Martin Pettersen

Segmentation of MR Images Using CNN

Ålesund – 01.10.2019

Master Thesis

NTNU
Norwegian University of
Science and Technology

Norwegian University of
Science and Technology

Abstract

The need for digital segmentation of human body parts is growing and having a digital twin of the body part with an injury, or a planned surgery can help people understand that body part better and be prepared for the tasks to come. To segment bodyparts are a helpful tool to both understand and examen the human body. You can add a segmented body part in a 3-dimensional world, look at it from different angles and discover something you could not have seen from a 2-dimensional view.

During the work of this thesis, there was created a Python-script of a Convolutional Neural Network (CNN) using Keras with TensorFlow as backend. This CNN is a U-Net inspired network created to classify each pixel in an MRI scan of a knee joint. It classifies bones, PCL, ACL which all are tissues located in the knee joint. The results are promising an there where several discoveries in the result of this thesis. The weighted loss function is a necessary function to classify lower weighted tissues such as the ligaments (PCL and ACL). And it also showed that the commonly used inputs not necessary are the best inputs for this network. The single input image T1 performed slightly better than the known method by using all of the images as an input.

Sammendarag

Anvendbarheten av digital segmentering av kroppsdeler er en økende trend, og det å ha en digital tvilling av skadede legemer, eller en planlagt operasjon kan bedre forståelsen av den kroppsdelen bedre, og forberede partene involvert. Segmentering av kroppsdeler er et nyttig verktøy for å bedre forstå og undersøke menneskekroppen. Segmenterte legemer kan vises som 3D-modeller, og manipuleres slik at den kan ses fra forskjellige vinkler, og muligens oppdage noe som ikke var like synlig i det originale bildesettet.

I løpet av masterprosjektet ble det utviklet et Python-script av et Convolutional Neural Network (CNN) ved hjelp av Keras med TensorFlow som backend. Dette CNN-nettverket er inspirert av U-Net og lagd for å klassifisere hver enkelt piksel i MRI-skanninger av kneledd. Den klassifiserer ben, fremre-, og bakre korsbånd, som alle er vev som ligger i kneleddet. Resultatene er lovende, og det ble avdekket flere gode resultater. En vektet tapsfunksjon er nødvendig for å klassifisere vev som opptrer sjeldnere, slik som korsbåndene. I tillegg viste det at den mest brukte inndataen ikke nødvendigvis var det beste. Et enkelt type bilde (T1) gjenga noe bedre resultater enn en etablert metode der alle bildetyper blir brukt.

Contents

Abstract **iii**

Sammendrag **v**

1 Introduction **1**

1.1 Problem and Motivation 2

1.2 Scope 2

1.3 Research Question 3

1.4 Organization of This Report 4

1.4.1 Chapter One 4

1.4.2 Chapter Two 4

1.4.3 Chapter Three 4

1.4.4 Chapter Four 5

1.4.5 Chapter Five 5

1.4.6 Chapter Six 5

2 Theory and Literature Review **6**

2.1 Knee Anatomy 8

2.1.1 Bones 8

2.1.2 Ligaments 9

2.1.3 Meniscus 9

2.1.4 Tendon 10

2.1.5 Nerve 10

2.1.6 Blood Vessels 11
2.1.7 Injuries 11
2.2 Magnetic Resonance Imaging 12
2.2.1 MRI Components 12
2.3 MRI Images File Format 14
2.4 Machine Learning 15
2.5 Neural Networks 16
2.5.1 Input Layer 16
2.5.2 Hidden Layers 17
2.5.3 Neurons 17
2.5.4 Activation Function 17
2.5.5 Output layer 18
2.5.6 Data Set 18
2.5.7 Multi-Class Classification 19
2.5.8 One-Hot Encoding 19
2.6 Convolutional Neural Network 19
2.6.1 Convolution Layer 20
2.6.2 Pooling Layer 21
2.6.3 Fully Connected Layer 21

3 Methodology **23**
3.1 Software 24
3.1.1 Keras and TensorFlow 24
3.1.2 Computer Specifications 24
3.2 Data-set 25
3.2.1 Raw Data 25
3.2.2 Mask/Label Annotation 27
3.3 Program 28
3.3.1 UNet 30
3.3.2 Activation Function 31

3.3.3 Loss Function . . . 32
3.4 Evaluation . . . 33
3.4.1 Confusion Matrix . . . 33
3.4.2 Measure performance . . . 35

4 Results **38**
4.1 Loss Function . . . 39
4.1.1 Categorical Cross-entropy . . . 40
4.1.2 Weighted Categorical Cross-entropy . . . 41
4.1.3 Weighted Categorical Cross-entropy 2nd Version . . . 42
4.2 Compared results of different input . . . 43
4.2.1 One Input . . . 43
4.2.2 Two inputs . . . 47
4.2.3 Three inputs . . . 51
4.3 Accuracy over Iterations . . . 53

5 Discussion **54**
5.1 Loss-Function with Weights . . . 55
5.2 Best Input to Build a Model . . . 55
5.3 Best Model . . . 56
5.4 Model error vs. Human error . . . 58

6 Conclusion **61**
6.1 Conclusion and Future Work . . . 62

Bibliography **65**

A All Results **67**

B CNN Summary **80**

C Code **83**

Chapter 1

Introduction

1.1 Problem and Motivation **2**

1.2 Scope **2**

1.3 Research Question **3**

1.4 Organization of This Report **4**

1.4.1 Chapter One 4

1.4.2 Chapter Two 4

1.4.3 Chapter Three 4

1.4.4 Chapter Four 5

1.4.5 Chapter Five 5

1.4.6 Chapter Six 5

1.1 Problem and Motivation

A well functioning knee joint is essential to mobility and a important part of daily activities as standing, walking and running. Knee injuries, for example injuries on ligaments, meniscus, cartilage or tendons is common in sport among professional and amateur athletes. These injuries may damage their career or life quality in a significant.

Joint diseases and load over a longer time span may also also affect the knee joint mobility, and may also affect the life quality.

A lot of this injuries is a non life-threatening and the consequences from the injuries are not reducing life-quality enough. This makes the queues long and you usually have to wait a long time to get the diagnosis you need to continue the treatment of the injury.

The Magnetic resonance imaging (MRI) is a widely used technique to image and diagnose injuries like knee-injuries and gives the doctors the ability to look inside tissues and organs of the body. The data produced by MRI is not segmented. They need some kind of segmentation tool to make the segmentation of the images. These tools use a lot of time and often end up with errors in the segmentation. This is limiting the quantitative uses of the MRI images and humans has to correct the errors.

Successful treatment of a knee with injuries depends on well trained doctors and a good knowledge about the injury and the rest of the anatomy in the knee. In this master thesis we will look into how Convolutional Neural Networks (CNN) can automatically segment MRI images form knees and further findings on how to optimize the CNN.

1.2 Scope

This thesis will study the use of deep learning and the method named Convolutional Neural Network within computer science. MRI images are a tool to map the inside

tissues of the body, and are widely used in medicine to diagnose injuries in the knee joint.

The scope of this thesis is therefore:

- Computer Science - Artificial Intelligence / Machine Learning - Deep Learning, CNN
- Medicine - MRI - MRI segmentation

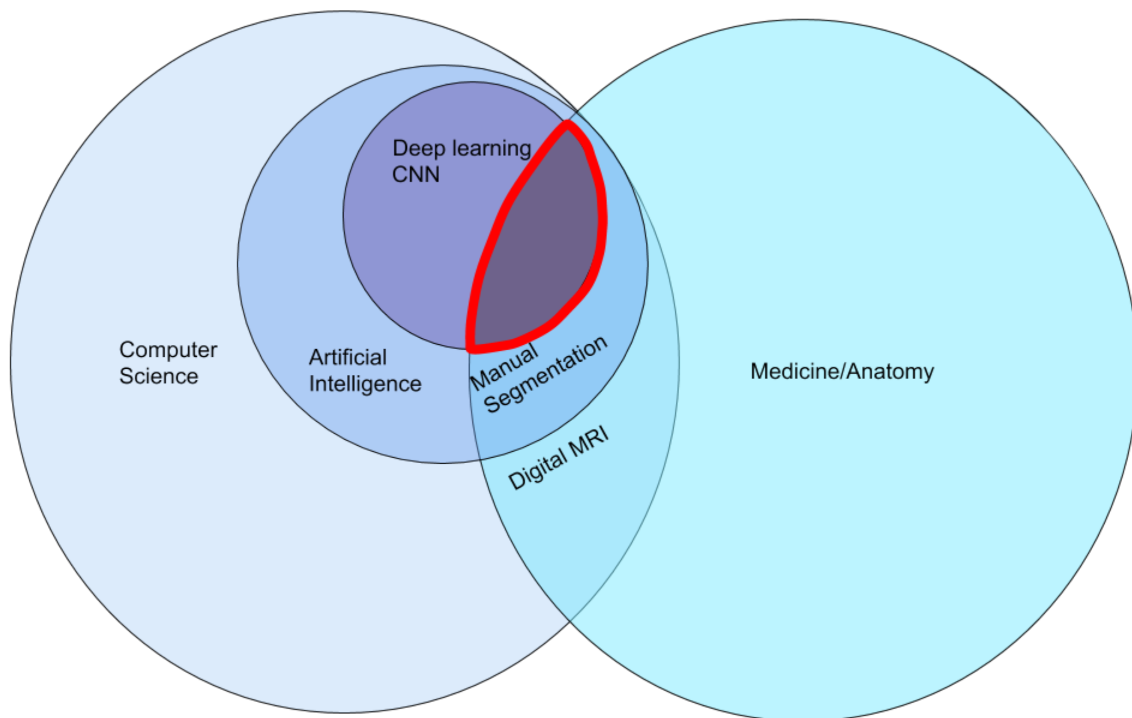

Figure 1.1: *Scope*

1.3 Research Question

Based on what has been mentioned in this document, the following research questions are formulated:

- Does the proposed Convolutional Neural Network (CNN) perform a good segmentation and make it more automatic then today's solution?
- How does the CNN perform in processing speed compared to older segmentation methods?
- What can be done to optimize the segmentation results?

