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being an Apache 2 licensed product, OpenCV makes it easy for businesses to utilize and modify the code. [37]

More than 2500 optimized algorithms are available in the library, including a wide range of both traditional and cutting-edge computer vision and machine learning algorithms. These algorithms can be used to find similar images from an image database, remove red eyes from flash-taken photos, follow eye movements, recognize scenery, and establish markers to overlay. They can also be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce high-resolution images of entire scenes, extract 3D models of objects from stereo cameras, and extract 3D models of objects Windows, Linux, Android, and Mac OS are all supported by OpenCV, which has interfaces in C++, Python, Java, and MATLAB. [37]

OpenCV was used for loading the images and saving output of the trained models. In the early stages of the project, OpenCV was used extensively for the code prototyping, however, later, OpenCV was replaced by Matplotlib library, scikit-learn library and in-built Tensorflow functionalities since they were more comfortable to work with. In summary, Matplotlib was used for data visualization, for array manipulation that included reshaping and normalazing the array, mainly the NumPy library was used. Personally, for color manipulation however, which was also intermittently used (especially for experimentig with various methods), OpenCV remains irreplaceable.

5.1.5 scikit-learn

scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is a NumFOCUS fiscally sponsored project. [38, 39]

scikit-learn is largely written in Python, and uses NumPy extensively for high-performance linear algebra and array operations. scikit-learn integrates well with many other Python libraries, such as Matplotlib and Plotly for plotting, NumPy for array vectorization, Pandas dataframes, SciPy, and many more.

scikit-learn library was used for splitting the data into training and testing subsets and for estimating a class weight.

5.1.6 Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. [40]

Pyplot is a Matplotlib module that provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python, and the advantage of being free and open-source. This module was used for data visualization, e.g. plotting the training accuracy and loss, viewing the segmentations results etc.

5.1.7 Editors and Training

For the code prototyping and data visualisation, Jupyter Notebook was used. Jupyter Notebook (formerly known as IPython notebook) is an interactive web application for creating and sharing computational documents. The project was first named IPython and later renamed Jupyter in 2014. It is a fully open-source product, and users can use every functionality available for free. It supports more than 40 languages including Python, R, and Scala. [41]

After the code was written and debugged in Jupyter Notebook locally, Python script was generated based on the notebooks and executed on the Metacentrum's infrastracuture. MetaCentrum Virtual Organization (MetaVO) operates and manages distributed computing infrastructure consisting of computing and storage resources owned by CESNET as well as those of co-operative academic centers within the Czech Republic. [42]

5.2 Dataset

For the purpose of the training, a dataset of 870 PNG images of atherosclerotic plaque and their corresponding labels was provided. Dimensions of both images and their labels are `544 x 544` pixels. Example of the image and the corresponding label are shown in the figure below:

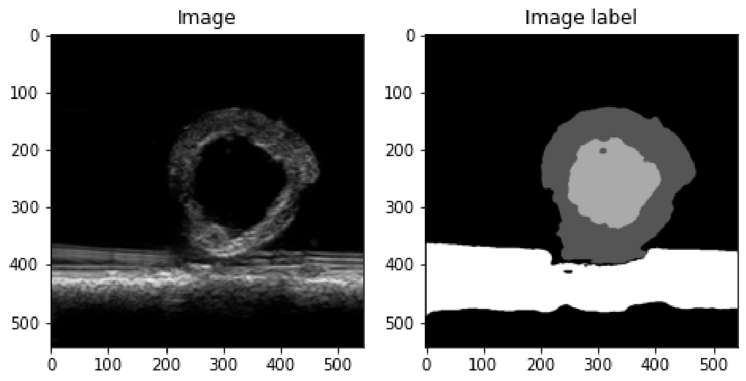


Figure 5.1: Dataset Example

As shown in the label, there are four classes to be segmented:

1. Background - Black
2. Plaque - Dark Gray
3. Lumen - Light Gray
4. Artifacts - White

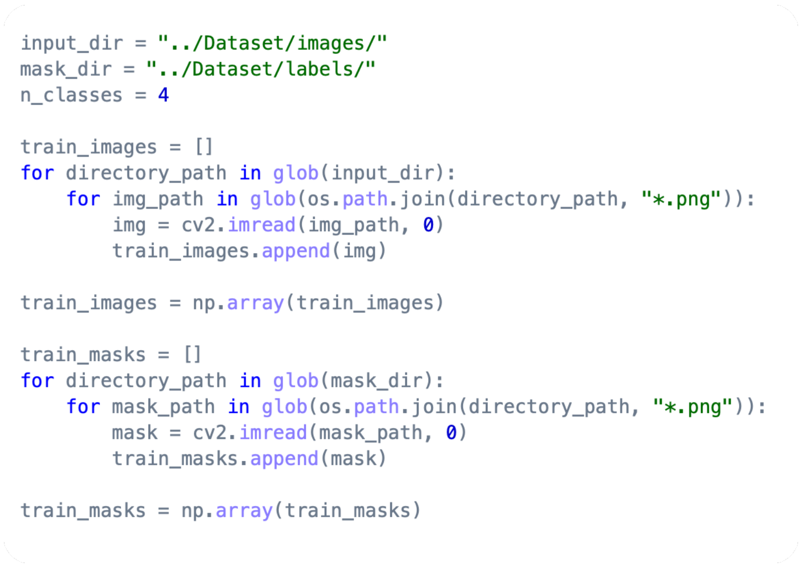
As shown in the image, slices of plaques are captured on ultrasound. Because the dataset comes from a collection of histological sections, part of a slide tray is captured together with the artery as well. Therefore, the neural network needs to learn to differentiate between the body of the artery and artifacts such as the slide tray.

5.3 Data Preprocessing

Before the training itself, images and labels were loaded into two separate arrays (`train_images` and `train_labels`) by iterating over their respective directories. Each image was loaded as an `544 x 544` array with the OpenCV library. Loaded image was then appended to `train_images` or `train_labels` array depending, whether it was the image itself or the label.

After the two series of iterations, both arrays were converted into NumPy arrays. The reason for conversion is that while a normal array is a basic data structure that stores a collection of elements of the same data type in contiguous memory locations, NumPy arrays are a powerful extension of the normal array that are optimized for numerical computations and operations, memory usage and efficiency.

They store data in contiguous blocks of memory, which means that they can be accessed and manipulated much more quickly than normal arrays. Also, a growing plethora of scientific and mathematical Python-based packages are using NumPy arrays. Though they typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. Since several libraries and functions that were used for data preprocessing work with NumPy arrays, they were converted in advance. [36]



```
input_dir = "../Dataset/images/"
mask_dir = "../Dataset/labels/"
n_classes = 4

train_images = []
for directory_path in glob(input_dir):
    for img_path in glob(os.path.join(directory_path, "*.png")):
        img = cv2.imread(img_path, 0)
        train_images.append(img)

train_images = np.array(train_images)

train_masks = []
for directory_path in glob(mask_dir):
    for mask_path in glob(os.path.join(directory_path, "*.png")):
        mask = cv2.imread(mask_path, 0)
        train_masks.append(mask)

train_masks = np.array(train_masks)
```

Figure 5.2: Loading Dataset

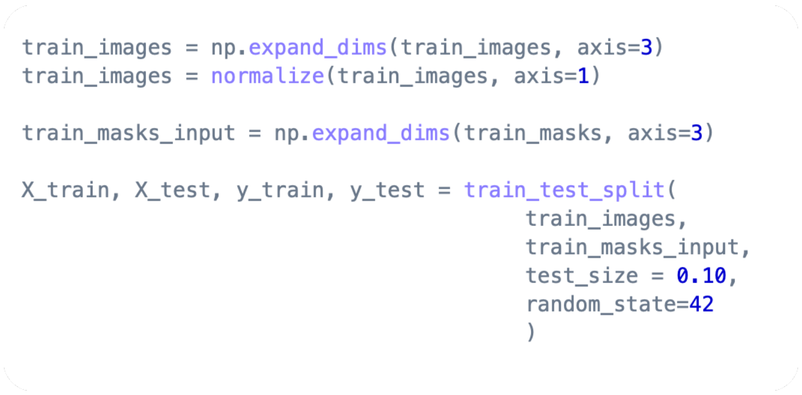
After the dataset was loaded, dimensions of both arrays were expanded. Original shape of the arrays was `(869, 544, 544)` - every image is `544 x 544` pixels and the array contains 870 images. After the expansion it became `(869, 544, 544, 1)`, so every individual pixel was represented by an array with a single value. It was done because the Tensorflow models are designed to take arrays as an input so in order to segment every single pixel, they have to be represented by an array.

Original array	Expanded array
`[[[0, 0, 0, ..., 0]]]`	`[[[[0], [0], [0], ..., [0]]]]`

scikit-learn package includes a method called `train_test_split`, which splits the dataset into training and testing subsets. This method was used to split `train_images` and `train_labels` arrays into `X_train`, `X_test`, `y_train` and `y_test` arrays, where `X` represents training images and `y` represents labels. As shown in the figure below, `test_size` argument is set to 0.10, which means 10% of the original dataset is used for validation.

Also, as shown in the figure 4.3, `train_images` array was normalized before the dataset split. Tensorflow `normalize` is the method available in the Tensorflow library that helps to bring out the normalization process for tensors in neural networks. The main purpose of this process is to bring the transformation so that all the features work on the same or similar level of scale. Normalization plays a vital role in boosting the training stability as well as the performance of the model.

The method will shift and scale inputs into a distribution centered around 0 with standard deviation 1. It accomplishes this by precomputing the mean and variance of the data, and calling `(input - mean) / sqrt(var)` at runtime. Before the normalization, the mean value of pixels was 18.65, after the normalization, the mean value of pixels became 0.017. [43, 44]



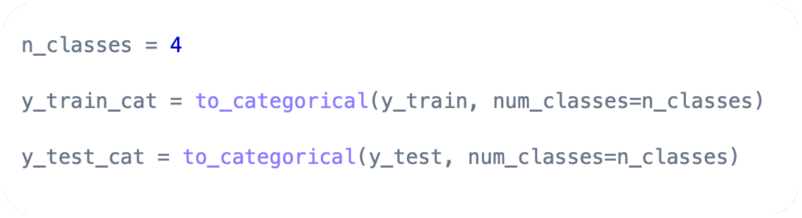
```
train_images = np.expand_dims(train_images, axis=3)
train_images = normalize(train_images, axis=1)

train_masks_input = np.expand_dims(train_masks, axis=3)

X_train, X_test, y_train, y_test = train_test_split(
                                        train_images,
                                        train_masks_input,
                                        test_size = 0.10,
                                        random_state=42
                                        )
```

Figure 5.3: Expanding dimensions and dataset split

The next step was to convert image labels to "categorical" format by using `to_categorical` function. The `to_categorical` function is a utility function provided by the Keras API in TensorFlow, which is used for converting a class vector (integer labels) into a binary class matrix. This function is typically used in multi-class classification problems where the target variable contains categorical data that needs to be converted to a one-hot encoded format. Converting integer labels to categorical arrays or one-hot encoded vectors is often necessary for deep learning models built using TensorFlow, as many deep learning models require the target variable to be represented in this way.



```
n_classes = 4

y_train_cat = to_categorical(y_train, num_classes=n_classes)

y_test_cat = to_categorical(y_test, num_classes=n_classes)
```

Figure 5.4: Converting label to "categorical" format

As shown in the figure above, `n_classes` variable is set to 4 (see figure 4.1 showing 4 classes in the label). The `n_classes` variable together with the label array are passed as arguments to the `to_categorical` function. Before the conversion, the labels contained 4 individual pixel values - 0, 1, 2 and 3, each representing different class to be segmented. In the table below are shown new values of each pixel:

Original value	New value
0	[1., 0., 0., 0.]
1	[0., 1., 0., 0.]
2	[0., 0., 1., 0.]
3	[0., 0., 0., 1.]

Converting data to categorical format was the last step of the data preprocessing and data could be finally used for model fitting.

5.4 Network Architecture

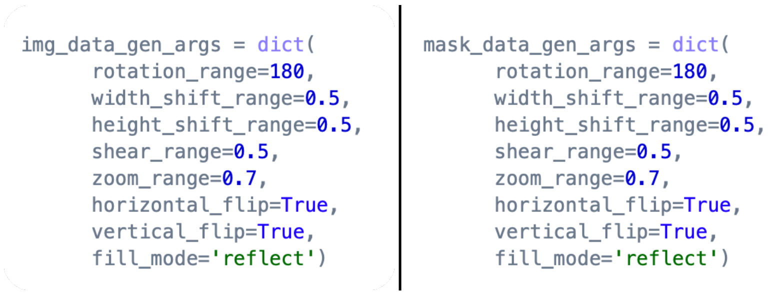
Based on the research part of this thesis (Chapter 4 Current State) and sheer amount of articles about U-net segmentation, I decided to start experimenting with U-net architecture.

First architecture I designed was U-net with 5 bottleneck contraction blocks and 4 expansive residual blocks. After that, the U-net with 6 bottleneck contraction blocks and 5 expansive residual blocks was designed to be able to better extract features from the images. Variations of these two architectures together with different training parameters were used throughout the course of working on the objectives. Because these two architectures yielded very satisfactory results and with each training there was usually a visible progress, I continued to refine these two architectures. Because of the limited time and some complications during the training in Metacenter, I have unfortunately not managed to develop and test other viable options, such as V-net or VGG.

The designed U-nets schemes are shown in the Appendix A.2.