1.4 Organization of This Report

The following document consist of six chapters that will cover this master thesis. Here is a overview of the chapters presented in this document.

1.4.1 Chapter One

The introduction is explaining the problem and gives a overview about the study and describe why this is a wanted tool.

1.4.2 Chapter Two

This chapter covers the literature review and the previous work related to this topic. It covers relevant information for understanding this thesis, and describes the scope more detailed.

1.4.3 Chapter Three

This chapter includes methods and descriptions of the CNN and processes used to solve the problem.

1.4.4 Chapter Four

The results are presented in this chapter with a description.

1.4.5 Chapter Five

Discussion of the results discovered in chapter four.

1.4.6 Chapter Six

Conclusions from this thesis.

Chapter 2

Theory and Literature Review

2.1 Knee Anatomy **8**

2.1.1 Bones 8

2.1.2 Ligaments 9

2.1.3 Meniscus 9

2.1.4 Tendon 10

2.1.5 Nerve 10

2.1.6 Blood Vessels 11

2.1.7 Injuries 11

2.2 Magnetic Resonance Imaging **12**

2.2.1 MRI Components 12

2.3 MRI Images File Format **14**

2.4 Machine Learning **15**

2.5 Neural Networks **16**

2.5.1 Input Layer 16

2.5.2 Hidden Layers 17

2.5.3 Neurons 17

2.5.4 Activation Function 17

2.5.5 Output layer 18

2.5.6 Data Set 18

2.5.7 Multi-Class Classification 19

2.5.8 One-Hot Encoding 19

2.6 Convolutional Neural Network 19

2.6.1 Convolution Layer 20

2.6.2 Pooling Layer 21

2.6.3 Fully Connected Layer 21

2.1 Knee Anatomy

It is important to understand the parts of the knee joint, to know how to evaluate the results of the CNN. The structures of the knee can be divided into several categories such as shown in the following list and figure [2]:

- Bones
- Joints
- Ligaments
- Tendons
- Muscles
- Nerves
- Blood Vessels

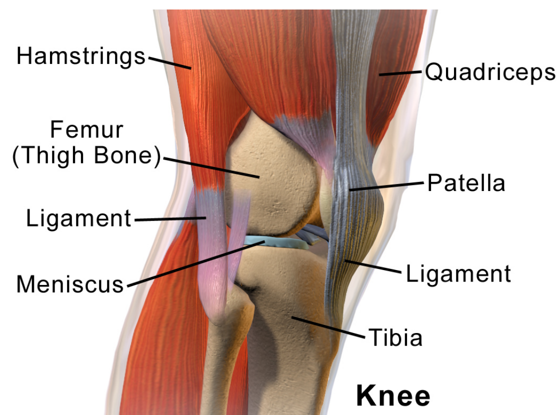

Figure 2.1: *Knee anatomy: https://upload.wikimedia.org/wikipedia/commons/b/bc/Blausen*$_0$*597*$_K$*neeAnatomy*$_S$*ide.p*

2.1.1 Bones

The knee joint is where the two bones femur (top) and tibia (bottom) meet. Other bones close related to the knee joint are the patella (knee cap) and fibula. At the end of each bone, in the joint where the bone meets another bone, you have the articular

cartilage. The articular cartilage has a slippery surface that allows the two surfaces at the end of each bone to slide against each other without damaging one another. The cartilage's main function is to absorb shock and give a slippery surface which helps the motion in the joint.

2.1.2 Ligaments

Ligaments are strong bands of tissue that connect the bones. You can find four ligaments in the knee joint. Two of them are located on each side of the knee joint. The inside ligament is named Medial Collateral Ligament (MCL) and the outside ligament is named Lateral Collateral Ligament (LCL).

The two other ligaments are inside the knee. The Anterior Cruciate Ligament (ACL) stretches from the front of the tibia to the back of the femur, and the Posterior Cruciate Ligament (PCL) stretches from the back of the tibia to the back of the femur. The LCL and MCL prevent movement in the side direction and ACL and PCL prevents movement too far in the front and back direction. The ligaments are the most important tissue to control the stability of the knee.

2.1.3 Meniscus

The meniscus is a fibrocartilage located between the femur and the tibia. The meniscus is important for two reasons:

- They work like a gasket to spread the force of the weight of the body over a larger area.
- They help the ligaments with the stabilization of the knee joint.

The meniscus distributes the weight from the femur over a larger area on the tibia and works like a pillow in between the two bones. This protects the bones from getting to much force and prevents the bones from taking damage. The meniscus is also thicker

on the edges and this helps the femur stay in place and not roll on the tibia.

The meniscus and ligaments are the most important part of the knee when it comes to stabilizing it. Without strong and tight ligaments to connect the two bones in the knee joint, you will end up with a loose knee joint and this can damage the knee.

2.1.4 Tendon

Tendons are similar to ligaments but connect muscles to bones. One of these is the quadriceps tendon that connects the large muscles in the tie named the quadriceps to the patella (knee cap). This tendon continues over the patella and connects to the patellar tendon which connects the patella to the tibia. The hamstring muscles in the back of the tie also have tendons that connect the hamstring with different places at the tibia.

The extensor mechanism in the knee is the motor that allows movements in the knee joint. It sits in the front of the knee joint and exist of the patella, patellar tendon, quadriceps tendon, and the quadriceps muscles. When the quadriceps muscles contract it straightens the knee joint like getting up from a squatting position.

2.1.5 Nerve

The most important nerves in the knee is the tibial nerve and the common peroneal nerve. Those nerves are positioned on the back of the knee. These two nerves travel to the lower leg and foot to give sensation and muscle control in the lower leg and foot. The sciatic nerve splits above the knee joint into the tibial nerve and the common peroneal nerve. Both the tibial nerve and the common peroneal nerve can be damaged by injuries in the knee joint.

2.1.6 Blood Vessels

The major blood vessels in the leg travels with the tibial nerve on the back of the knee. The popliteal artery and popliteal vein is the largest blood supplies to the leg. The artery carries blood out to the foot and the vein carries the blood back to the heart. If big damages happen to the popliteal artery and there are no possibilities for repair, there is most likely not possible to save the leg.

2.1.7 Injuries

The knee has an unstable design and has to support the body's full weight while standing and much more than that when walking, running or jumping. This is one of the reasons why knee problems are a common complaint among people of all ages.

All ligaments in the knee can be injured. It can be stretched, partially torn or completely torn. Among these injuries, the most common injury is completely torn. Symptoms of this are pain, popping sound when the accident happens, instability of the knee joint and joint swelling. In half of the cases where one ligament is torn, the surrounding ligaments, cartilage or meniscus are also damaged.

This includes all of the following ligaments:

- Meniscus tear
- ACL strain or tear
- PCL strain or tear
- MCL strain or tear
- LCL strain or tear

In case of injuries, an experienced hand can be accurate and tell if the ligaments are torn or not. But to confirm the injury it is often used MRI to provide images of the soft tissues like ligaments and cartilage in the knee.

2.2 Magnetic Resonance Imaging

MRI uses a magnetic field and radio frequencies to take pictures of the body instead of ionizing radiation as used in x-ray and CT scans. The magnetic field in an MRI machine is measured in Tesla which represents the magnetic flux density of the magnet. MRI machines are usually from 1.5T to 3.0T. This produces a very strong magnetic field, compared with the magnetic field of the earth that is 0.00003T. The strength of this magnet is strong enough to pick up a car.

Our body is composed of 70% water or H2O and the MRI relies on the magnetic properties of the hydrogen atom to produce MRI images. The hydrogen atom has a single proton in the center of the atom. The atom can be charged with a spinning momentum that produces a magnetic field also named a magnetic moment. Normally the protons are oriented randomly in all directions when there is no magnetic field presence. When the hydrogen atom is in a strong constant magnetic field their magnetic moment line up parallel or anti-parallel to the field. This is also referred to as a longitudinal magnetization. This can be done by a superconducting magnet such as the magnetic field produced by the primary magnet in an MRI machine.

This property of hydrogen in a magnetic field is what makes the MRI possible and is, therefore, an important part of MRI imaging.

2.2.1 MRI Components

The MRI has several components to be able to map the body in a 3-dimensional image. The components are as follows:

- Primary magnet
- Gradient Magnets
- Radio frequency (RF) coils
- Computer system

This chapter will describe the different parts and their properties.

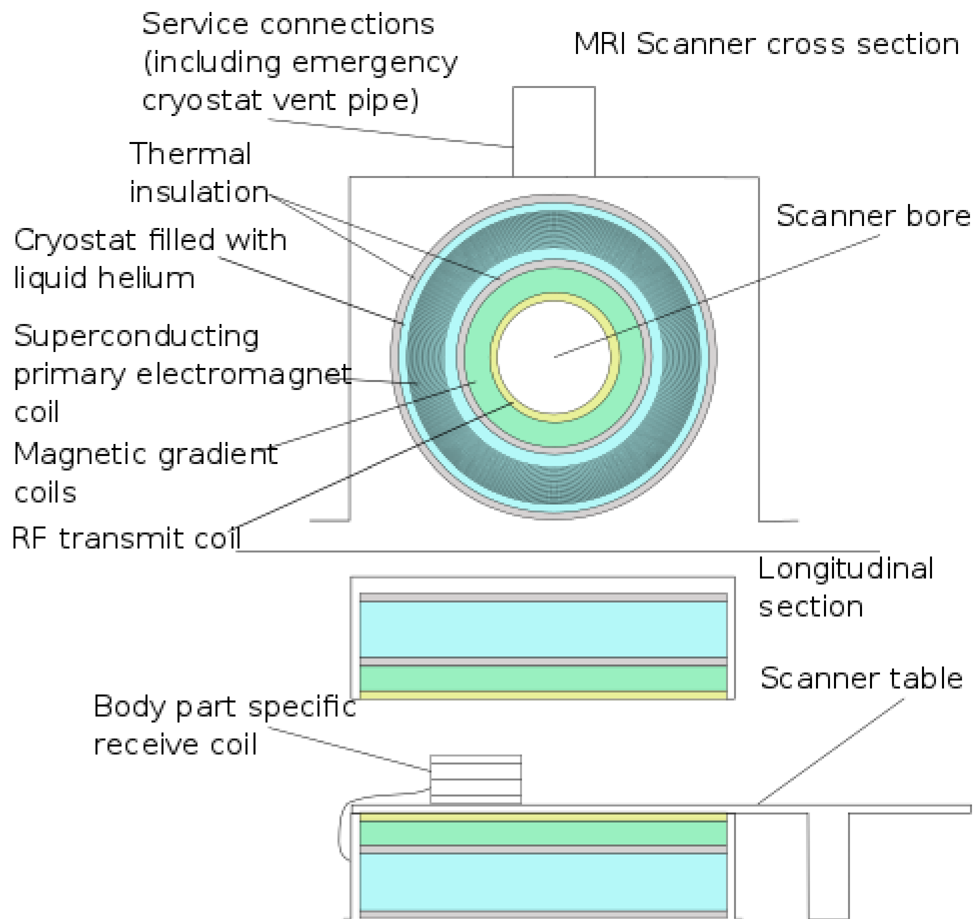

Figure 2.2: *MRI components:* `https://en.wikipedia.org/wiki/File:Mri_scanner_schematic_labelled.svg`

Primary Magnet

This is the magnet which makes the strong constant magnetic field in the MRI machine. As mentioned earlier this magnetic field has a strength between 1.5T and 3.0T and covers the entire machine.

Gradient Magnets

The gradient coils generate a second magnetic field overt the primary field. There are three gradient coils in different directions. The directions are in z, x, y-axis and represent the directions the MRI can take pictures. The gradient coils produce a gradient magnetic field that helps the localization of pixels in the image.

Radio Frequency(RF) Coils

The RF coils are responsible for transmitting the radio frequency or RF pulse and helps you receive images in MRI. The RF coils come in several designs to suit the part of the body that is going to be imaged. The RF pulse is responsible for flipping the protons into a high energy state and decreasing the longitudinal magnetization. It also synchronizes the protons. This turns the magnetization vector of the proton towards the transverse plane also named a transverse magnetization. The proton does not stay in this state for a long time and will end up in its normal state within a small amount of time. This change induces an electrical signal in the RF receptions and the transverse plane picks up this signal and stores it in the computer system.

Computer System

The computer system receives the signal from the RF and converts it from analog to a digital signal. This data is then applied to a Fourier Transformation and this produces the result image.

2.3 MRI Images File Format

The results produced from an MRI is an output file named DICOM. This section is about the output file.

The image format that an MRI scan is returning is a Digital Imaging and Communications in Medicine (DICOM) file. This is the standard for handling, saving, printing and transferring medical images and other information regarding this image/patient. The DICOM-file consists of a header and a data set containing the data that you want to save of the patient. The header is the part of the filer where the information about the patient is stored. Example for information that can be stored in the header is:

- Patients name, age, sex, weight and height.
- Acquisitions made from the doctor about what he thought of the image
- Image dimensions
- Matrix size
- Colour space
- Or other...

The reason why the header and the image data is in the same file is because it should be hard to separate the two of them. You do not want to lose the patient info connected to the given image. When converting the file from DICOM to another type of file you will lose the patient data. This is in our case necessary because the patient data is classified and unauthorized people are not allowed to work with this type of data. The DICOM format stores any kind of data and images are stored as pixel data from an MRI session. Since the DICOM file only stores one image each file, you need to store as many files as there are layers in the z-direction of the image. All the images are stored with the file type "*.dmc". If there is an MRI scan producing a 255*255*255 pixel image, you end up with 255 "DCM" files containing one image each file.

2.4 Machine Learning

Machine learning is all about understanding and extracting knowledge from data. It is a researching field within Artificial Intelligence and has a lot of branches branching

out of it. The use of applications helped by a machine learning algorithm is to find in all our lives. Everything from picking recommended music, videos, movies, and other entertainment, to unlocking our phones and in general makes tasks in our daily lives easier. The idea of machine learning is not a new field of research. AI was first discussed by Alan Turing in the 1950s and this raised the question: Could a computer go beyond "what we know how to order it to perform" and learn on its own how to perform a specified task? The use of machine learning has a big influence on how data-driven research is done today. It helps science understand problems that not necessarily are recognized by humans such as finding particles, analyzing DNA and recognizing cancer.

2.5 Neural Networks

Neural networks are inspired by our human brain. The human brain contains roughly 86 billion neurons and the connections between these neurons are what make our mind so powerful. Our mind controls everything from your body, thoughts, memories and more. The concept of using a neural network-model in computing is not a new field of research [XXX]. It was first presented for over 60 years ago, but the technology at that time had no possibilities to apply such a model. In this section, we are going to look closer into how neural networks function and particularly how the convolution neural networks function (CNN). CNN is the computer equal to human eyes and is copying how shapes, color, and shades are processed in our brains. There are many kinds of neural networks and some of the easiest to imagine is a feed-forward neural network. The neural network is containing the following parts:

2.5.1 Input Layer

The input-layer works as an entrance to the hidden layers. This entrance is tailored for the data to fit into the neural network. The data often needs to be processed to fit

into an input layer and when the data is in the same size and dimensions as the input layer, the data is ready to be processed in the neural network.

2.5.2 Hidden Layers

The hidden layers are what makes up the neural network, and the "magic" it does. In the hidden layer, we are talking about numbers of hidden layers. This is the number of layers with a decided amount of neurons in the depth of the network. In between the layers are connections/weights which separates the neurons from the previous layer to the next layer.

2.5.3 Neurons

Neurons are connected to the previous layer and receives values from the previous neurons which are weight and then pushed through an activation function. The value of the current neuron can be updated by changing the value of the weights coming into the neuron.

2.5.4 Activation Function

As mentioned in the previous section, there is a function used to calculate the new value of a neuron. This is used to normalize data and keep the neurons under control. A neuron can often make conclusions by looking at noise (data that are misleading or confusing) and keep evolving around this misunderstanding. There are therefore a lot of different types of activation functions depending on what type of problem you want to solve. E.g. the activation function named ReLu is converting all negative values to 0 and all positive values are still the same.

2.5.5 Output layer

The output layer is similar to the input layer. It is the exit of the neural network and this is where you get your results. Therefore the output layer has to be in the same dimensions and size as the expected results. This means that the labeled known training results should have the same size and dimensions as the output layer. So if you are classifying an image to be either a “Dog” or a “Cat” the output should be an approximation of what the network thinks it is e.g. [0.95, 0.02]. This is showing that this neural network thinks it is a “Dog”.

2.5.6 Data Set

Picking better data

To solve the problem, it is useful to pick the data that suits the problem best. This includes removing data that are not useful or generating/add more data that we want and supports our conclusion.

Pre-Processing and Scaling

In neural networks and convolutional neural networks, the algorithms are sensitive to the scale of the data. Therefore, it is common practice to scale and pre-process the data set before giving it to the network. The data should be adjusted in such a way that it suits and optimizes the performance of the network. In a lot of cases, the data is too detailed, the contents can be irrelevant for the problem that you are trying to solve, or just not fit for the network to function optimally. In this case scale the data to fit the network by e.g. reducing dimensions, changing the data type, or fitting it into another shape that suits better. A very important thing to remember when you are changing the training data is that the test data should be treated in the same way. If it is necessary to keep the original data types and scale for the results, this can be achieved by undoing the processes done before the neural network.

2.5.7 Multi-Class Classification

In a lot of cases, it is enough to calculate single/binary values like is it a dog in this image or not. But in cases where there are more than two classes to classify a problem appear. The problem is an instance of multi-class classification. When this is the case and every single point should be classified into only one of the categories, meaning that one of the categories is 1 and the rest is 0, you have an instance of single-label, multi-class classification.

2.5.8 One-Hot Encoding

When classifying multiple classes or solving categorical variables (single-label, multi-class classification) such as the one mentioned in the above chapter. The most common way to represent the data is using the one-hot encoding also known as categorical encoding. The reason behind this is to simplify the variables into true-false, or 0 -1 values to represent what class it is representing. E.g it is possible to classify an image and tell if there is a cat or a dog in it.

This can also be done per pixel in the entire image.

2.6 Convolutional Neural Network

The human vision is an advanced sensor. Within fractions of a second, we can identify all objects in our line of sight without even thinking about it. We can name the object, tell how far away it is, calculate its movement and predict where the object is going to be in the near future. Our eyes take raw pixels of colors and transform them into more primitive shapes like shadows, lines, curves, and other shapes. And then it gets process into one object and classified. The human vision is the motivation for the convolutional neural network and the difference between how an ordinary neural network and the CNN processes data.

Convolutional Neural Network (CNN) is a deep learning, feed-forward artificial neural network. CNN is good at recognizing objects in images and that is why CNN is a good solution for recognizing different tissues in an MRI image. The input to a CNN can be a 2D image, 3D image, sound or any data where the columns and rows are closely related to each other. This is because of the unique properties of the CNN algorithm.