5.5 Data Augmentation

In order to yield better results, data augmentation was used during the training. Data were generated dynamically during the training. For this purpose, `ImageDataGenerator` function from the Tensorflow library was used. Before each training, arguments for the data augmentation were specified[1] as shown in the figure below:



```
img_data_gen_args = dict(
      rotation_range=180,
      width_shift_range=0.5,
      height_shift_range=0.5,
      shear_range=0.5,
      zoom_range=0.7,
      horizontal_flip=True,
      vertical_flip=True,
      fill_mode='reflect')
```

```
mask_data_gen_args = dict(
      rotation_range=180,
      width_shift_range=0.5,
      height_shift_range=0.5,
      shear_range=0.5,
      zoom_range=0.7,
      horizontal_flip=True,
      vertical_flip=True,
      fill_mode='reflect')
```

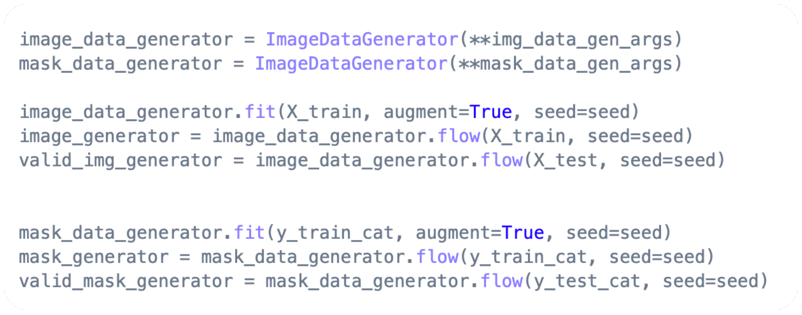
Figure 5.5: Example of data augmentation arguments

`fill_mode` is set to "reflect" because otherwise the function may during image transformations fill in random pixels, which of course makes a successful training impossible.

Next, `image_data_generator` and `mask_data_generator` were initialized with `ImageDataGenerator` function, to which arguments for the augmentation were passed. The `image_data_generator` and `mask_data_generator` where then fit on the data

[1]Examples of how some augmentations look like is available in the Appendix A.5

from the provided dataset. `ImageDataGenerator`'s `flow` method then creates a data generator, which generates batches of augmented data, ready-to-use for the training.



```
image_data_generator = ImageDataGenerator(**img_data_gen_args)
mask_data_generator = ImageDataGenerator(**mask_data_gen_args)

image_data_generator.fit(X_train, augment=True, seed=seed)
image_generator = image_data_generator.flow(X_train, seed=seed)
valid_img_generator = image_data_generator.flow(X_test, seed=seed)


mask_data_generator.fit(y_train_cat, augment=True, seed=seed)
mask_generator = mask_data_generator.flow(y_train_cat, seed=seed)
valid_mask_generator = mask_data_generator.flow(y_test_cat, seed=seed)
```

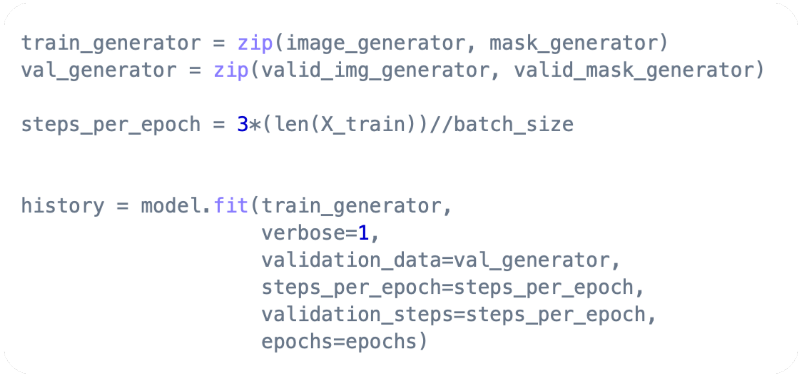
Figure 5.6: Data generators fitting

The figure shows 4 data generators were created:

1. `image_generator` - generates training images
2. `valid_img_generator` - generates validation images
3. `mask_generator` - generates training labels
4. `valid_mask_generator` - generates validation labels

Also, it is crucial to use the same seed as an argument to `fit` and `flow` functions so the same transformations are applied to both images and labels. Otherwise, the augmentation will produce incompatible data and the training will fail.

The last step is to "zip" the image and labels generators into `train_generator` and `val_generator`, and the generators are ready to be used in the training. The `zip()` method returns a zip object, which is an iterator of tuples where the first and second items in each provided iterator are coupled together etc. [45]



```
train_generator = zip(image_generator, mask_generator)
val_generator = zip(valid_img_generator, valid_mask_generator)

steps_per_epoch = 3*(len(X_train))//batch_size


history = model.fit(train_generator,
                    verbose=1,
                    validation_data=val_generator,
                    steps_per_epoch=steps_per_epoch,
                    validation_steps=steps_per_epoch,
                    epochs=epochs)
```

Figure 5.7: Zipping the image and labels generators

5.6 Testing

Methods used for testing were Pixel Accuracy, Intersection Over Union and Mean Intersection Over Union.

Pixel Accuracy Pixel accuracy is a ratio between the number of correctly classified pixels and the total number of pixels. This metric can be often very misleading, especially if the image classes are not balanced (e.g. most of the image is background). It such cases it is possible to reach accuracy of 95 % even when the model is not able to segment a minority class at all. Therefore, using IOU and Mean IOU is far more descriptive option. [27]

$$PA = \frac{\sum_{i=0}^{k} P_{ii}}{\sum_{i=0}^{k} \sum_{j=0}^{k} P_{ij}}$$

Intersection Over Union Intersection over union, also known as Jaccard index, is the percent overlap between the target mask and the prediction output. [27]

$$IOU = \frac{\sum_{i=0}^{k} P_{ii}}{\sum_{i=0}^{k} \sum_{j=0}^{k} P_{ij} - \sum_{j=0}^{k} P_{jj}}$$

Mean Intersection Over Union Pixel accuracy is a ratio between the number of correctly classified pixels and the total number of pixels. [27]

$$MIOU = \frac{1}{k+1} \frac{P_{ii}}{\sum_{j=0}^{k} P_{ij} + \sum_{j=0}^{k} P_{ji} - P_{ii}}$$

Chapter 6

Results

In the course of working on this thesis, altogether 12 U-net models were trained. Overview of them is shown in the table below:

Model	Num. of blocks	Epochs	Data Augmentation	Pretrained
20221030	9	50	No	-
20221031	11	50	No	-
20230325	9	20	Yes	-
20230326	11	20	Yes	-
20230329	9	20	Yes	-
20230330	9	30	Yes	-
20230331	9	50	Yes	-
20230421	9	15	Yes	20221030
20230422	9	30	Yes	20221030
20230423	9	100	Yes	20221030
20230501	11	30	Yes	20230326
20230505	11	100	Yes	20230326

Table 6.1: Trained models overview

As shown in the table above, I was testing two U-net architectures - one with 9 blocks, and other with 11. Depending on the results of each training, training parameters - number of epochs, number of layers and augmentation parameters - were adjusted for the next training. In the training phase, I was making assumptions about the parameters for the next training mainly by looking at the results of the specific cases of segmentation (shown in the Appendix A.4 First Glance Overview). Except for the first two models (20221030 and 20221031), all models were trained on augmented data. "Pretrained" column shows, whether (and if so, which one) a pretrained model was used for the training (fine-tuning).

6.1 Testing dataset with 10 degrees rotation range

When looking at the tables like the table below, columns Mean, Background, Plaque, Lumen and Artifacts show IOU values.

Model	Accuracy	Mean	Backg.	Plaque	Lumen	Artifacts
20221031	0.96814	0.86494	0.96469	0.87944	0.73492	0.88073
20221031	0.96827	0.86327	0.9659	0.87862	0.72695	0.88162
20230325	0.97267	0.88133	0.97006	0.8919	0.7633	0.90005
20230326	0.97356	0.88595	0.97111	0.89441	0.77516	0.9031
20230329	0.96526	0.8497	0.96567	0.84807	0.71325	0.87182
20230330	0.96266	0.84365	0.9629	0.83956	0.71214	0.86001
20230331	0.96923	0.86764	0.96673	0.87479	0.74119	0.88787
20230421	0.96551	0.85292	0.96483	0.85282	0.72285	0.87121
20230422	0.96276	0.84328	0.9663	0.81995	0.72884	0.85803
20230423	0.972	0.87734	0.97034	0.88421	0.75703	0.89778
20230501	0.96848	0.86895	0.96631	0.87665	0.75302	0.87982
20230505	0.97251	0.88097	0.97006	0.88972	0.76441	0.89969

Table 6.2: Testing with 10 degree range rotation augmentation

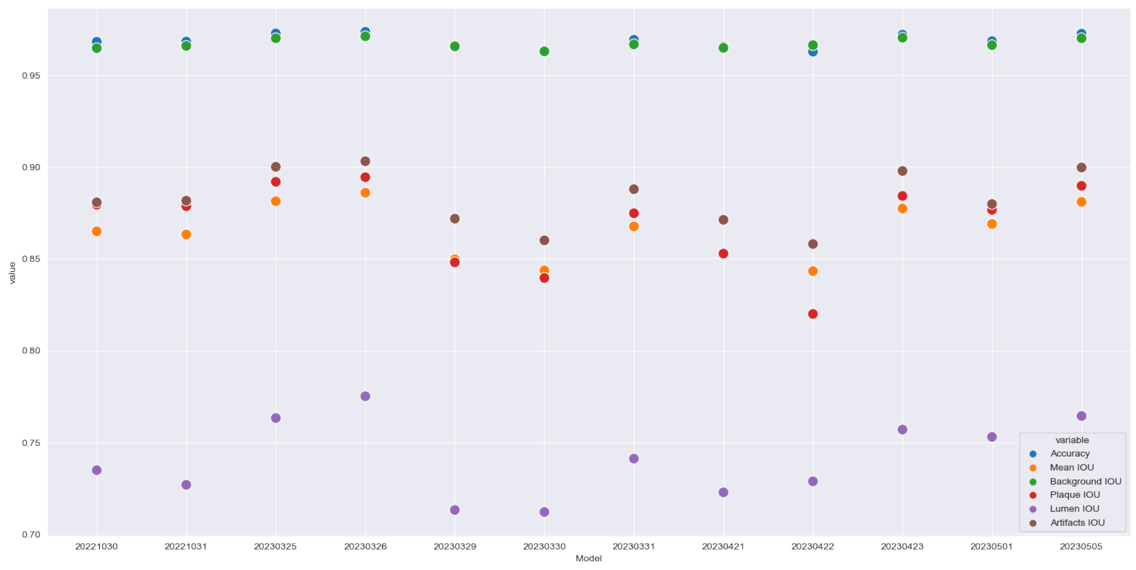


Figure 6.1: Testing on 10 degrees rotation range - accuracy and IOUs

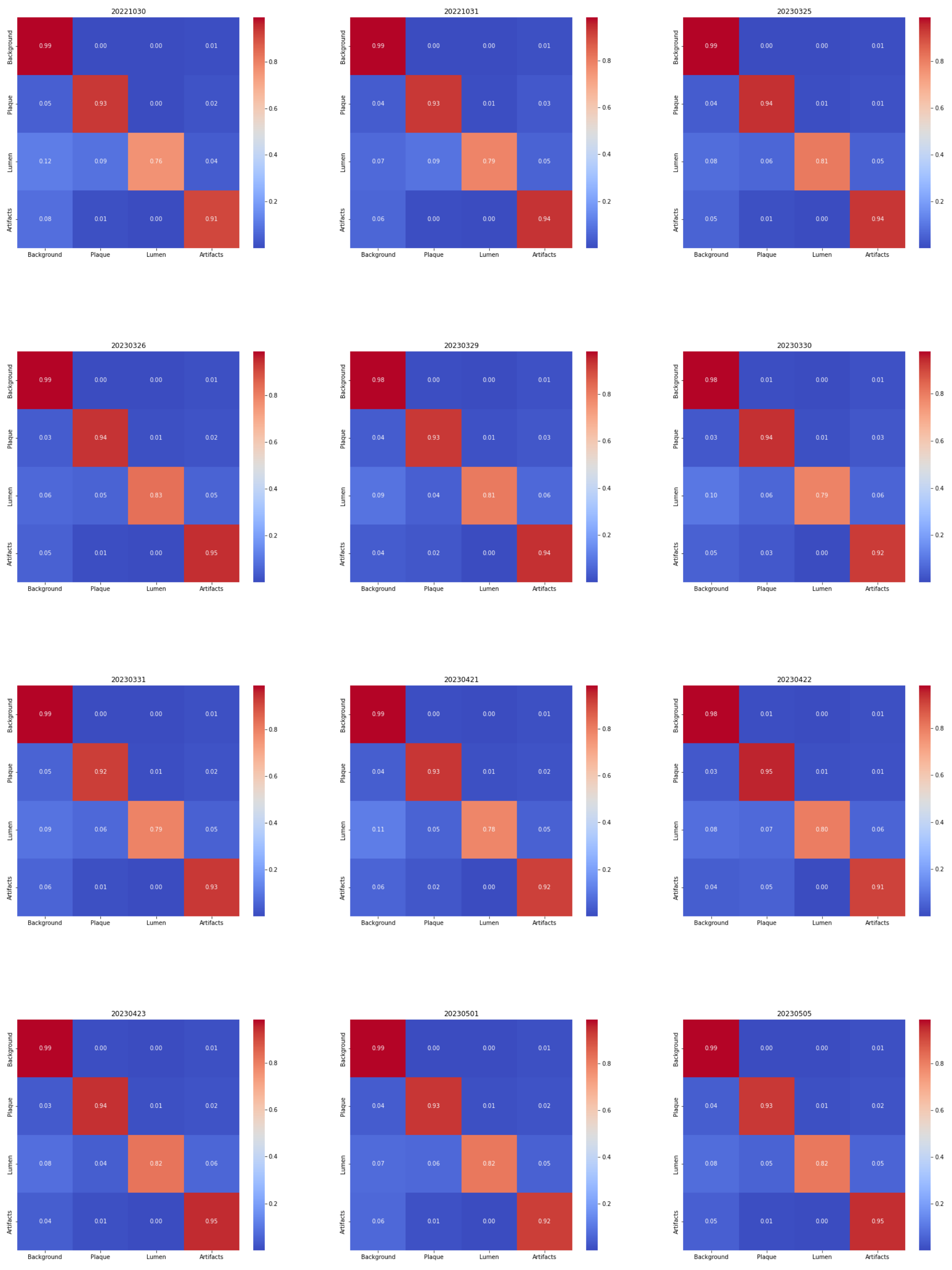


Figure 6.2: Testing on 10 degrees rotation range - confusion matrix

6.2 Testing dataset with 280 degrees rotation range

Model	Accuracy	Mean	Backg.	Plaque	Lumen	Artifacts
20221030	0.8301	0.4925	0.9225	0.3751	0.4179	0.2545
20221031	0.8227	0.4749	0.8999	0.3875	0.2714	0.3408
20230325	0.9129	0.7026	0.9443	0.581	0.6502	0.6351
20230326	0.92	0.7328	0.9485	0.6009	0.7226	0.659
20230329	0.9409	0.7862	0.9434	0.7521	0.6727	0.7768
20230330	0.9378	0.7815	0.94	0.7472	0.6753	0.7633
20230331	0.9512	0.8186	0.9486	0.8097	0.7011	0.8148
20230421	0.9422	0.7902	0.9444	0.756	0.682	0.7783
20230422	0.938	0.7724	0.9485	0.7017	0.6721	0.7673
20230423	0.9552	0.8312	0.9542	0.8224	0.7216	0.8265
20230501	0.9426	0.8023	0.9419	0.7762	0.7114	0.7797
20230505	0.9552	0.833	0.9512	0.8375	0.7148	0.8287

Table 6.3: Testing with 280 degree range rotation augmentation

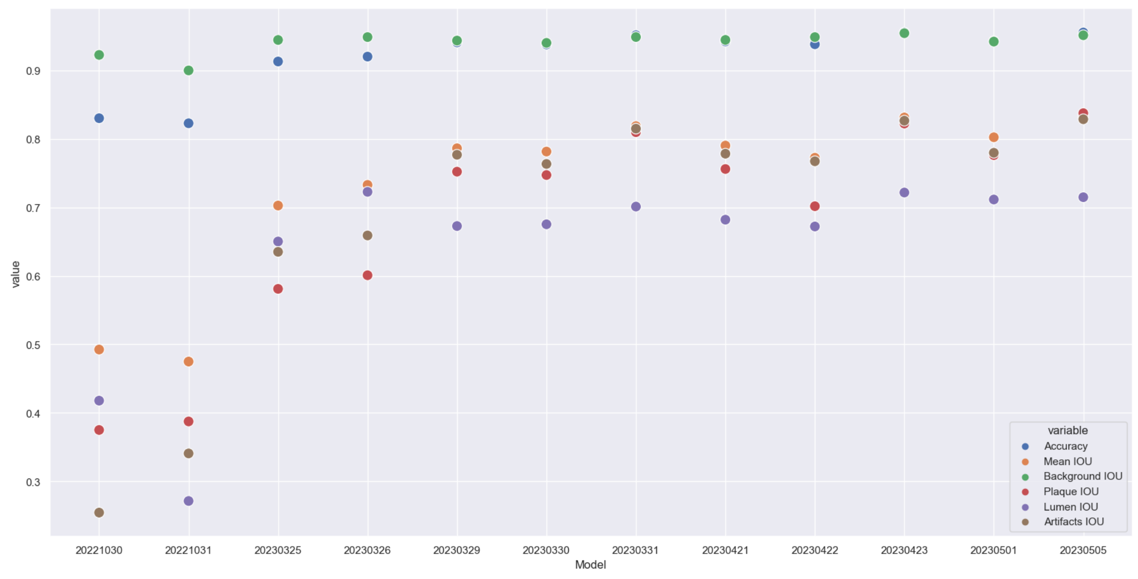


Figure 6.3: Testing on 280 degrees rotation range - accuracy and IOUs

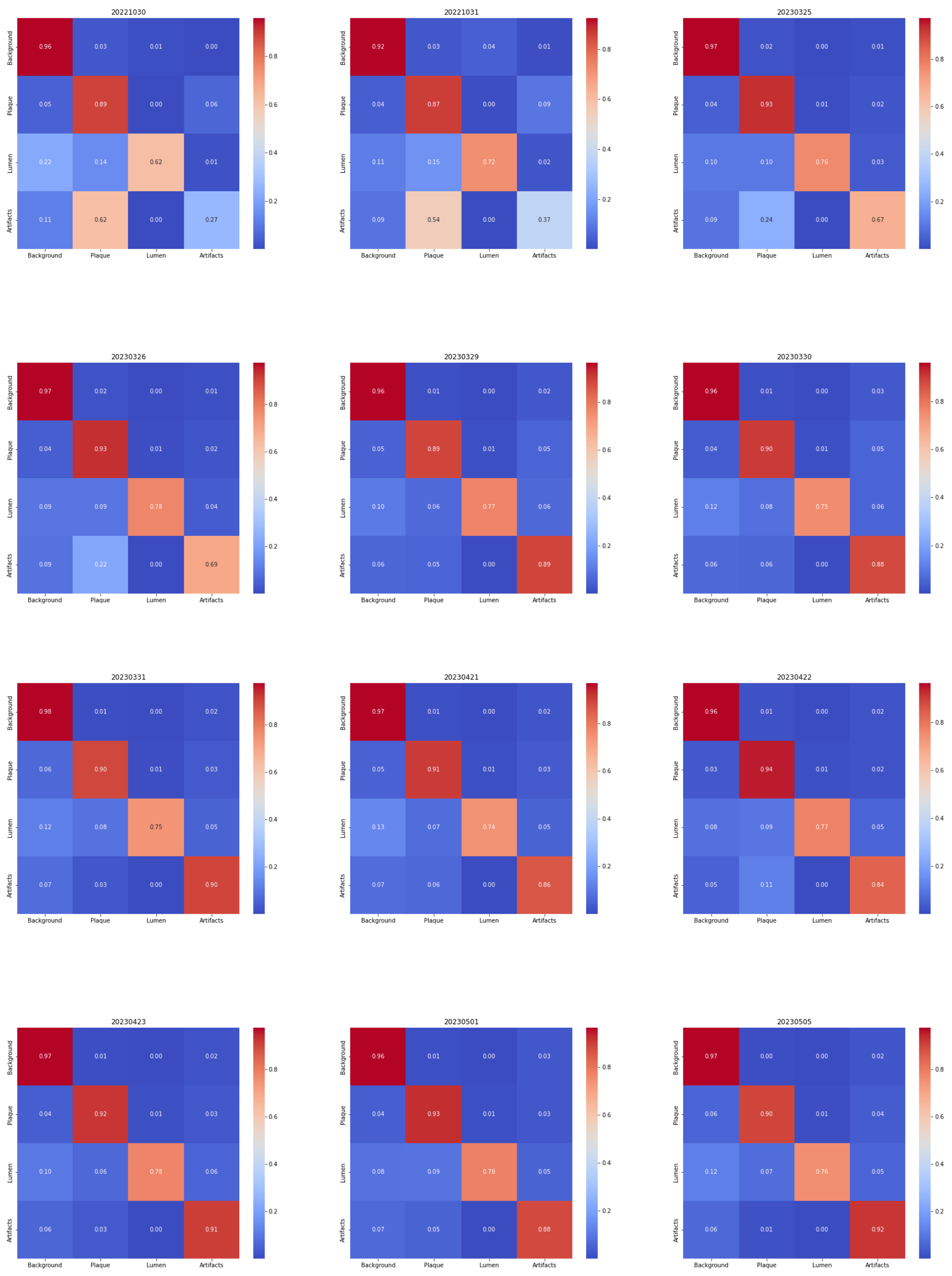


Figure 6.4: Testing on 280 degrees rotation range - confusion matrix

6.3 Testing dataset with vertical flip

Model	Accuracy	Mean	Backg.	Plaque	Lumen	Artifacts
20221030	0.951	0.8099	0.9583	0.758	0.7287	0.7944
20221031	0.9436	0.7642	0.9526	0.7322	0.5812	0.7907
20230325	0.9755	0.9113	0.9715	0.9138	0.8583	0.9016
20230326	0.9761	0.9148	0.9719	0.9155	0.8674	0.9043
20230329	0.9664	0.8733	0.9646	0.8647	0.7969	0.8672
20230330	0.9644	0.8678	0.9625	0.857	0.7926	0.859
20230331	0.9724	0.8984	0.9681	0.9002	0.8339	0.8914
20230421	0.9673	0.8779	0.9647	0.8734	0.805	0.8683
20230422	0.9649	0.8698	0.9659	0.8394	0.8148	0.8592
20230423	0.9746	0.9065	0.9706	0.9077	0.8479	0.9
20230501	0.9715	0.8988	0.9677	0.8979	0.8463	0.8834
20230505	0.9751	0.9106	0.9708	0.9117	0.8591	0.9009