CNN is a multiple layer neural network. This means that it uses several layers in depth to produce a result. The following sections will describe the layers in detail.

2.6.1 Convolution Layer

The convolutional layers perform a 2D convolution that compares a squared filter over the entire images. The filter could be an edge, line, dark spot or light spot, that scores the parts of the image to how similar that part of the image is to the filter. After applying the filters the convolutional layer gives the result to the next layer.

The first convolutional layers look at simple filters, but the filters get more complex in the following layers and could represent entire objects instead of lines or spots. In the following figure you can see how the different layers have different complexity in the filters and can recognize objects like faces and cars in the last layer.

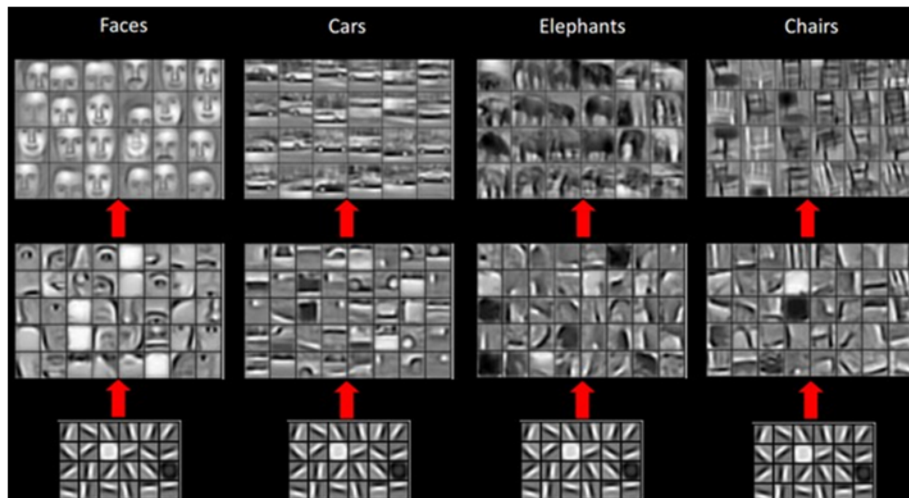

Figure 2.3: *CNN Convolution Layers*

2.6.2 Pooling Layer

The convolutional networks often include local or global pooling layers. This down-sampled the output and uses, for example, max pooling or average pooling to define the down-sampled value. The pooling layer also makes the filter less sensitive to the position. This is good because you do not want to focus on an exact position but what is appearing in the image.

2.6.3 Fully Connected Layer

The final layer in a CNN is named Fully Connected Layer. This is where every value that has gone through all the filters earlier gets a vote to find what the answer is going to be. We take all the values and list them up in a single fully-connected table where it weights against all the objects. The one object with the strongest average is what the CNN will return as its result.

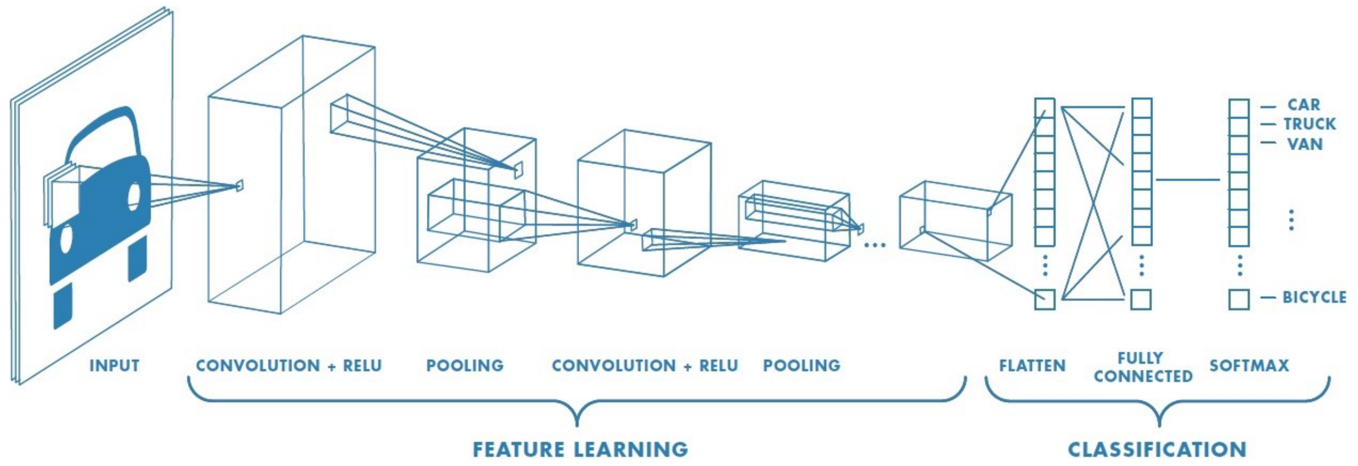

Figure 2.4: *Full CNN network*

Chapter 3

Methodology

3.1 Software **24**
3.1.1 Keras and TensorFlow 24
3.1.2 Computer Specifications 24
3.2 Data-set **25**
3.2.1 Raw Data 25
3.2.2 Mask/Label Annotation 27
3.3 Program **28**
3.3.1 UNet 30
3.3.2 Activation Function 31
3.3.3 Loss Function 32
3.4 Evaluation **33**
3.4.1 Confusion Matrix 33
3.4.2 Measure performance 35

3.1 Software

The code for this was built using Python programming language, the Keras library with TensorFlow as backend.

3.1.1 Keras and TensorFlow

Keras is a widely used high-level neural network API. The Keras library is a user-friendly, modular and extensible library that allows fast prototyping. It is an excellent library for developing CNNs and runs seamlessly on both CPU and GPU. It is an open-source library that is written in python which makes it easy to understand and debug if any problems should occur.

Keras is using TensorFlow as a backend. TensorFlow is an open-source platform used for machine learning.

The U-Net model is programmed such that it is running on both of the graphic-cards on the computer.

3.1.2 Computer Specifications

The computer used to perform these tasks is a computer owned by NTNU. This computer has good hardware and is a good environment to perform a task such as this. The hardware specifications is as follows:

- Intel(R) Core(TM) i9-9960X CPU 3.10GHz (Cores)
- 64GB Ram
- SSD 1024 GB
- 2 x NVIDIA GeForce RTX 2080 with 8GB of VRAM

3.2 Data-set

The data-set is from 20 different knees, but one is not used because it occurred problems when reading the file. There are used several different imaging techniques used to image the knee. This is done by the MRI scanner and uses the different properties of the tissue to get different values from the individual tissues. In this data-set, we are using three different imaging techniques. This makes it more likely to distinguish the tissues from one another. The different imaging types that are used are T1, PD, and FS. The images are anonymous and delivered by MR-Klinikken Møre for research on this topic. The images are from the right knees from 20 individual subjects with an estimated healthy knee. The subjects are all adults in the age between 27 and 64. And it is a 50% distribution between males and females. The average age is 46.

All images are inside a preferable quality. This means that the images are in between the good conditions where the image is sharp and with good contrast and some images that did not qualify within these parameters were thrown away.

The labels or also known as the ground through is segmented by qualified persons in MR-Knlinkken Møre with help of tools that are widely used to segment such images.

3.2.1 Raw Data

There are 20 test images delivered with the raw data images from T1, PD, and FS as well as the ground truth/label saved as Nifty files.