Table 6.4: Testing with vertical flip augmentation

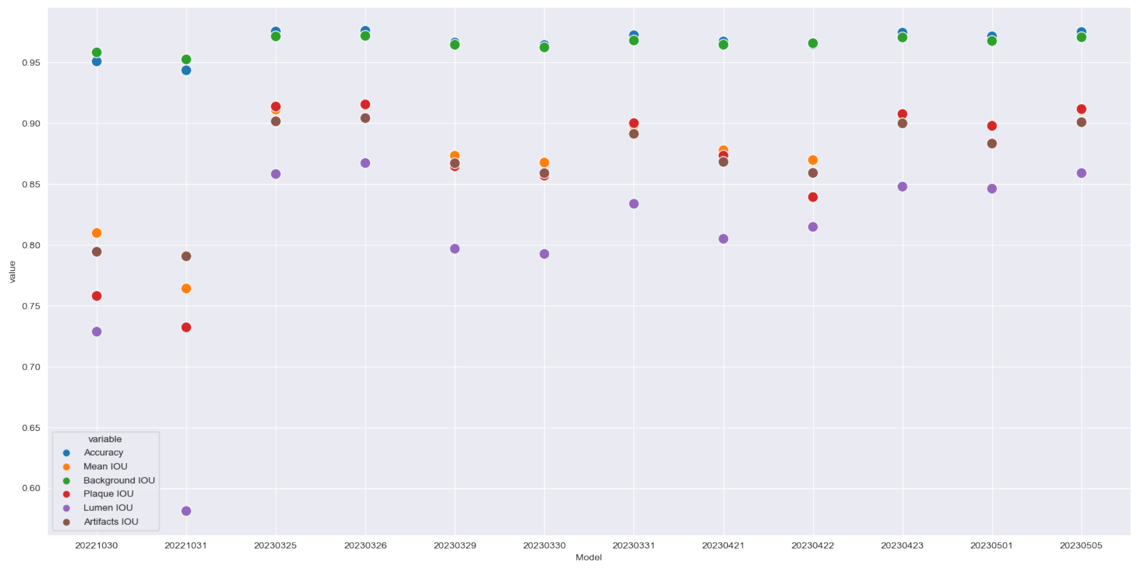


Figure 6.5: Testing on vertical flip - accuracy and IOUs

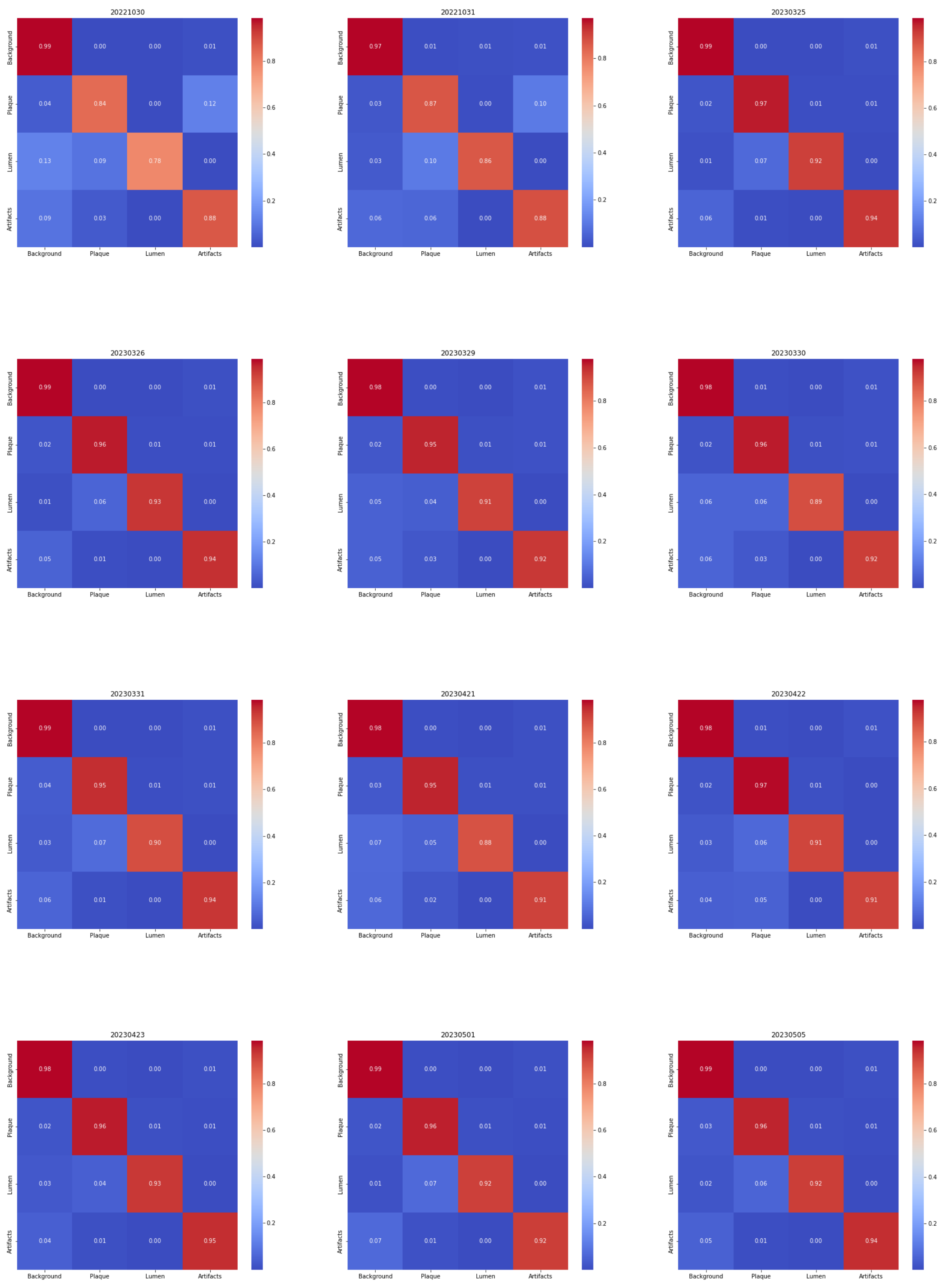


Figure 6.6: Testing on vertical flip - confusion matrix

6.4 Testing dataset with image enlargement

Model	Accuracy	Mean	Backg.	Plaque	Lumen	Artifacts
20221030	0.9223	0.6861	0.9332	0.6845	0.3842	0.7427
20221031	0.9256	0.7144	0.9366	0.7016	0.4708	0.7485
20230325	0.9269	0.7147	0.9381	0.6828	0.4791	0.7591
20230326	0.9316	0.7303	0.9424	0.7025	0.5061	0.7701
20230329	0.9269	0.7336	0.9403	0.6901	0.553	0.7512
20230330	0.9312	0.7414	0.9398	0.7259	0.5329	0.767
20230331	0.9418	0.7762	0.9467	0.7595	0.5962	0.8023
20230421	0.9372	0.7561	0.9422	0.7532	0.539	0.7899
20230422	0.937	0.7687	0.9469	0.7406	0.6068	0.7805
20230423	0.9548	0.8261	0.9542	0.8313	0.671	0.848
20230501	0.9502	0.8172	0.9506	0.8125	0.6806	0.8253
20230505	0.9569	0.8422	0.9566	0.8315	0.7307	0.8497

Table 6.5: Testing with image enlargement augmentation

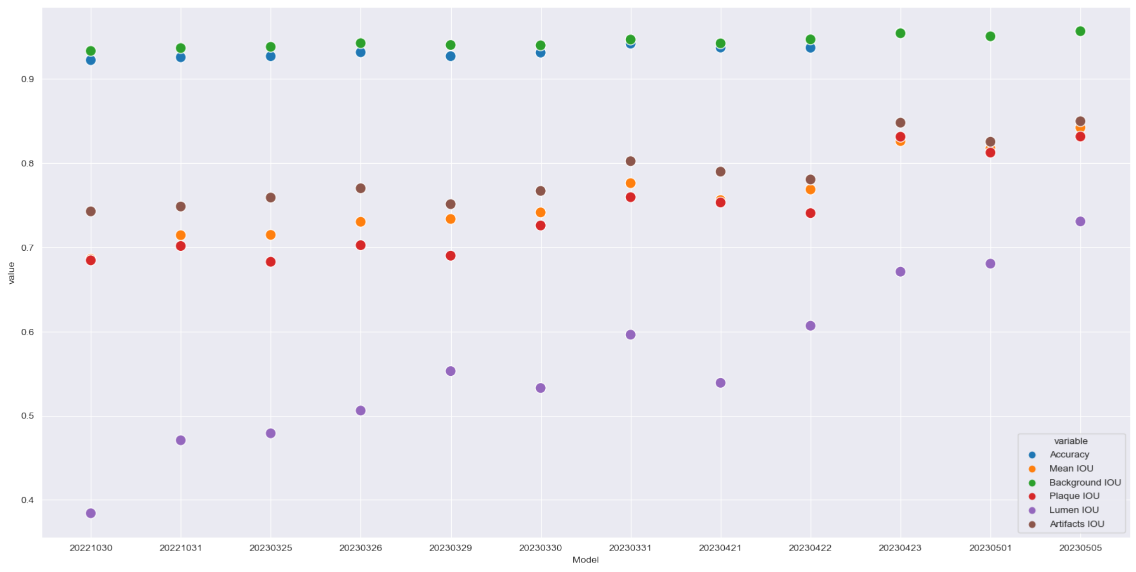


Figure 6.7: Testing on image enlargement - accuracy and IOUs

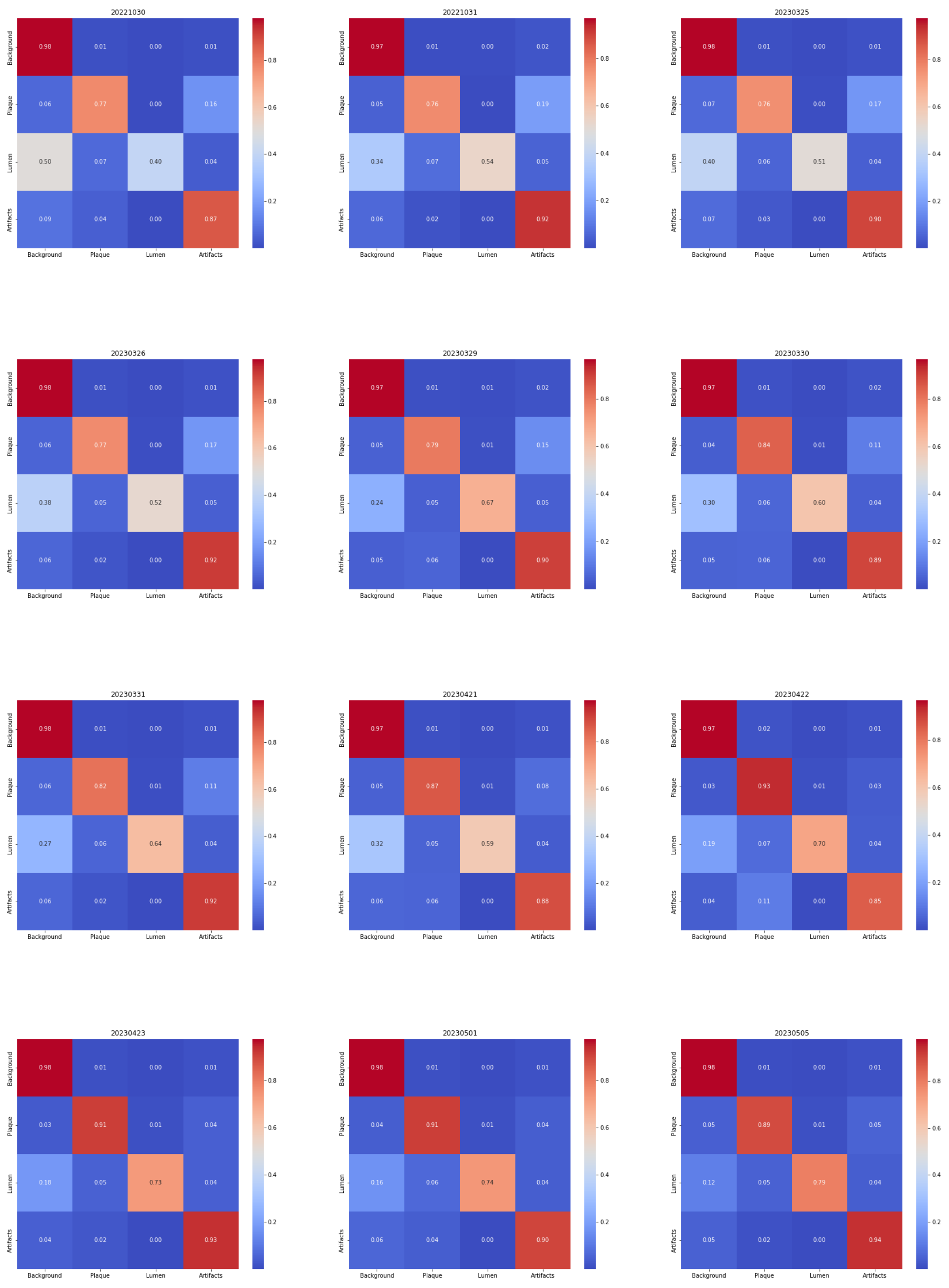


Figure 6.8: Testing on image enlargement - confusion matrix

6.5 Testing dataset with image enlargement and shear mapping

Model	Accuracy	Mean	Backg.	Plaque	Lumen	Artifacts
20221030	0.8912	0.6015	0.9158	0.5825	0.2498	0.6578
20221031	0.8927	0.6241	0.9173	0.5942	0.3225	0.6626
20230325	0.8891	0.6119	0.9167	0.5396	0.3313	0.6599
20230326	0.8927	0.6147	0.9205	0.5534	0.3161	0.6687
20230329	0.8894	0.6351	0.9228	0.5502	0.4282	0.6391
20230330	0.8962	0.6467	0.9233	0.5983	0.4077	0.6575
20230331	0.9033	0.6578	0.9287	0.6016	0.4213	0.6798
20230421	0.9039	0.6602	0.925	0.6303	0.4011	0.6845
20230422	0.9099	0.6944	0.9316	0.6588	0.4952	0.6921
20230423	0.9237	0.7271	0.9384	0.7078	0.5247	0.7376
20230501	0.9184	0.715	0.9331	0.6899	0.514	0.7229
20230505	0.9187	0.7207	0.9392	0.6658	0.5645	0.7133