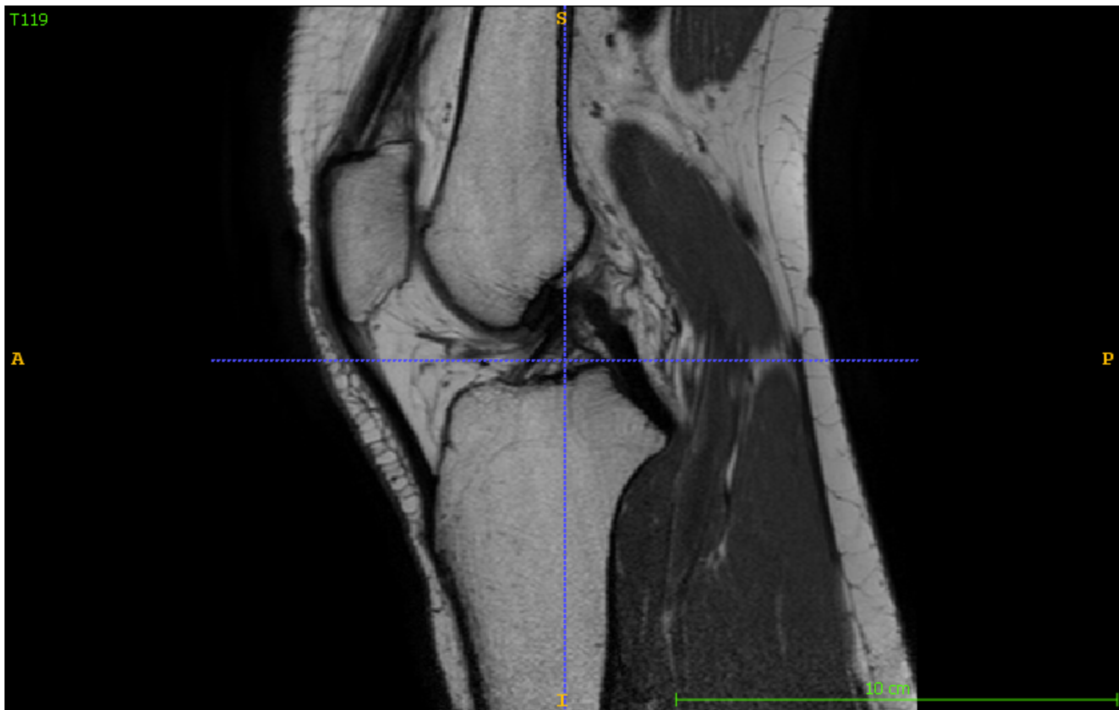

Figure 3.1: *T1 raw image*

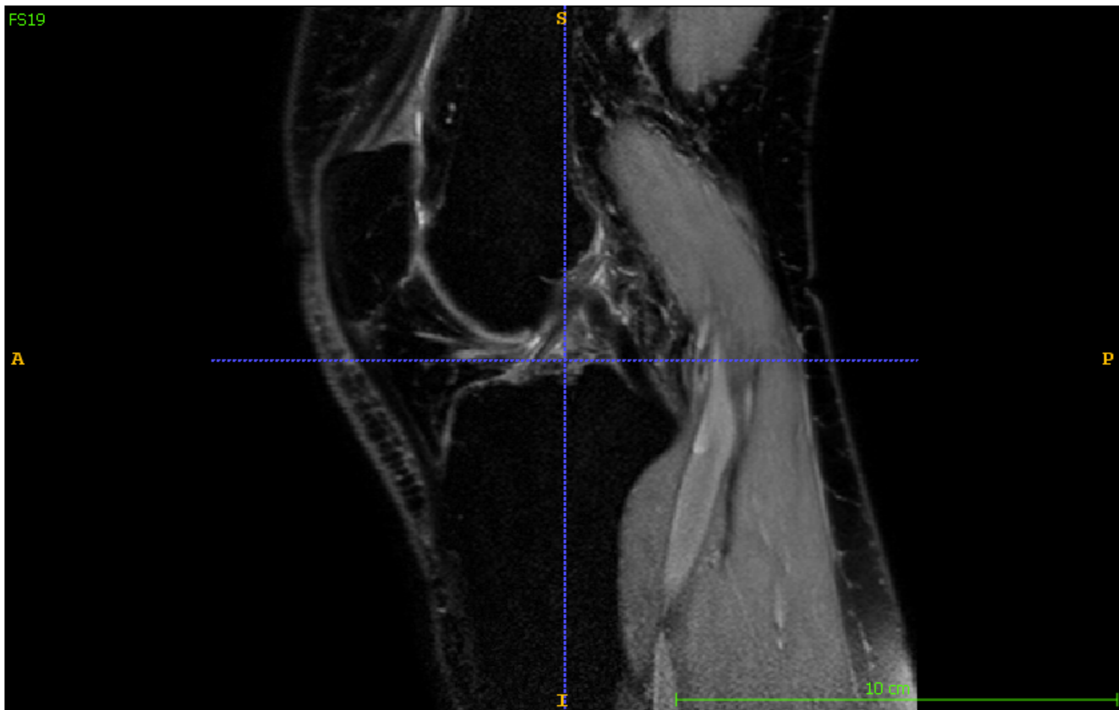

Figure 3.2: *FS raw image*

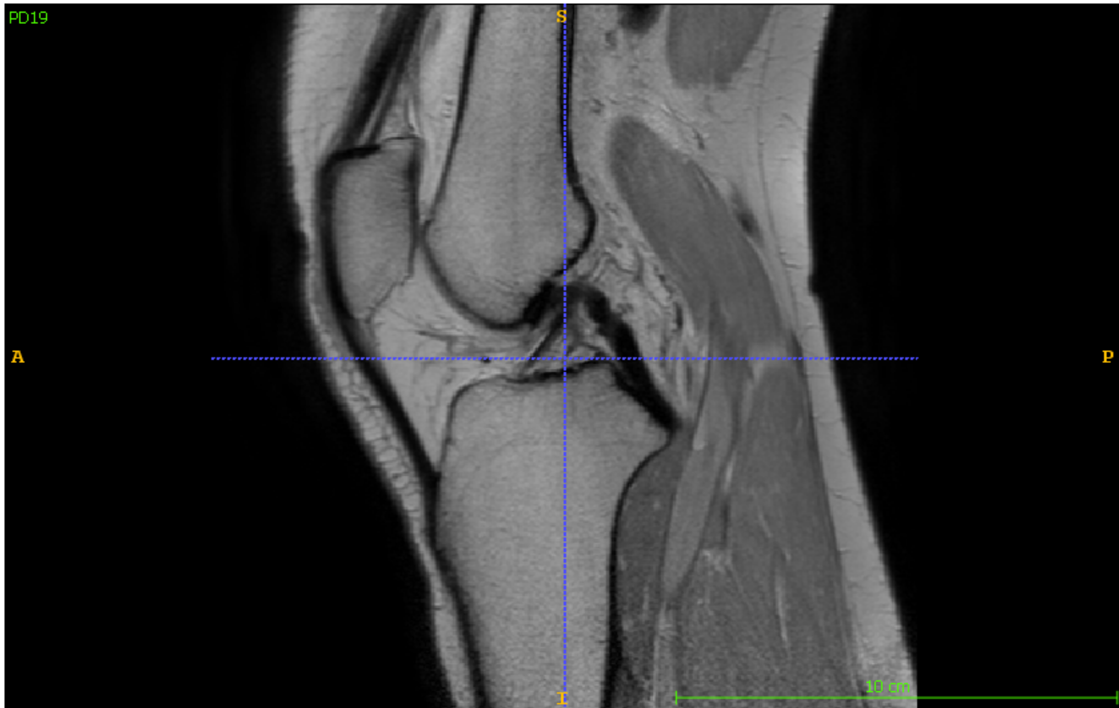

Figure 3.3: *PD raw image*

3.2.2 Mask/Label Annotation

The ground truth also named mask or label is a 3-dimensional image in the same shape/size as the raw image data. The difference is that it only contains values of the pixels that are classified as some of the tissues that are in the knee. This means that if a pixel is outside the knee joint or an unspecified tissue in the knee, it has the value 0. Depending on how many tissues are labeled in the image, the range of the pixels is integers reaching from 0 to the number of tissues classified in the image.

Since the labeled image contains an integer with the range from 0 to the number of tissues that are classified, the labeled image needs to be converted into one-hot encoding to fit the input values of the model [as discussed in the section about one hot encoding]. This is done while loading the labels and all the ground truth (labels) are stored as this while the program is processing the data. The labels used in this data-set is the following:

- Empty/not classified - Value 0 - Black
- Bone - Value 1 - Dark gray
- PCL - Value 2 - Light gray
- ACL - Value 3 - White

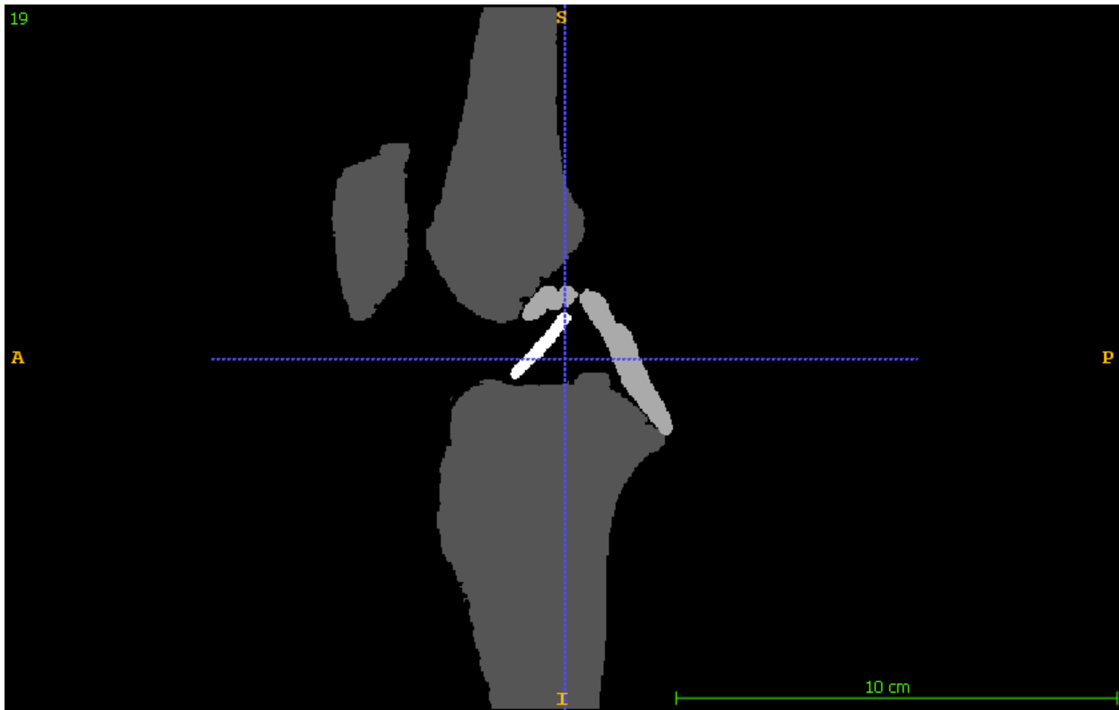

Figure 3.4: *Label Image*

3.3 Program

The following figure shows the flow of the python script and how it manipulates the data to get the predicted result. You can find the source code as an appendix XX.

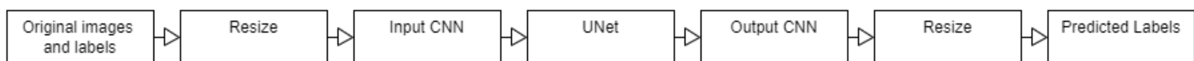

Figure 3.5: *Program Flow*

The data we are using are 3-dimensional images of human knee joints. The original size

of the images is 400x400x275 where each pixel represents a space of 0.XX cm in real life. The original image is scaled down to make the images fit into the convolutional neural network. This is done by resizing the image to the size 256x256x256. The resizing of the image also applies to the labeled image, to make sure that the input and output pixels are the same amount. The input format is then given a new dimension and then the dimensions look like one of the following: 256x256x256x1, 256x256x256x2, 256x256x256x3. This depends on how many of the raw data image types we want the CNN to process.