Table 6.6: Testing dataset with image enlargement and shear mapping

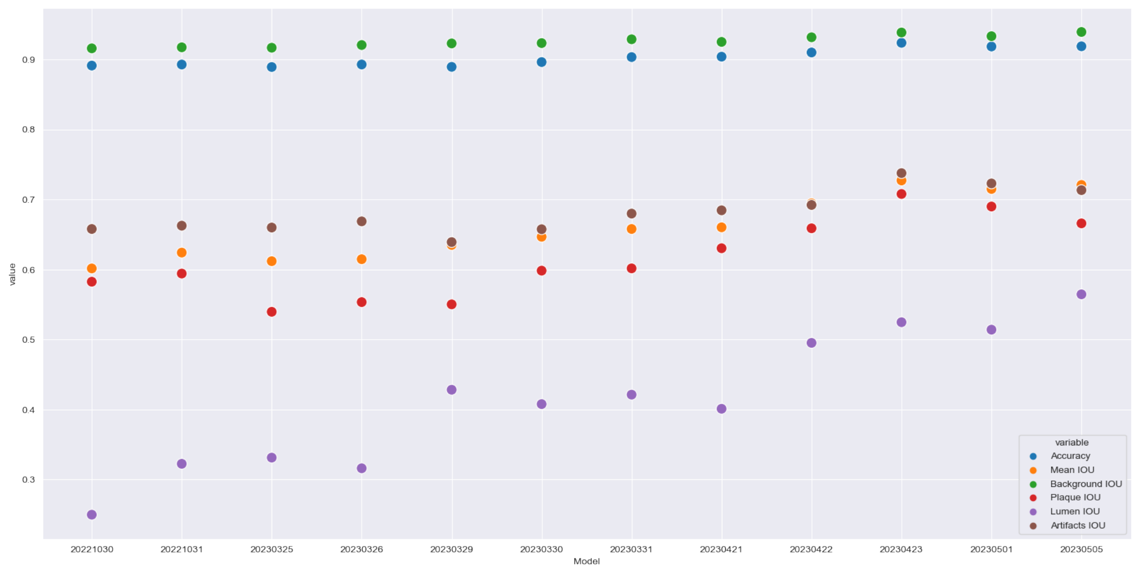


Figure 6.9: Testing on image enlargement and shear mapping - accuracy and IOUs

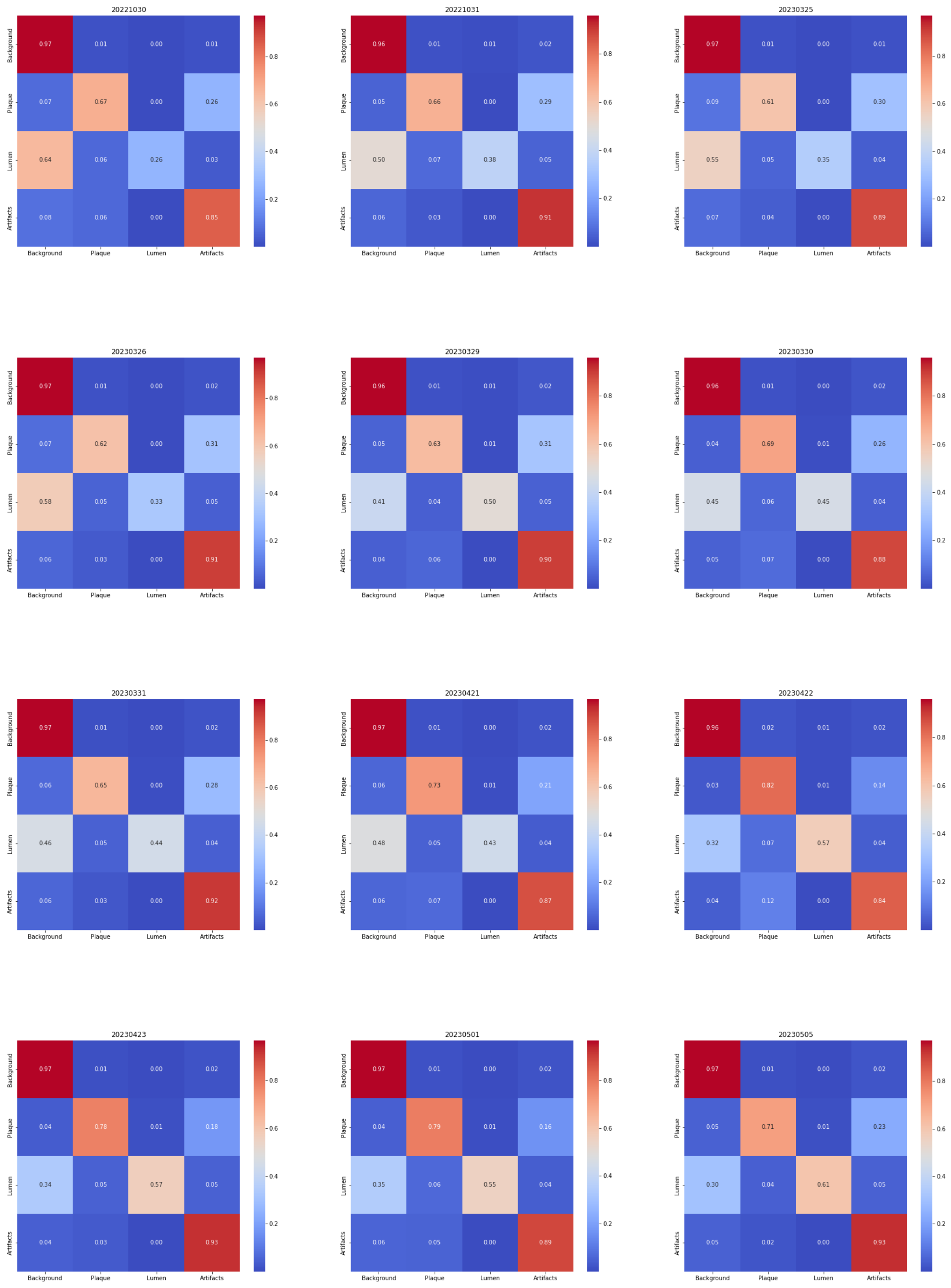


Figure 6.10: Testing on image enlargement and shear mapping - confusion matrix

Chapter 7

Discussion

In this chapter I will discuss each aspect of the assignment and its results.

7.1 Testing Results Discussion

7.1.1 Testing dataset with 10 degrees rotation range

With only 10 degree-range rotation augmentation, the testing dataset was supposed to be closely resembling the original dataset. It is clear from the results that in the case of only minimal changes the majority of the models performed quite satisfyingly, with the exception of the 20230330, 20230421, 20230422, and 20230501 models. Probably, the unsatisfactory results are caused by small amount of training epochs and low "depth" of the architecture.

The two best performing models were the 20230326 and the 20230505 model. The 20230326 model was used as a pretrained model for the 20230505 model training. As shown in the Results, the performance of these models was almost identical, with the 20230326 even performing slightly better than the other. The results indicate additional training will probably have no or only limited effect on the original dataset segmentation.

Since both of the best performing models have 11 blocks, we can assume "deeper" U-net is more suitable for feature extraction.

7.1.2 Testing dataset with 280 degrees rotation range

The next testing case was the dataset with the rotation range of 280 degrees. We can clearly see the models trained without data augmentation failed drastically, with mean IOU being only around 50 %. We can see the 20221031 model with 11 blocks performed slightly worse that the 20221030 model, which contains only 9 blocks. Based on other graphs in the Results and on the example in the Figure A.6, where the model 20221031 tried to find lumen between the artifacts, we can assume training a deeper U-net on the limited dataset leads to overfitting and the model performs poorly on the data that it has not "seen" yet. This proves that in this case, data variability has higher impact on training process than architecture complexity.

7.1.3 Testing dataset with vertical flip

This testing dataset contained vertically flipped images. Just like in the previous example, it is clear training with the augmentation leads to far better results than without it. Also, we can observe the same overfitting of the 20221031 model, with the lumen segmentation accuracy plummeting.

What is also in my opinion interesting is that the models 20230423 and 20230501 generally yield similar results. The difference between them is that the 20230423 model is the 9 block U-net trained with 100 epochs while the 20230501 model is the 11 block U-net trained with 30 epochs.

7.1.4 Testing dataset with image enlargement

Once we start testing with warping transformations, it becomes clear that especially amount of training epochs leads to better results (the two best models were both trained with 100 epochs). The 11 block U-net (20230505), however, reaches slightly better results, with the most significant improvement in the lumen segmentation. This is also the first testing case where the 20221031 model does not worsen in comparison to the 20221030 model, which is personally quite surprising.

I suspect the 11 block U-net's ability to extract features is less affected by image warping then by image rotating because it has learnt to extract features in a specific orientation. Since with this augmentation there are no rotation augmentations, we did not observe such dramatic worsening as we did in the previous cases.

7.2 Strengths and Limitations

Overall, the trained models achieved decent level of accuracy, however, there are aspects in which the models perform better than in others. In the results it is clear the models learnt to differentiate the artifacts in the images quite well. Especially the last model (20230505) is able to recognize where is the border between the plaque and the artifacts decently. This was the issue throughout the course of the trainings and as shown in the examples in the Appendix A.4, often plaques and artifacts "blended together".

As a limitation I consider the ability of the models to segment the lumen. We can see in all graphs in the Results that there is a gap between the lumen IOU and the other classes' IOUs. It seems it is a problematic task for the model to learn, which "hole" should count as lumen and which should not. We can see in the examples that often the model does not count the whole lumen as lumen and some parts of it segments as a background or on the contrary considers most of the openings as lumen and even segments some parts of the artifacts around the openings as plaque. Improper lumen segmentation could also be explained by irregularities in training and testing dataset. In some cases, plaque tissue was ruptured during operation, which leads to images, where lumen is not fully encapsulated by plaque, and in other cases, lumen was not present at all. These irregularities could lead to misinterpretation in lumen segmentation. I assume the solution for this issue is to fine-tune the best performing models on even more data, so the models learns to recognize patterns properly.

7.3 Possible methods for further optimization

One of the interesting methods for further optimization I considered was to apply a Gabor filter on the input data to enhance its features. There was, however, a drawback to this method - the labels for the data would have to be manually adjusted before the training. As shown in the figure below, the Gabor filter highlights details that were not visible before (or were hardly visible), therefore, they are not even marked in the original labels. This method was not implemented due to time limitations, however, I do believe this method would increase the accuracy dramatically if tried.

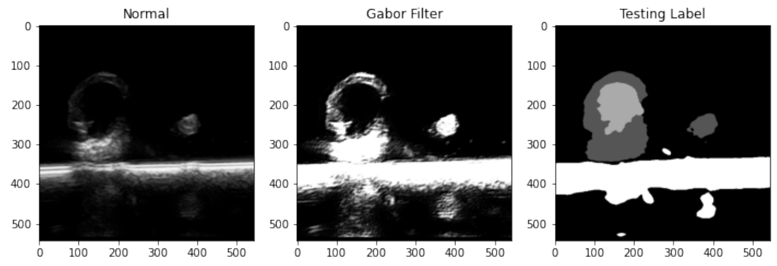


Figure 7.1: Gabor filter application

Chapter 8

Conclusion

In conclusion, this bachelor thesis has investigated the training methods for the CNNs in atherosclerosis segmentation tasks. Through a comprehensive examination of data, CNN architecture, data augmentation and results I have successfully achieved the stated objectives of this thesis. While neural networks proved to be an efficient instrument for medical image segmentation task, problematics of limited dataset proved to have a significant impact on the results, especially in datasets with higher variability. The findings of this thesis have provided me with valuable insights into medical segmentation topic.

Appendix A

Attachments

A.1 Source Code

All the source code for the thesis' objectives, together with the documentation, is available on GitHub. The repository might be subject to changes even after the thesis submission, however, the changes will for the code optimization only. Functionality of the code will not be affected

Source code is available in the link below:

Figure A.1: Source Code Repository

`https://github.com/david-pilny/bachelor-thesis`

A.2 U-net architectures

Schemes of used architectures for the training. The U-net architectures were designed - one with 9 layers and other with 11.

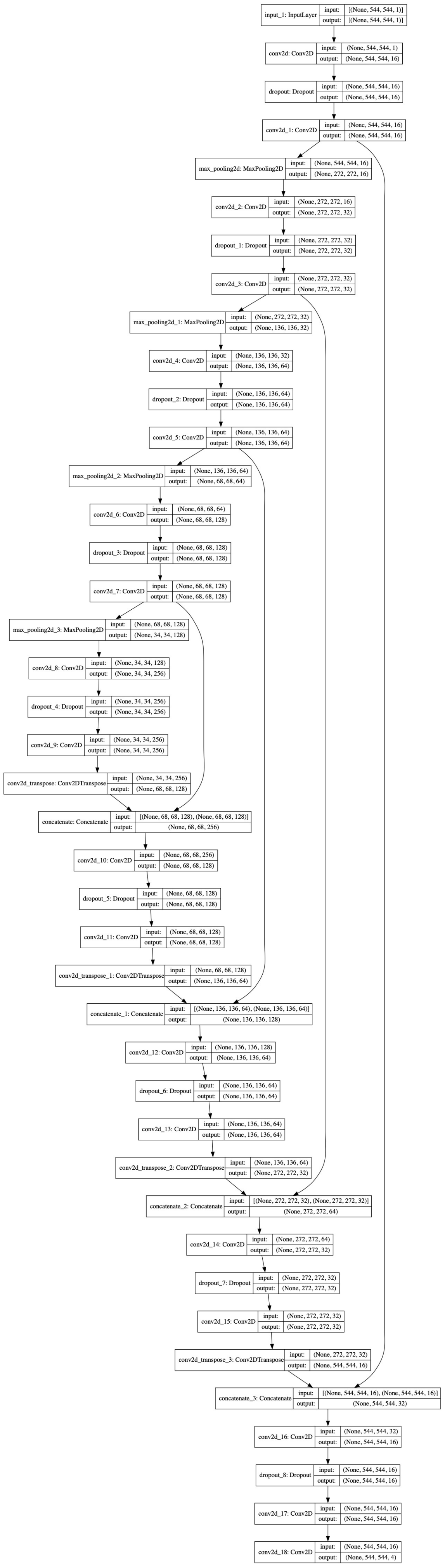

input_1: InputLayer | input: [(None, 544, 544, 1)] | output: [(None, 544, 544, 1)]
conv2d: Conv2D | input: (None, 544, 544, 1) | output: (None, 544, 544, 16)
dropout: Dropout | input: (None, 544, 544, 16) | output: (None, 544, 544, 16)
conv2d_1: Conv2D | input: (None, 544, 544, 16) | output: (None, 544, 544, 16)
max_pooling2d: MaxPooling2D | input: (None, 544, 544, 16) | output: (None, 272, 272, 16)
conv2d_2: Conv2D | input: (None, 272, 272, 16) | output: (None, 272, 272, 32)
dropout_1: Dropout | input: (None, 272, 272, 32) | output: (None, 272, 272, 32)
conv2d_3: Conv2D | input: (None, 272, 272, 32) | output: (None, 272, 272, 32)
max_pooling2d_1: MaxPooling2D | input: (None, 272, 272, 32) | output: (None, 136, 136, 32)
conv2d_4: Conv2D | input: (None, 136, 136, 32) | output: (None, 136, 136, 64)
dropout_2: Dropout | input: (None, 136, 136, 64) | output: (None, 136, 136, 64)
conv2d_5: Conv2D | input: (None, 136, 136, 64) | output: (None, 136, 136, 64)
max_pooling2d_2: MaxPooling2D | input: (None, 136, 136, 64) | output: (None, 68, 68, 64)
conv2d_6: Conv2D | input: (None, 68, 68, 64) | output: (None, 68, 68, 128)
dropout_3: Dropout | input: (None, 68, 68, 128) | output: (None, 68, 68, 128)
conv2d_7: Conv2D | input: (None, 68, 68, 128) | output: (None, 68, 68, 128)
max_pooling2d_3: MaxPooling2D | input: (None, 68, 68, 128) | output: (None, 34, 34, 128)
conv2d_8: Conv2D | input: (None, 34, 34, 128) | output: (None, 34, 34, 256)
dropout_4: Dropout | input: (None, 34, 34, 256) | output: (None, 34, 34, 256)
conv2d_9: Conv2D | input: (None, 34, 34, 256) | output: (None, 34, 34, 256)
conv2d_transpose: Conv2DTranspose | input: (None, 34, 34, 256) | output: (None, 68, 68, 128)
concatenate: Concatenate | input: [(None, 68, 68, 128), (None, 68, 68, 128)] | output: (None, 68, 68, 256)
conv2d_10: Conv2D | input: (None, 68, 68, 256) | output: (None, 68, 68, 128)
dropout_5: Dropout | input: (None, 68, 68, 128) | output: (None, 68, 68, 128)
conv2d_11: Conv2D | input: (None, 68, 68, 128) | output: (None, 68, 68, 128)
conv2d_transpose_1: Conv2DTranspose | input: (None, 68, 68, 128) | output: (None, 136, 136, 64)
concatenate_1: Concatenate | input: [(None, 136, 136, 64), (None, 136, 136, 64)] | output: (None, 136, 136, 128)
conv2d_12: Conv2D | input: (None, 136, 136, 128) | output: (None, 136, 136, 64)
dropout_6: Dropout | input: (None, 136, 136, 64) | output: (None, 136, 136, 64)
conv2d_13: Conv2D | input: (None, 136, 136, 64) | output: (None, 136, 136, 64)
conv2d_transpose_2: Conv2DTranspose | input: (None, 136, 136, 64) | output: (None, 272, 272, 32)
concatenate_2: Concatenate | input: [(None, 272, 272, 32), (None, 272, 272, 32)] | output: (None, 272, 272, 64)
conv2d_14: Conv2D | input: (None, 272, 272, 64) | output: (None, 272, 272, 32)
dropout_7: Dropout | input: (None, 272, 272, 32) | output: (None, 272, 272, 32)
conv2d_15: Conv2D | input: (None, 272, 272, 32) | output: (None, 272, 272, 32)
conv2d_transpose_3: Conv2DTranspose | input: (None, 272, 272, 32) | output: (None, 544, 544, 16)
concatenate_3: Concatenate | input: [(None, 544, 544, 16), (None, 544, 544, 16)] | output: (None, 544, 544, 32)
conv2d_16: Conv2D | input: (None, 544, 544, 32) | output: (None, 544, 544, 16)
dropout_8: Dropout | input: (None, 544, 544, 16) | output: (None, 544, 544, 16)
conv2d_17: Conv2D | input: (None, 544, 544, 16) | output: (None, 544, 544, 16)
conv2d_18: Conv2D | input: (None, 544, 544, 16) | output: (None, 544, 544, 4)