Since the labeled data set is labeled with 3 labels. That is not included no label. This means that the output data has 4 different values as a possible value. In the labels, there are classified bones, and the ACL and PCL (see. Medicinal section). The same one-hot encoding is applied on the labeled data-set to fit it into the CNN.

The following table show how the size and dimensions are changed throughout the programming script.

Table 3.1: *Dimension Size*

Image Type	**Raw image dimensions**	**Input dimensions**	**Output dimensions**	**Saved classified**
T1	1 x 400x400x275	256x256x256x1	256x256x256x4	400x400x275x4
PD	1 x 400x400x275	256x256x256x1	256x256x256x4	400x400x275x4
FS	1 x 400x400x275	256x256x256x1	256x256x256x4	400x400x275x4
PD, T1	2 x 400x400x275	256x256x256x2	256x256x256x4	400x400x275x4
PD, FS	2 x 400x400x275	256x256x256x2	256x256x256x4	400x400x275x4
FS, T1	2 x 400x400x275	256x256x256x2	256x256x256x4	400x400x275x4
PD, T1, FS	3 x 400x400x275	256x256x256x3	256x256x256x4	400x400x275x4

3.3.1 UNet

The Convolutional neural network is a U-Net [8] inspired network and it is built using the Keras library. The figure [xxx] shows how the network structure is. It shows 5 levels of layers where the first layer is the input layer. After the input layer, there is a convolutional layer that is discussed in section 2.6.1. All the convolutional layers are followed up by a batch normalization layers. After two convolutional layers with a corresponding batch normalization layer, there is a max-pooling layer. The next level starts with a dropout layer where the first dropout layer is halved to prevent the model from forgetting any good finds in the first layer. Then it repeats the same pattern as described above. This repeats in all 5 levels and then the 5 decoding levels starts, where the goal is to reach the original image size with the labels instead of pixel values. The first level is started with an up transposed and followed by a merging with the concatenation and followed by a new dropout layer. Then it is new layers with two times of convolution and batch normalization. Next, there is a new transposed layer to double the size of the image and the layer is done. It is then repeated until it is scaled up to the same size as the input and then the output is generated. All max-pooling layers and transpose layers contain a size of 2x2, so it gets half the size each time the max-pooling is applied and double size when the transpose layer is applied.

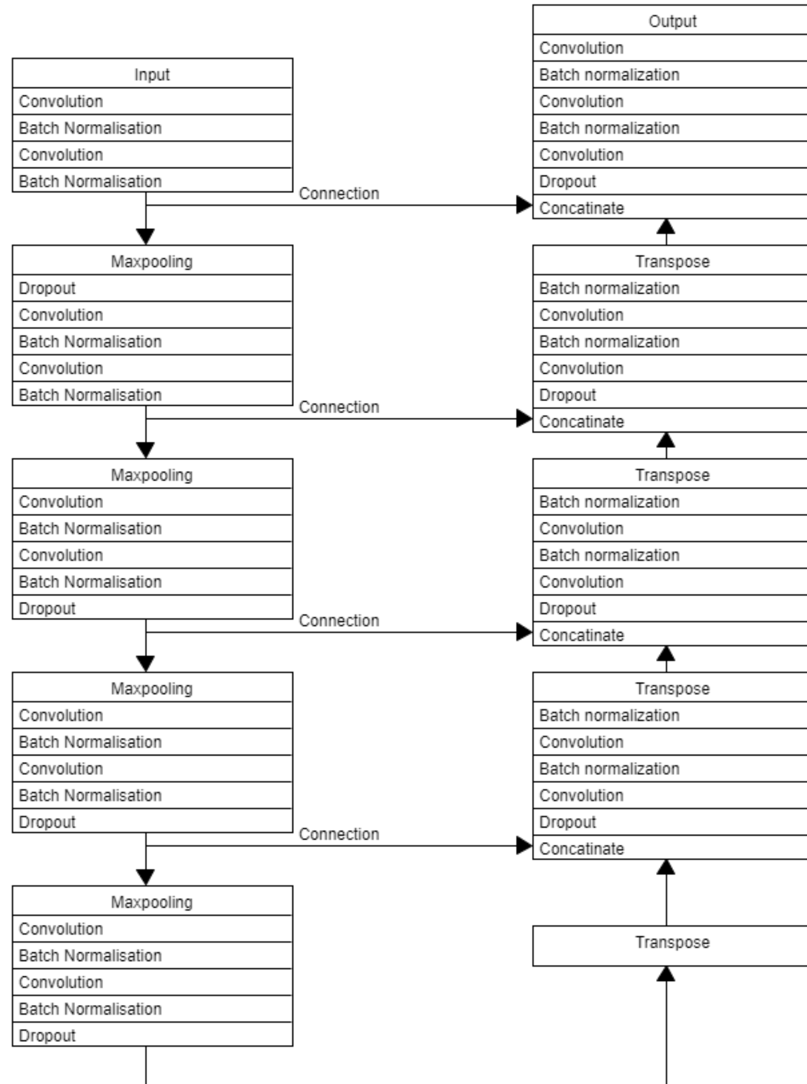

Figure 3.6: *CNN structure. See detailed summary in Apendix B CNN Summary.*

3.3.2 Activation Function

The convolutional layers are using ReLu as an activation function and the padding is the same to keep the same size of the image.

The last activation function is a sigmoid function. It could also have been a softmax function but this is solved later by applying an argmax function (A function that gives the class with highest probability a new value of 1 and the other classes a value of 0.)

to the output and then get the highest scored output. Sigmoid and softmax are pretty similar but softmax also applies the argmax. Because it was used sigmodi in this last layer the argmax function has to be applied after the output has been generated.

3.3.3 Loss Function

Categorical Cross-entropy

To calculate the loss with categorical cross-entropy you take the true value and multiplies this with the log value of the prediction. Say that we are looking at only one class and one pixel where the true values is 1. And for example, the prediction is right. This gives us the following loss: -1*log(1)= 0. Say that the prediction is only 50% probability on the right class, the loss will be as follows: -1*log(0.5) = 0.693 The loss is added together for all the classes in all the pixels of the image and this is how you get the total loss in the image.

Weighed Categorical Cross-entropy

For an unbalanced data-set such as this one, the original categorical cross-entropy usually updates the weights so that the most common classes are prioritized and the lower weighed classes often get ignored. This is because a class with lesser pixels in the ground truth does not add together to a big enough loss to make a difference.

To solve this there is added a weight to the calculated loss. If there is a class that has the true value of 1 and it is a class that needs to be weighed up with a value of 2 and the predicted value is 0.5. This gives us the following loss: -1*log(0.5) * 2 = 1.386

To predict the given problem and the unbalanced classes in our data-set, it is calculated the original categorical cross-entropy, and two versions of weighted categorical cross-entropy. The following values are the weights used on the loss function, where the first value represents value and class 0 and continuing left up to value and class 3:

$$WCC = [1.0, 1.05, 6.9, 11.8]$$

$$WCC2 = [1.0, 1.1, 12.0, 19.7]$$

3.4 Evaluation

Sometimes when evaluation the results, it is not enough to look at the accuracy. This especially true when you are looking at a multi-class classification problem. Like other problems such as binary classification, it is possible to evaluate the results in multiple ways and then calculate the average of all classes. This chapter will tell more about the evaluation methods used in this master theses

3.4.1 Confusion Matrix

Confusion matrices are often used for the evaluation of a model. When solving a binary classification (see chapter about binary classification) this could be illustrated pretty easily. In the next table you can see how binary classification results can be distributed four different categories. This is True Positive, True Negative, False Positive and False Negative, also known as TP, TN, FP, and FN. True positive and true negative is when you are predicting the correct value of that given point either if it is a positive value or a negative value. When you hit a false positive the model gives a false alarm a predicts this point to be something that it is not. And a false negative is when it is predicting noting but missing on a positive value.

Table 3.2: *Confusion Matrix*

True Predictions	**Predicted True**	**Predicted False**
True	TP	FN
False	FP	TN

Metrics for Multi-Class Classification

In short steps, the multi-class classification is similar to the binary classification, but you take the average over all the classes given in the problem. When classes are imbalanced as it often is. The accuracy is not any good type of evaluating the network. An example of an imbalanced data-set where the model has classified 90 % to be class A, 6 % class B, and 4 class C. In this case, the accuracy does not mean anything at all. You could score 85 % accuracy, but this does not tell if you are correctly predicting anything from class B or C.

Multi-Class Confusion Matrix

The confusion matrix is, in this case, the only way to control that the model is correctly predicting the separate classes. Considering the same three classes (A, B, and C) as the model example above. You can see in the following table that the model is predicting the classes with a bad rate. Although the accuracy is tolerably high.

Table 3.3: *Confusion Matrix with values.*

True Values	**Predicted A**	**Predicted B**	**Predicted C**
A	180	0	0
B	2	6	2
C	0	3	7

This confusion matrix shows in detail how the predictions are made by the model. Known classes is representing the label data that is known to be the correct answer to this problem. The Predicted Class is what the result from the predictions the model has made. The diagonal elements in this matrices are the correctly predicted classes and the other values show how the model misses and tells you a lot about what issues this model is having with the problem it is trying to solve.