Figure A.2: U-Net with 9 layers

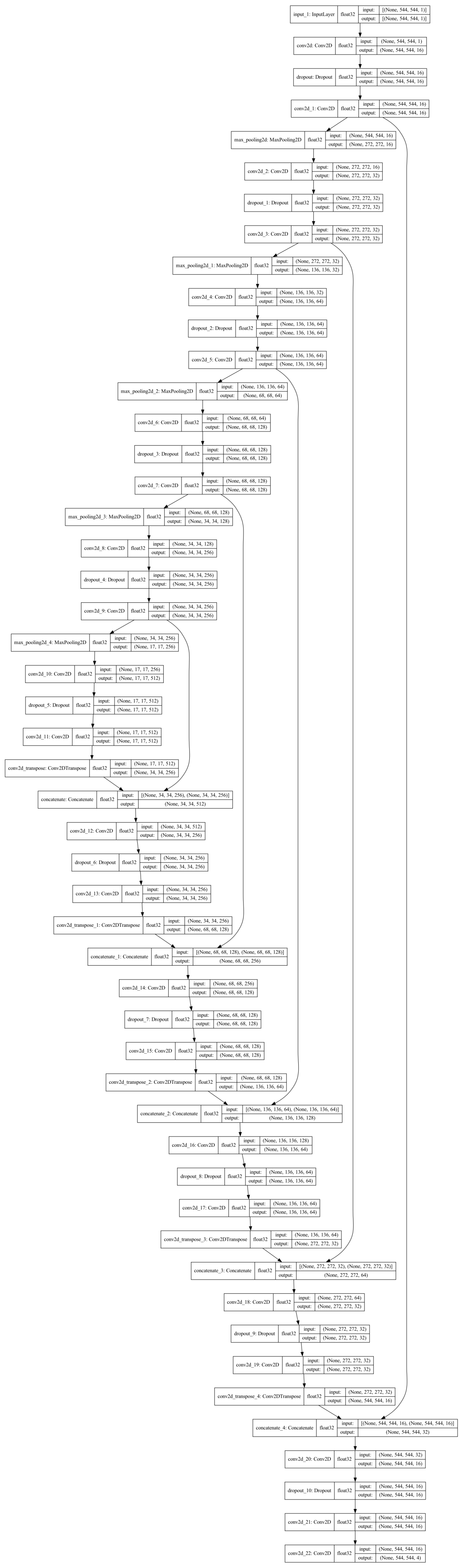

input_1: InputLayer float32 input: [(None, 544, 544, 1)] output: [(None, 544, 544, 1)]
conv2d: Conv2D float32 input: (None, 544, 544, 1) output: (None, 544, 544, 16)
dropout: Dropout float32 input: (None, 544, 544, 16) output: (None, 544, 544, 16)
conv2d_1: Conv2D float32 input: (None, 544, 544, 16) output: (None, 544, 544, 16)
max_pooling2d: MaxPooling2D float32 input: (None, 544, 544, 16) output: (None, 272, 272, 16)
conv2d_2: Conv2D float32 input: (None, 272, 272, 16) output: (None, 272, 272, 32)
dropout_1: Dropout float32 input: (None, 272, 272, 32) output: (None, 272, 272, 32)
conv2d_3: Conv2D float32 input: (None, 272, 272, 32) output: (None, 272, 272, 32)
max_pooling2d_1: MaxPooling2D float32 input: (None, 272, 272, 32) output: (None, 136, 136, 32)
conv2d_4: Conv2D float32 input: (None, 136, 136, 32) output: (None, 136, 136, 64)
dropout_2: Dropout float32 input: (None, 136, 136, 64) output: (None, 136, 136, 64)
conv2d_5: Conv2D float32 input: (None, 136, 136, 64) output: (None, 136, 136, 64)
max_pooling2d_2: MaxPooling2D float32 input: (None, 136, 136, 64) output: (None, 68, 68, 64)
conv2d_6: Conv2D float32 input: (None, 68, 68, 64) output: (None, 68, 68, 128)
dropout_3: Dropout float32 input: (None, 68, 68, 128) output: (None, 68, 68, 128)
conv2d_7: Conv2D float32 input: (None, 68, 68, 128) output: (None, 68, 68, 128)
max_pooling2d_3: MaxPooling2D float32 input: (None, 68, 68, 128) output: (None, 34, 34, 128)
conv2d_8: Conv2D float32 input: (None, 34, 34, 128) output: (None, 34, 34, 256)
dropout_4: Dropout float32 input: (None, 34, 34, 256) output: (None, 34, 34, 256)
conv2d_9: Conv2D float32 input: (None, 34, 34, 256) output: (None, 34, 34, 256)
max_pooling2d_4: MaxPooling2D float32 input: (None, 34, 34, 256) output: (None, 17, 17, 256)
conv2d_10: Conv2D float32 input: (None, 17, 17, 256) output: (None, 17, 17, 512)
dropout_5: Dropout float32 input: (None, 17, 17, 512) output: (None, 17, 17, 512)
conv2d_11: Conv2D float32 input: (None, 17, 17, 512) output: (None, 17, 17, 512)
conv2d_transpose: Conv2DTranspose float32 input: (None, 17, 17, 512) output: (None, 34, 34, 256)
concatenate: Concatenate float32 input: [(None, 34, 34, 256), (None, 34, 34, 256)] output: (None, 34, 34, 512)
conv2d_12: Conv2D float32 input: (None, 34, 34, 512) output: (None, 34, 34, 256)
dropout_6: Dropout float32 input: (None, 34, 34, 256) output: (None, 34, 34, 256)
conv2d_13: Conv2D float32 input: (None, 34, 34, 256) output: (None, 34, 34, 256)
conv2d_transpose_1: Conv2DTranspose float32 input: (None, 34, 34, 256) output: (None, 68, 68, 128)
concatenate_1: Concatenate float32 input: [(None, 68, 68, 128), (None, 68, 68, 128)] output: (None, 68, 68, 256)
conv2d_14: Conv2D float32 input: (None, 68, 68, 256) output: (None, 68, 68, 128)
dropout_7: Dropout float32 input: (None, 68, 68, 128) output: (None, 68, 68, 128)
conv2d_15: Conv2D float32 input: (None, 68, 68, 128) output: (None, 68, 68, 128)
conv2d_transpose_2: Conv2DTranspose float32 input: (None, 68, 68, 128) output: (None, 136, 136, 64)
concatenate_2: Concatenate float32 input: [(None, 136, 136, 64), (None, 136, 136, 64)] output: (None, 136, 136, 128)
conv2d_16: Conv2D float32 input: (None, 136, 136, 128) output: (None, 136, 136, 64)
dropout_8: Dropout float32 input: (None, 136, 136, 64) output: (None, 136, 136, 64)
conv2d_17: Conv2D float32 input: (None, 136, 136, 64) output: (None, 136, 136, 64)
conv2d_transpose_3: Conv2DTranspose float32 input: (None, 136, 136, 64) output: (None, 272, 272, 32)
concatenate_3: Concatenate float32 input: [(None, 272, 272, 32), (None, 272, 272, 32)] output: (None, 272, 272, 64)
conv2d_18: Conv2D float32 input: (None, 272, 272, 64) output: (None, 272, 272, 32)
dropout_9: Dropout float32 input: (None, 272, 272, 32) output: (None, 272, 272, 32)
conv2d_19: Conv2D float32 input: (None, 272, 272, 32) output: (None, 272, 272, 32)
conv2d_transpose_4: Conv2DTranspose float32 input: (None, 272, 272, 32) output: (None, 544, 544, 16)
concatenate_4: Concatenate float32 input: [(None, 544, 544, 16), (None, 544, 544, 16)] output: (None, 544, 544, 32)
conv2d_20: Conv2D float32 input: (None, 544, 544, 32) output: (None, 544, 544, 16)
dropout_10: Dropout float32 input: (None, 544, 544, 16) output: (None, 544, 544, 16)
conv2d_21: Conv2D float32 input: (None, 544, 544, 16) output: (None, 544, 544, 16)
conv2d_22: Conv2D float32 input: (None, 544, 544, 16) output: (None, 544, 544, 4)

Figure A.3: U-Net with 11 layers

A.3 Trained Models

All the trained models mentioned in this thesis are availible on Google Drive in the link below:

Figure A.4: Google Drive with trained models

`https://drive.google.com/drive/folders/1I9BIEL_nJYIU_yoD-zGbWxn2U8cgBE-k?usp=sharing`

A.4 First Glance Overview

This attachment shows the general visual overview of the trained models' segmentation capabilities. By analyzing these specific cases in-between the training sessions, I was then able to make better assumptions about which parameters should be modified for the next training. It also offers and intuitive representation of how each model handles different kind of input.