Table 3.4: *Confusion Matrix Notation*

True Values	**Predicted A**	**Predicted B**	**Predicted C**
A	TP_a	e_{ab}	e_{ac}
B	e_{ba}	TP_b	e_{bc}
C	e_{ca}	e_{cb}	TP_c

The notations used for values taken from the confusion matrix when evaluation multi-class classification is as the table shown above.

3.4.2 Measure performance

There are several methods used to calculate the performance of a model. The reason behind having several methods is to understand and measure the performance of the model in more detail.

Accuracy

Accuracy is the simplest way to measure performance and often used to evaluate the model while training. It is total predictions that are true divided by total data point in the entire data set.

$$Accuracy = (TP)/(total_data_predicted)$$

Precision

Precision also named positive predictive value is measuring the accuracy within a given class. This means that if the model has predicted this class, what is the accuracy for tat this is a correct prediction.

Binary classification

$$Precision = tp/(tp + fp)$$

Multi-class classification

$$Precision(a) = TP_a/(TP_a + e_b a + e_c a) = 180/180 + 2 + 0 = 0.989$$

To calculate the total precision for the entire model, you calculate the average precision of each class.

Recall

Given that this is a value that should be positive, what is the accuracy for the model to find and predict the true value.

$$Recall = sensitivity = tp/(tp + fn)$$

$$Recall(a) = TP_a/(TP_a + e_a b + e_a c) = 180/(180 + 0 + 0) = 1.0$$

F1-score

Say that you have two different models or algorithms scoring differently on accuracy, precision, and recall. Where one is better in precision and the other is better on recall. f1-score helps you classify which one of these models/algorithms is the better one. The mathematical formula for f1-score is:

$$f1 - score = 2 * (precision * recall)/(precision + recall)$$

When scoring a multi-class classification there are two possible ways to calculate the f1-score, which is macro-f1 and weighted-f1.

Macro f1-score

Macro f1-score is the simplest way to calculate the f1-score for multi-class classification. This takes the average of the f1-scores for each class. And the mathematical formula

is as follows [7]:

$$Macro - F1 = (F1_a + F1_b + F1_c)/3$$

The macro f1-score does not consider that the classes are differently weighted. this tells us a lot about how it scores on each class but does not consider the total image.

Weighted f1-score

The weighted f1-score takes to a count that the classes may not be equally weighted. This means that the F1-score gets weighted by how many percentage of this class is represented in the data-set.

The mathematical formula for this is as follows:

$$weighted - f1 = (F1_a * true_a + F1_b * true_b + F1_c * true_c)/(total_predictions)$$

Chapter 4

Results

4.1 Loss Function 39

4.1.1 Categorical Cross-entropy 40

4.1.2 Weighted Categorical Cross-entropy 41

4.1.3 Weighted Categorical Cross-entropy 2nd Version 42

4.2 Compared results of different input 43

4.2.1 One Input 43

4.2.2 Two inputs 47

4.2.3 Three inputs 51

4.3 Accuracy over Iterations 53

4.1 Loss Function

As discussed earlier there are used 3 different loss functions used to calculate the loss on CNN. See section 3.3.3. Early findings in the project show that the original categorical cross-entropy was not able to score any of the lower weighted classes.

As showing in the upcoming tables, there is a significant difference in the performance between the functions used. Especially when training the model for lesser iterations, you will see that the categorical cross-entropy has ignored the lower weighted classes and does not hit any pixels in those classes.

4.1.1 Categorical Cross-entropy

The evaluations of the categorical cross-entropy with one T1 input and 10 iterations and all three inputs with 1000 iterations are shown in the tables below. The categorical cross-entropy has greater problems when scoring the lower weighted classes. But the scores in the first two classes seems to be a good representation of the ground truth. The macro F1-score shows that the average over the F1-scores is pretty bad. Although the model that used 1000 iterations of training scores way better, and are evening out the gap from the other two loss-functions. It still scores the worst on the smallest class value 3.

Table 4.1: *CC (T1 10)*

Class	**True Positive**	**Recall**	**Precision**	**F-Score**
0	39847412	0.997962	0.9975	0.998
1	3969431	0.98535043	0.9799	0.983
2	0	0	Nan	Nan
3	0	0	Nan	Nan

Note: Macro F1-score: 0.495077424, Weighted F1-Score: 0.995356337.

Table 4.2: *CC (PDFST1 1000)*

Class	**True Positive**	**Recall**	**Precision**	**F-Score**
0	39826899	0.997448257	0.998432467	0.997940119
1	3974149	0.986521602	0.979427857	0.982961931
2	31011	0.82651919	0.763216184	0.793607329
3	2361	0.449971412	0.191670726	0.26883006

Note: Macro F1-score: 0.76083486, Weighted F1-Score: 0.996307596.

4.1.2 Weighted Categorical Cross-entropy

The evaluations of the weighted categorical cross-entropy is shown in the tables below. The tables are showing one T1 input with 10 iterations of training and all three inputs with 1000 iterations of training. As the macro f1-score shows that the weight helps the training find the lower weighed classes early in the training.

Table 4.3: *WCC (T1 10)*

Class	True Positive	Recall	Precision	F-Score
0	39789051	0.9965	0.999	0.9977
1	3987212	0.9898	0.972	0.981
2	34902	0.9302	0.649	0.7649
3	3904	0.744	0.313	0.4402

Note: Macro F1-score: 0.795939725, Weighted F1-Score: 0.995899592.

Table 4.4: *WCC (PDFST1 1000)*

Class	True Positive	Recall	Precision	F-Score
0	39821311	0.997308308	0.998482868	0.997895242
1	3974473	0.98660203	0.978368421	0.982467975
2	32523	0.866817697	0.701349953	0.775354027
3	2467	0.470173432	0.260699567	0.335418083

Note: Macro F1-score: 0.772783832, Weighted F1-Score: 0.996214022.

4.1.3 Weighted Categorical Cross-entropy 2nd Version

The evaluations of the weighted categorical cross-entropy version two is shown in the tables below. The tables are showing one T1 input with 10 iterations of training and all three inputs with 1000 iterations of training.

Table 4.5: *WCC2 (T1 10)*

Class	True Positive	Recall	Precision	F-Score
0	39820557	0.99729	0.9982	0.99773
1	3956394	0.98211	0.9834	0.98277
2	35418	0.94398	0.5659	0.70763
3	4858	0.92586	0.2354	0.37535

Note: Macro F1-score: 0.765870872, Weighted F1-Score: 0.996036625.

Table 4.6: *WCC2 (PDFST1 1000)*

Class	True Positive	Recall	Precision	F-Score
0	39819616	0.997265857	0.998468482	0.997866807
1	3974554	0.986622137	0.977598792	0.982089738
2	31909	0.850453092	0.687560602	0.760380798
3	2311	0.440442157	0.31796918	0.36931682

Note: Macro F1-score: 0.777413541, Weighted F1-Score: 0.996144863.

4.2 Compared results of different input

In this section, we will show the results of the performance of the different imaging types. As discussed earlier in section 3.2 there are 3 different imaging techniques delivered in this data-set and we will look deeper into how these images performed. And maybe discover if one of them are containing more information or a better choice when you want to segment the classes.

4.2.1 One Input

In this section, we will look at how the different image techniques performers when training the model. Comparing the three different imaging types FS, PD, and T1 to see which one of these images performs the best. We will look into how it scores in the evaluation methods discussed in section 3.4.2. There seems to be a big gap between how the different imaging types scores in the early stages. As shown in the tables below there are some imaging types that are not able to detect some of the classes. Although FS and T1 seem to do pretty decent there are some big differences with the PD images.

PD

The figure shows how the PD image type has evolved the weights in the network with the weighed categorical cross-entropy after just 10 iterations of training. You can see that this input has problems detecting the lower weighted classes such as the ACL and PCL values 2 and 3. In the table, you can see the evaluation of the entire 3-dimensional image. The calculated average F-scores is bad because it does not score the third class at all. The average F1-score and weighted F1-score is as follows:

$$Average = 0.629306106$$

$$Weighted = 0.993824558$$

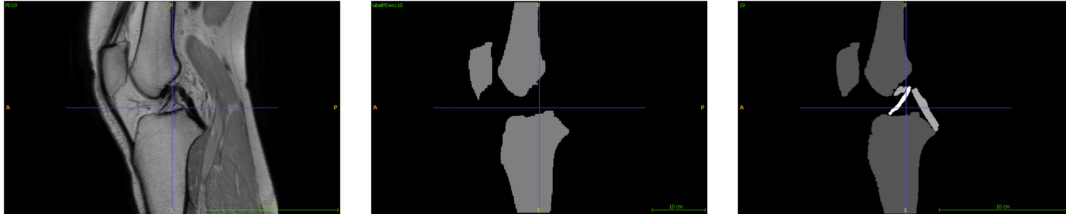

Figure 4.1: *Raw PD, Predicted output, Ground truth*

Table 4.7: *PD 10 iterations*

Class	True Positive	Recall	Precision	F-Score
0	39777731	0.996216865	0.997099323	0.996657899
1	3935749	0.97698939	0.965525878	0.971223809
2	18446	0.49163113	0.609261461	0.544161897
3	0	0	Nan	Nan

FS

The following figure shows how the FS image type has evolved the weights in the network with the weighed categorical cross-entropy after just 10 iterations of training. It manages to find all the classes but still has some problems classification the lowest weighted classes.