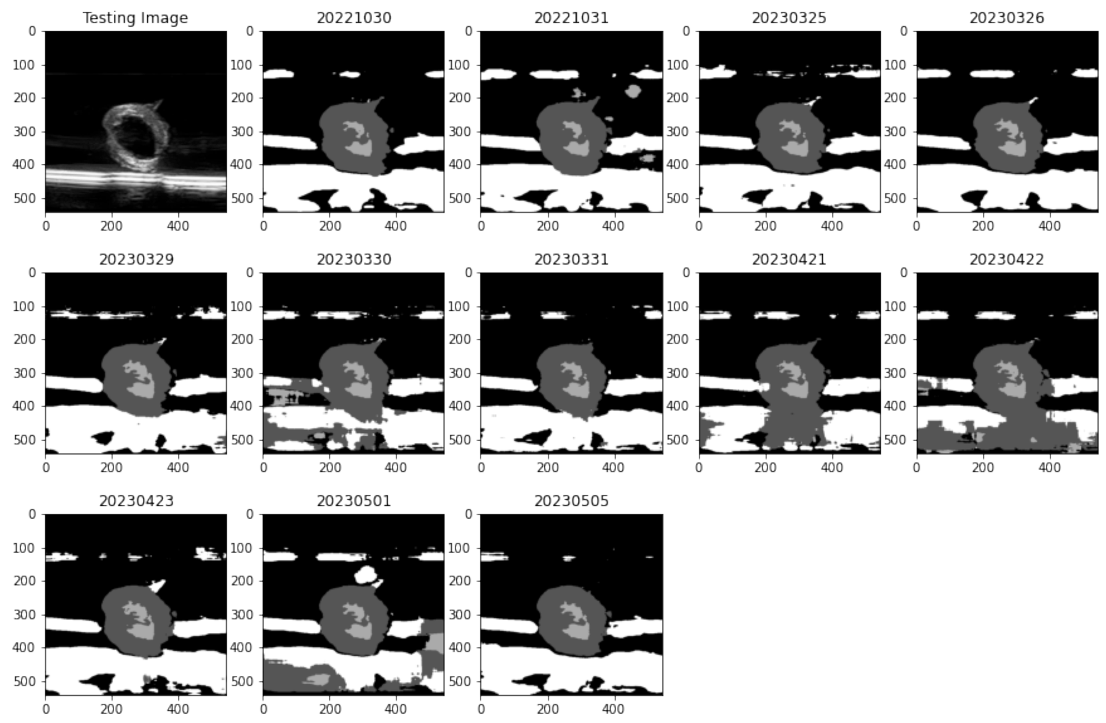


Figure A.5: Original-dataset-like image

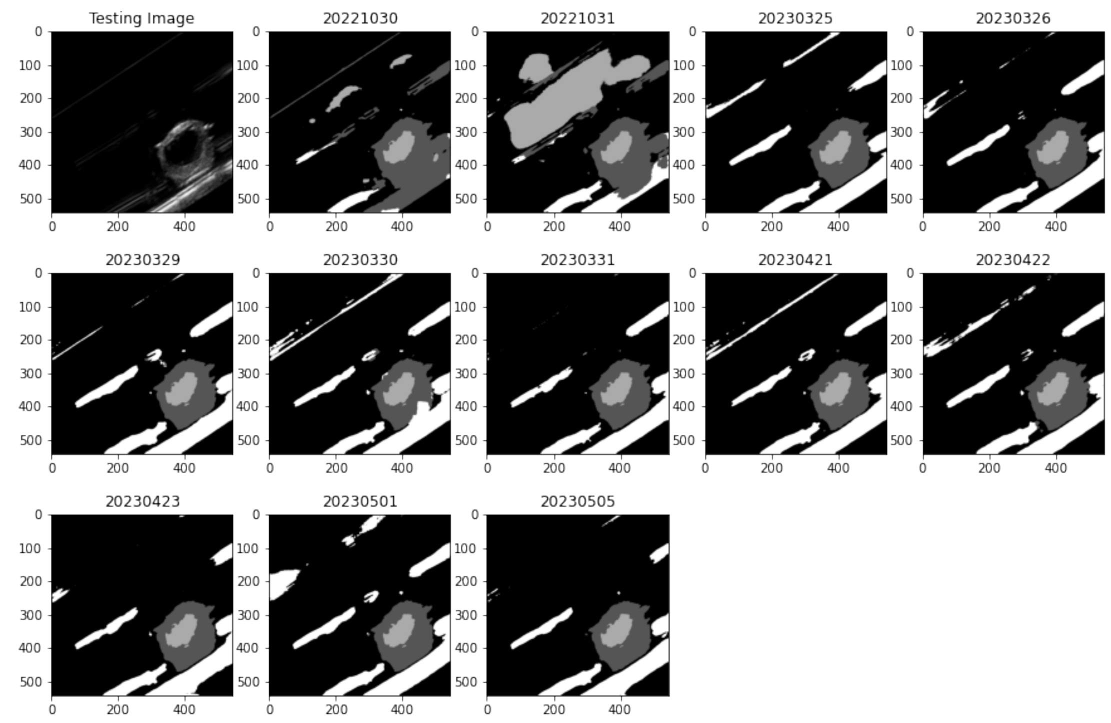


Figure A.6: Rotated image testing

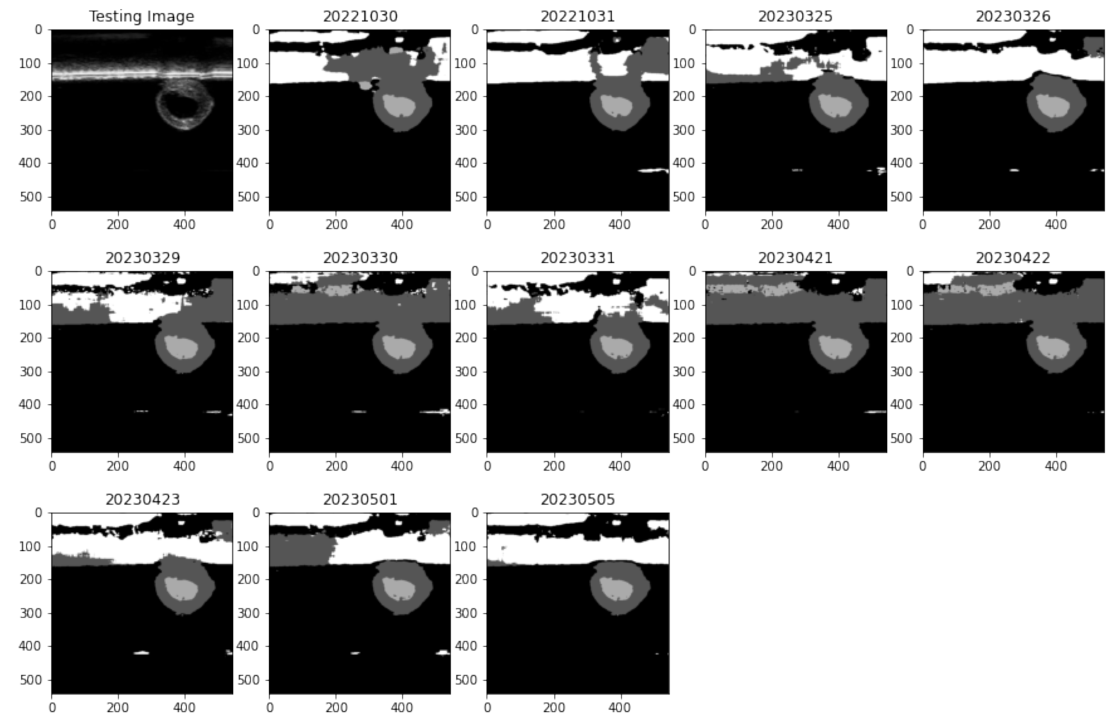


Figure A.7: Upside down image

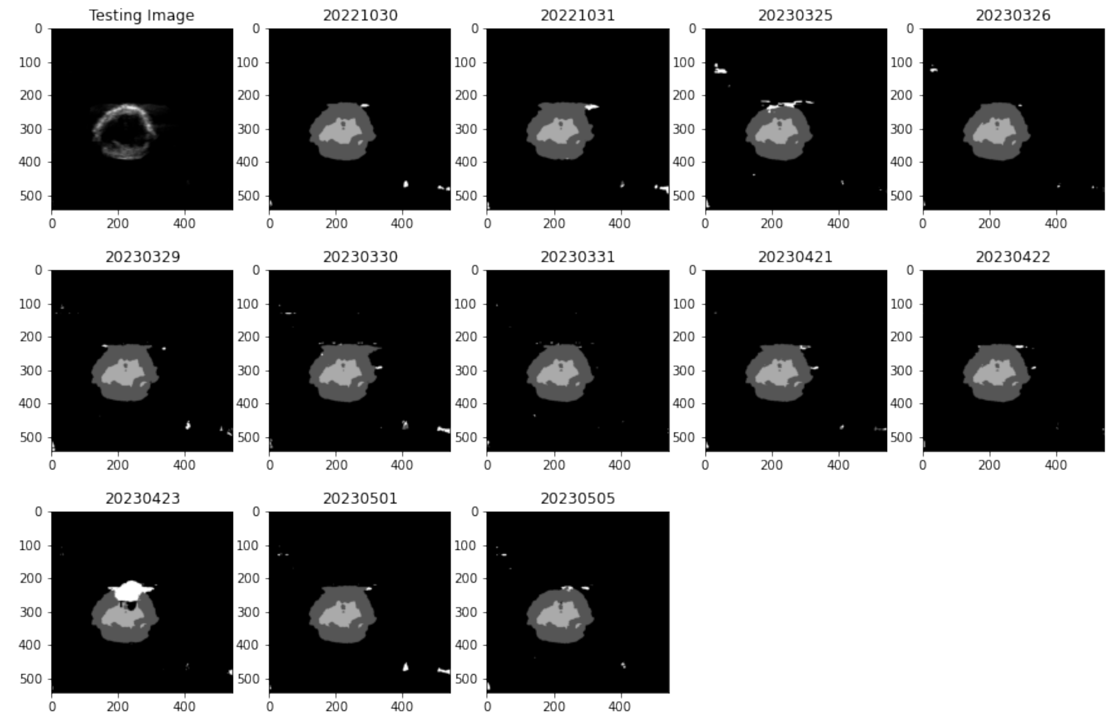


Figure A.8: Image without artifacts - small plaque

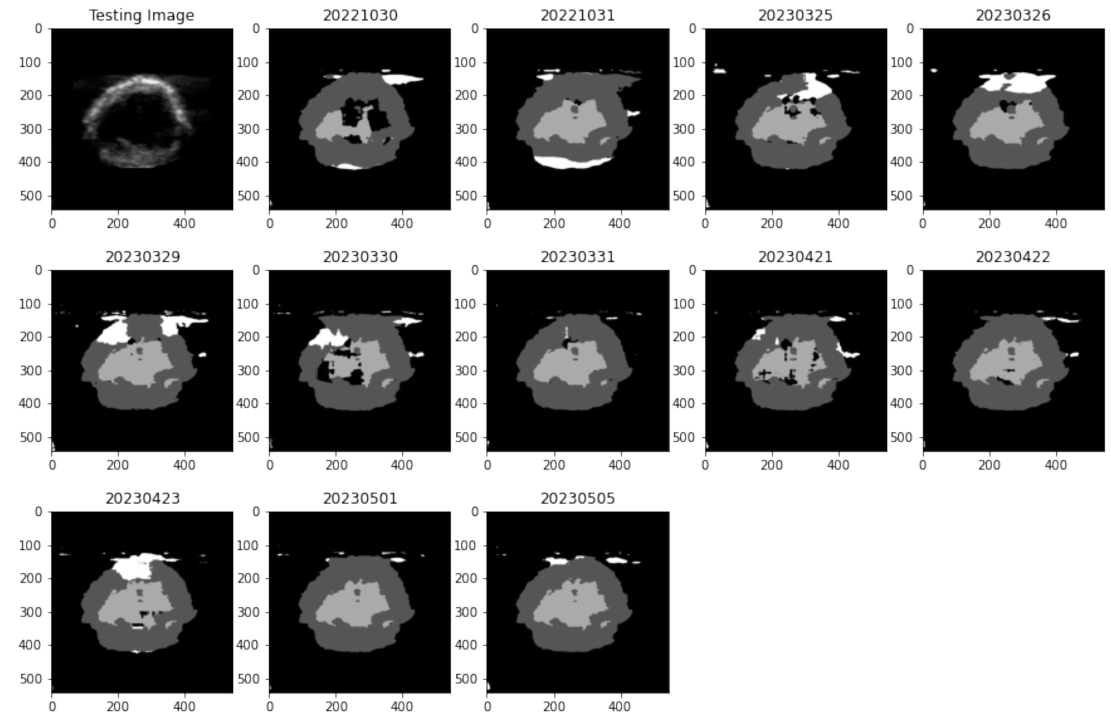


Figure A.9: Image without artifacts - larger plaque

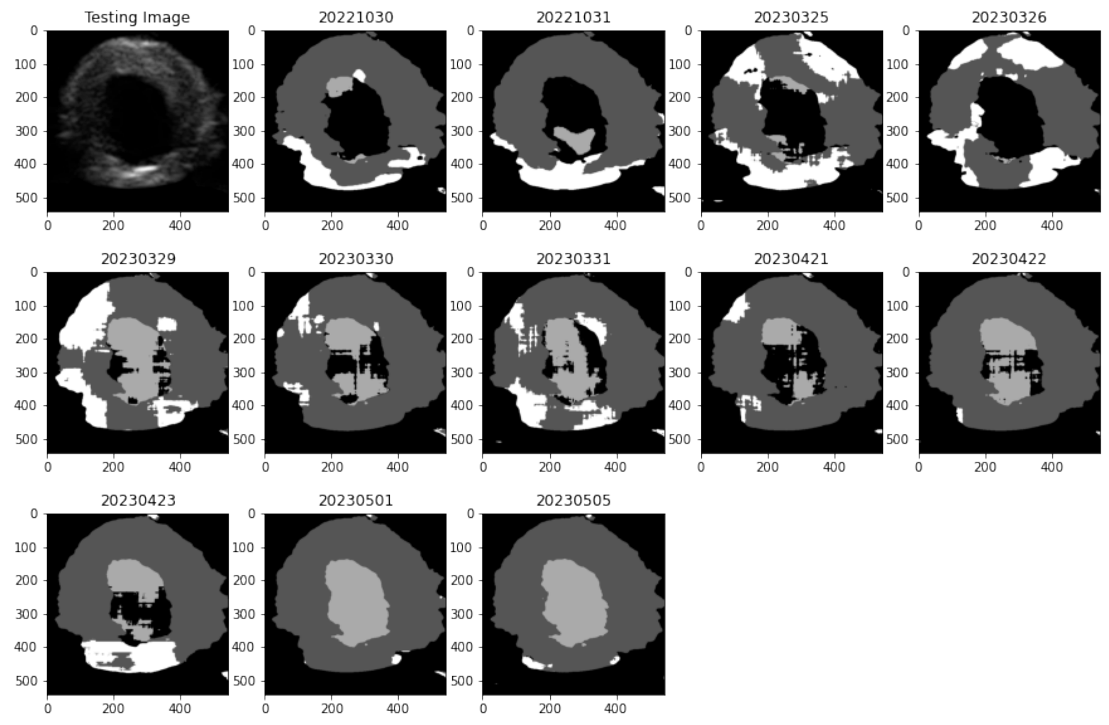


Figure A.10: Image without artifacts - large plaque

A.5 Image augmentation examples

The "translate" augmentation in the figure is the same as `width_shift` and `height_shift` in the Tensorflow `ImageDataGenerator`.

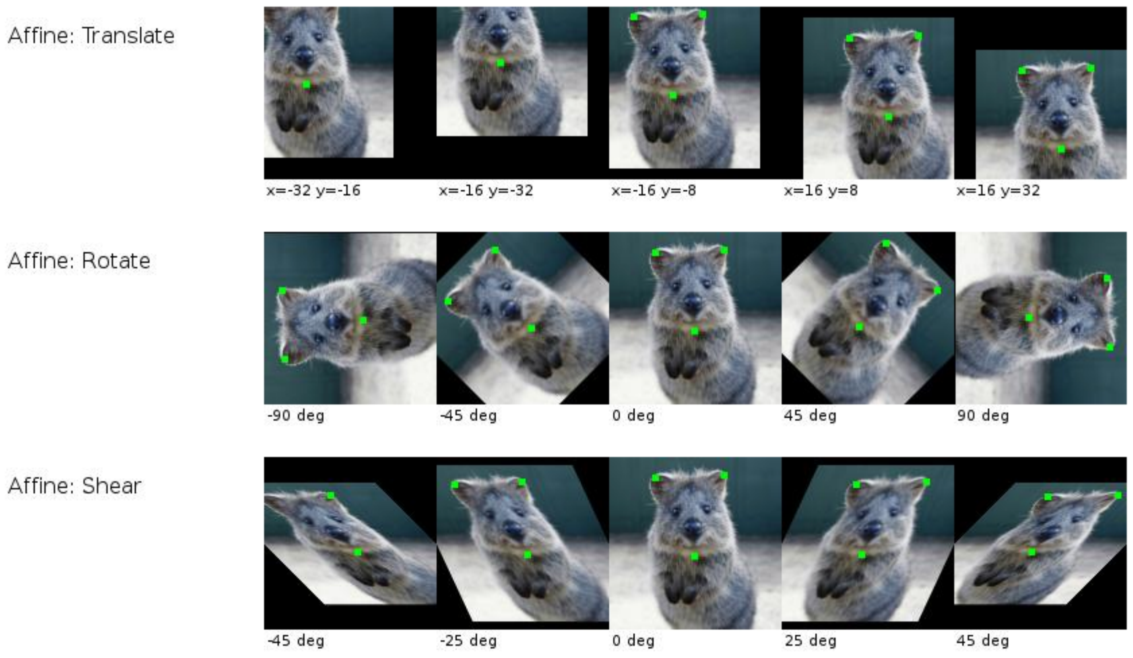


Figure A.11: Examples of image augmentation, from [46]

Appendix B

References

1. *What Is Atherosclerosis?* [online]. National Heart, Lung, and Blood Institute, 2022-03-24 [visited on 2023-05-10]. Available from: `https://www.nhlbi.nih.gov/health/atherosclerosis`.
2. *Atherosclerosis* [online]. Medical News | Medical Articles, 2019-05-29 [visited on 2023-05-10]. Available from: `https://www.news-medical.net/health/Atherosclerosis.aspx`.
3. *Overview* [online]. Mayo Clinic, 2023 [visited on 2023-05-10]. Available from: `https://www.mayoclinic.org/diseases-conditions/arteriosclerosis-atherosclerosis/symptoms-causes/syc-20350569`.
4. *Prevention* [online]. National Heart, Lung, and Blood Institute, 2022-03-24 [visited on 2023-05-10]. Available from: `https://www.nhlbi.nih.gov/health/atherosclerosis/prevention`.
5. NATIONAL HEART, Lung; INSTITUTE, Blood. *Normal artery and an artery with plaque buildup.* 2022. Available also from: `https://www.nhlbi.nih.gov/sites/default/files/inline-images/Atherosclerosis%5C%20diagram.gif`. [Online; accessed May 10, 2023].
6. JAN, Hendl. *Big data: Věda o datech - základy a aplikace.* Grada Publishing a.s., 2021. ISBN 978-80-271-3031-3.
7. HOLLEY, L. H.; KARPLUS, M. Protein secondary structure prediction with a neural network. *Proceedings of the National Academy of Sciences* [online]. 1989, vol. 86, pp. 152–156 [visited on 2023-05-11]. Available from DOI: `10.1073/pnas.86.1.152`.
8. CALUDE, Cristian S; HEIDARI, Shahrokh; SIFAKIS, Joseph. What perceptron neural networks are (not) good for? *Information Sciences.* 2023, vol. 621, pp. 844–857.
9. PLATFORM, Simplilearn | Online Courses - Bootcamp Certification. *Model of node with 4 inputs.* 2023. Available also from: `https://www.simplilearn.com/ice9/free_resources_article_thumb/Perceptron/general-diagram-of-perceptron-for-supervised-learning_4.jpg`. [Online; accessed May 11, 2023].

10. BANOULA, Mayank (comp.). *What is Perceptron: A Beginners Guide for Perceptron* [online]. Simplilearn | Online Courses - Bootcamp Certification Platform, 2023-05-10 [visited on 2023-05-11]. Available from: `https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron`.

11. TECHNOLOGY, O'Reilly Media -; TRAINING, Business. *ANN with 4 layers.* 2023. Available also from: `https://www.oreilly.com/api/v2/epubs/9781838642709/files/assets/61bc8450-f3ac-4d81-b405-3d748e30d04a.png`. [Online; accessed May 11, 2023].

12. CHOLLET, François; PECINOVSKÝ, Rudolf. *Deep learning v jazyku Python : knihovny Keras, Tensorflow.* Grada Publishing, 2019.

13. SAHA, Sumit (comp.). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way* [online]. Saturn Cloud | Your data science cloud environment, 2018-12-15 [visited on 2023-05-14]. Available from: `https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/`.

14. SMITH, Steven W. *The scientist and engineer's guide to digital signal processing.* California Technical Pub, 1999. Available also from: `https://www.analog.com/media/en/technical-documentation/dsp-book/dsp_book_ch6.pdf`.

15. KIMURA, Nobuaki. *Depiction of the convolution layer with a filter in convolutional neural network (CNN).* 2023. Available also from: `https://www.researchgate.net/publication/338190342/figure/fig3/AS:840701663268864@1577450304039/Depiction-of-the-convolution-layer-with-a-filter-in-convolutional-neural-network-CNN_W640.jpg`. [Online; accessed May 12, 2023].

16. VIDHYA.COM, Analytics. *CNN Schema.* 2023. Available also from: `https://editor.analyticsvidhya.com/uploads/94787Convolutional-Neural-Network.jpeg`. [Online; accessed May 12, 2023].

17. SHARMA, Pranshu (comp.). *Basic Introduction to Convolutional Neural Network in Deep Learning* [online]. Analytics Vidhya | Learn everything about Data Science, 2022-03-01 [visited on 2023-05-12]. Available from: `https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-convolutional-neural-network-in-deep-learning/`.

18. *How ReLU and Dropout Layers Work in CNNs* [online]. Baeldung, 2023-04-14 [visited on 2023-05-14]. Available from: `https://www.baeldung.com/cs/ml-relu-dropout-layers`.

19. BAELDUNG. *Dropout.* 2023. Available also from: `https://www.baeldung.com/wp-content/uploads/sites/4/2020/05/2-1-2048x745-1.jpg`. [Online; accessed May 14, 2023].

20. MESSINA, Domenico; BORRELLI, Pasquale; RUSSO, Paolo; SALVATORE, Marco; AIELLO, Marco. Voxel-Wise Feature Selection Method for CNN Binary Classification of Neuroimaging Data. *Frontiers in Neuroscience.* 2021, vol. 15. ISSN 1662-453X. Available from DOI: `10.3389/fnins.2021.630747`.

21. BERTELS, Jeroen; ROBBEN, David; LEMMENS, Robin; VANDERMEULEN, Dirk. *Convolutional neural networks for medical image segmentation.* 2022. Available from arXiv: `2211.09562 [cs.CV]`.

22. CIRESAN, Dan; GIUSTI, Alessandro; GAMBARDELLA, Luca; SCHMIDHUBER, Jürgen. Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images. 2012, vol. 25. Available also from: `https://proceedings.neurips.cc/paper_files/paper/2012/file/459a4ddcb586f24efd9395aa7662bc7c-Paper.pdf`.

23. LI, Hongsheng; ZHAO, Rui; WANG, Xiaogang. *Highly Efficient Forward and Backward Propagation of Convolutional Neural Networks for Pixelwise Classification.* 2014. Available from arXiv: `1412.4526 [cs.CV]`.

24. SHELHAMER, Evan; LONG, Jonathan; DARRELL, Trevor. Fully convolutional networks for semantic segmentation. *IEEE transactions on pattern analysis and machine intelligence.* 2017, vol. 39, no. 4, pp. 640–651.

25. RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas. U-net: Convolutional networks for biomedical image segmentation. In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18.* Springer, 2015, pp. 234–241.

26. KAMNITSAS, Konstantinos; LEDIG, Christian; NEWCOMBE, Virginia FJ; SIMPSON, Joanna P; KANE, Andrew D; MENON, David K; RUECKERT, Daniel; GLOCKER, Ben. Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation. *Medical image analysis.* 2017, vol. 36, pp. 61–78.

27. WANG, Ziyang. *Deep Learning in Medical Ultrasound Image Segmentation: a Review.* 2021. Available from arXiv: `2002.07703 [eess.IV]`.

28. NOH, Hyeonwoo; HONG, Seunghoon; HAN, Bohyung. *Proceedings of the IEEE International Conference on Computer Vision (ICCV).* Learning Deconvolution Network for Semantic Segmentation. 2015.

29. CUNNINGHAM, Ryan; SÁNCHEZ, Maria B; LORAM, Ian D. Ultrasound segmentation of cervical muscle during head motion: A dataset and a benchmark using deconvolutional neural networks. 2019.

30. KIM, Sekeun; JANG, Yeonggul; JEON, Byunghwan; HONG, Youngtaek; SHIM, Hackjoon; CHANG, Hyukjae. Fully automatic segmentation of coronary arteries based on deep neural network in intravascular ultrasound images. In: *Intravascular Imaging and Computer Assisted Stenting and Large-Scale Annotation of Biomedical Data and Expert Label Synthesis: 7th Joint International Workshop, CVII-STENT 2018 and Third International Workshop, LABELS 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 16, 2018, Proceedings 3.* Springer, 2018, pp. 161–168.

31. RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas. *U-Net: Convolutional Networks for Biomedical Image Segmentation.* 2015. Available from arXiv: `1505.04597 [cs.CV]`.

32. LEI, Yang; TIAN, Sibo; HE, Xiuxiu; WANG, Tonghe; WANG, Bo; PATEL, Pretesh; JANI, Ashesh B; MAO, Hui; CURRAN, Walter J; LIU, Tian, et al. Ultrasound prostate segmentation based on multidirectional deeply supervised V-Net. *Medical physics*. 2019, vol. 46, no. 7, pp. 3194–3206.

33. AKKUS, Zeynettin; GALIMZIANOVA, Alfiia; HOOGI, Assaf; RUBIN, Daniel L; ERICKSON, Bradley J. Deep learning for brain MRI segmentation: state of the art and future directions. *Journal of digital imaging*. 2017, vol. 30, pp. 449–459.

34. KURAMA, Vihar (comp.). *PyTorch vs. TensorFlow: Key Differences to Know for Deep Learning* [online]. Built In: National Tech Startups, 2022-06-10 [visited on 2023-05-07]. Available from: `https://builtin.com/data-science/pytorch-vs-tensorflow`.

35. *Keras Tutorial* [online]. Javatpoint, 2022-06-10 [visited on 2023-05-07]. Available from: `https://www.javatpoint.com/keras`.

36. *What is NumPy?* [online]. NumPy, 2022 [visited on 2023-05-08]. Available from: `https://numpy.org/doc/stable/user/whatisnumpy.html`.

37. TEAM, OpenCV (comp.). *About* [online]. OpenCV, 2023 [visited on 2023-05-08]. Available from: `https://opencv.org/about/`.

38. PEDREGOSA, Fabian; VAROQUAUX, Gaël; GRAMFORT, Alexandre; MICHEL, Vincent; THIRION, Bertrand; GRISEL, Olivier; BLONDEL, Mathieu; PRETTENHOFER, Peter; WEISS, Ron; DUBOURG, Vincent; VANDERPLAS, Jake; PASSOS, Alexandre; COURNAPEAU, David; BRUCHER, Matthieu; PERROT, Matthieu; DUCHESNAY, Édouard. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, vol. 12, no. 85, pp. 2825–2830. Available also from: `http://jmlr.org/papers/v12/pedregosa11a.html`.

39. *Sponsored Projects* [online]. Austin: NumFOCUS, 2023 [visited on 2023-05-08]. Available from: `https://numfocus.org/sponsored-projects`.

40. *API Reference* [online]. Matplotlib - Visualization with Python, 2023 [visited on 2023-05-08]. Available from: `https://matplotlib.org/stable/api/index.html#module-pylab`.

41. *Jupyter Notebook: What is Jupyter Notebook?* [online]. San Francisco: Domino Data Lab, Inc., 2023 [visited on 2023-04-11]. Available from: `https://www.dominodatalab.com/data-science-dictionary/jupyter-notebook`.

42. *About MetaCentrum* [online]. Prague: MetaCentrum (MetaVO) - virtuální organizace pro celou akademickou obec, 2021-12-23 [visited on 2023-04-11]. Available from: `https://metavo.metacentrum.cz/en/about/index.html`.

43. *TensorFlow normalize* [online]. EDUCBA | Best Online Training Video Courses Certification, 2023 [visited on 2023-05-09]. Available from: `https://www.educba.com/tensorflow-normalize/`.

44. *tf.keras.layers.Normalization* [online]. TensorFlow, 2023-03-23 [visited on 2023-05-09]. Available from: `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Normalization`.

45. *Python zip() Function* [online]. W3Schools Online Web Tutorials, 2023 [visited on 2023-05-12]. Available from: `https://www.w3schools.com/python/ref_func_zip.asp`.

46. WIKIDOCS. *Examples of Image Augmentation*. 2023. Available also from: `https://wikidocs.net/images/page/164364/Augmentationpng.png`. [Online; accessed May 17, 2023].