$$Average = 0.714017027$$

$$Weighted = 0.989512527$$

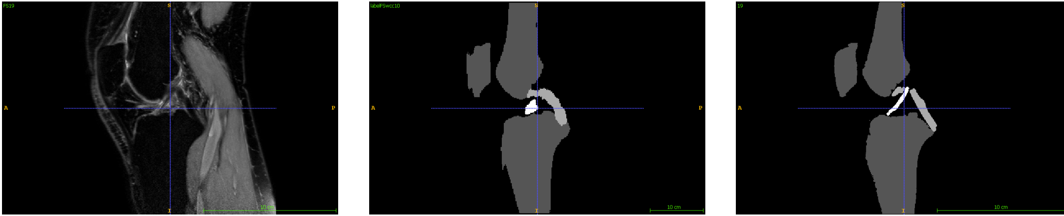

Figure 4.2: *Raw FS, Predicted output, Ground truth*

Table 4.8: *FS 10 iterations*

Class	True Positive	Recall	Precision	F-Score
0	39669131	0.993497023	0.994866632	0.994181355
1	3825669	0.949663716	0.944144318	0.946895974
2	31276	0.83358209	0.611851243	0.705709805
3	2963	0.56470364	0.128440765	0.209280972

T1

The following figure shows how the T1 image type has evolved the weights in the network with the weighed categorical cross-entropy after just 10 iterations of training. The average Macro F1-score is the best score in all of the results with only 10 iterations.

$$Average = 0.795939725$$

$$Weighted = 0.995899592$$

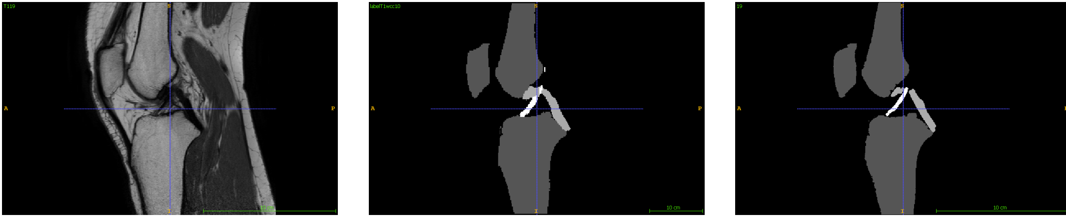

Figure 4.3: *Raw T1, Predicted output, Ground truths*

Table 4.9: *T1 10 iterations*

Class	True Positive	Recall	Precision	F-Score
0	39789051	0.99650037	0.998898098	0.997697793
1	3987212	0.989764291	0.97229485	0.9809518
2	34902	0.930223881	0.649436195	0.764874756
3	3904	0.744044216	0.312595084	0.440234551

4.2.2 Two inputs

By combining two of the images to the input layers. The evaluations start looking better. There are still some of the imaging types that perform better than the others. The different combinations of 2 image types are shown in the below sections where the tables show the scoring of evaluation methods discussed in section 3.4.2.

FS and PD

There are some problems when classifying lower classes in data containing two images where the PD image type is present. In this case, none of the lower classes were detected.

$$Average = 0.492114709$$

$$Weighted = 0.993381669$$

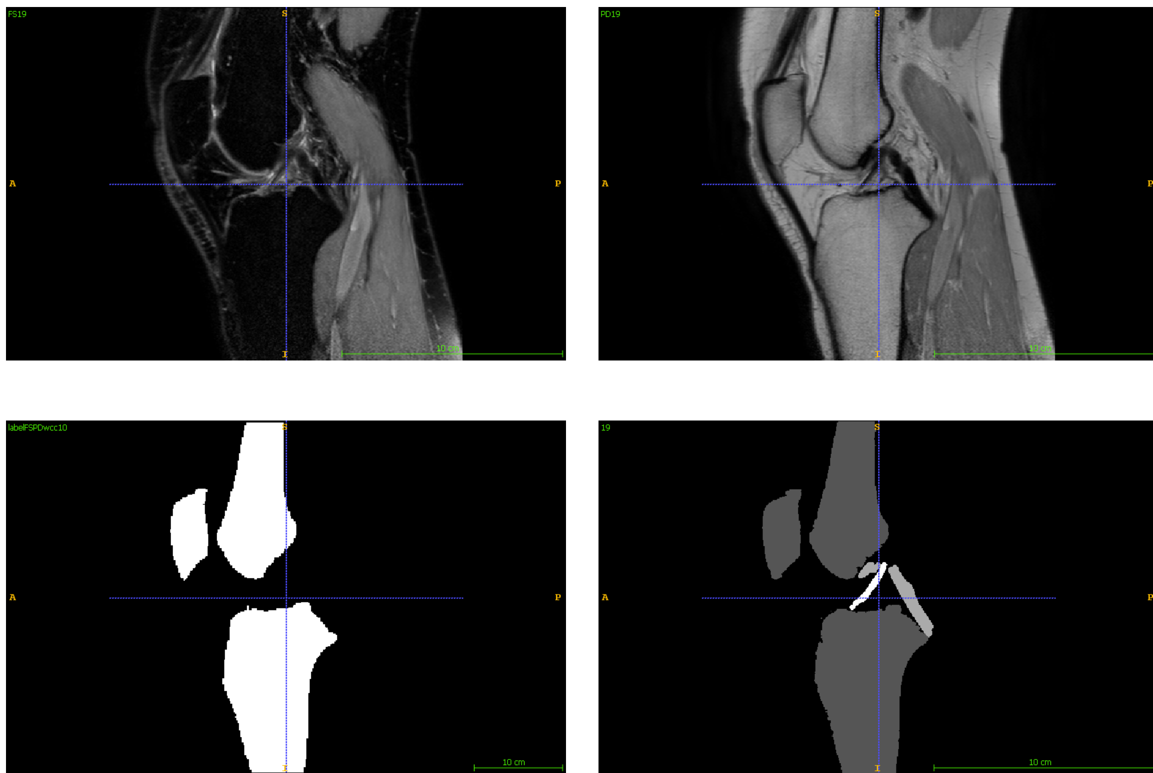

Figure 4.4: *Raw FS, Raw PD, Predicted output, Ground truths*

Table 4.10: *FS, PD 10*

Class	True Positive	Recall	Precision	F-Score
0	39796975	0.996698823	0.996539267	0.996619038
1	3932679	0.976227309	0.967491549	0.971839799
2	0	0	Nan	Nan
3	0	0	Nan	Nan

FS and T1

This combination scores the best of the inputs containing two image types.

$$Average = 0.756220155$$

$$Weighted = 0.995905556$$

Table 4.11: *FS, T1 10*

Class	True Positive	Recall	Precision	F-Score
0	39795367	0.996658551	0.99865021	0.997653386
1	3975564	0.986872854	0.977049596	0.981936658
2	36510	0.973081023	0.5853682	0.730996786
3	3893	0.74194778	0.199375192	0.314293788

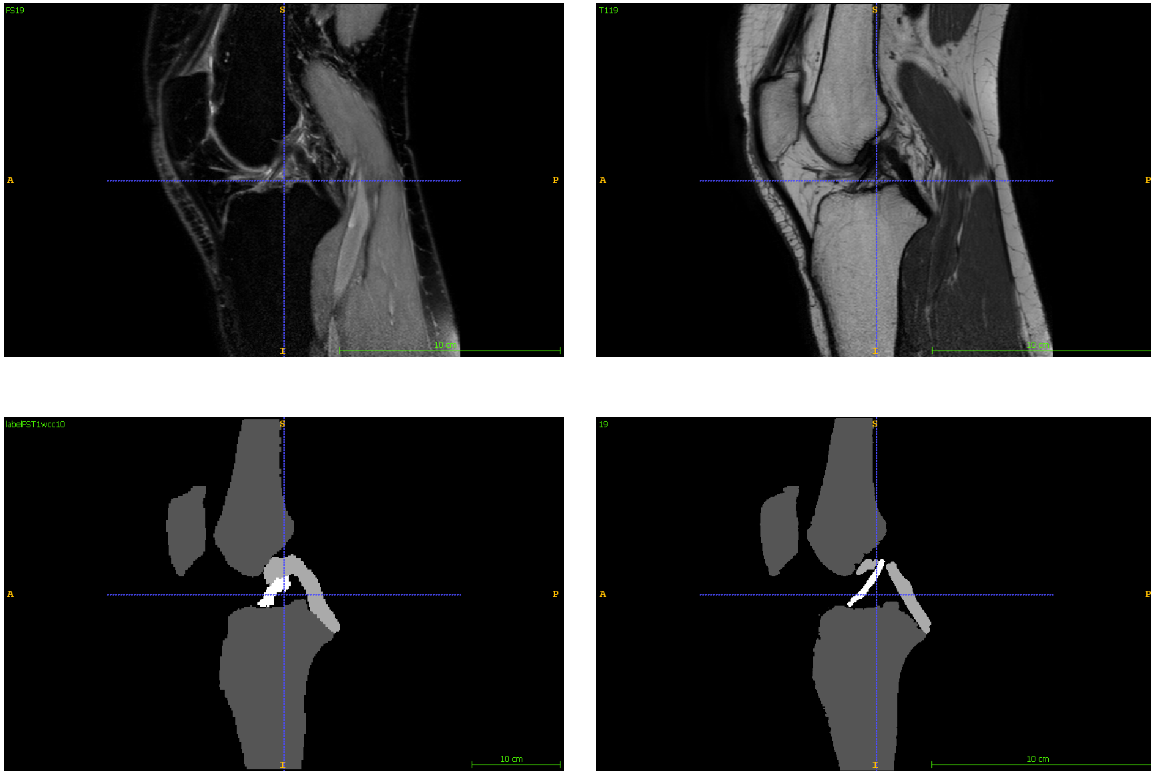

Figure 4.5: *Raw FS, Raw T1, Predicted output, Ground truths*

PD and T1

There are some still some problems when classifying lower classes in data containing two images where the PD image type is present. Combined with the T1 image type the PD scores a better then whit the FS.

$$Average = 0.68750736$$

$$Weighted = 0.996072251$$

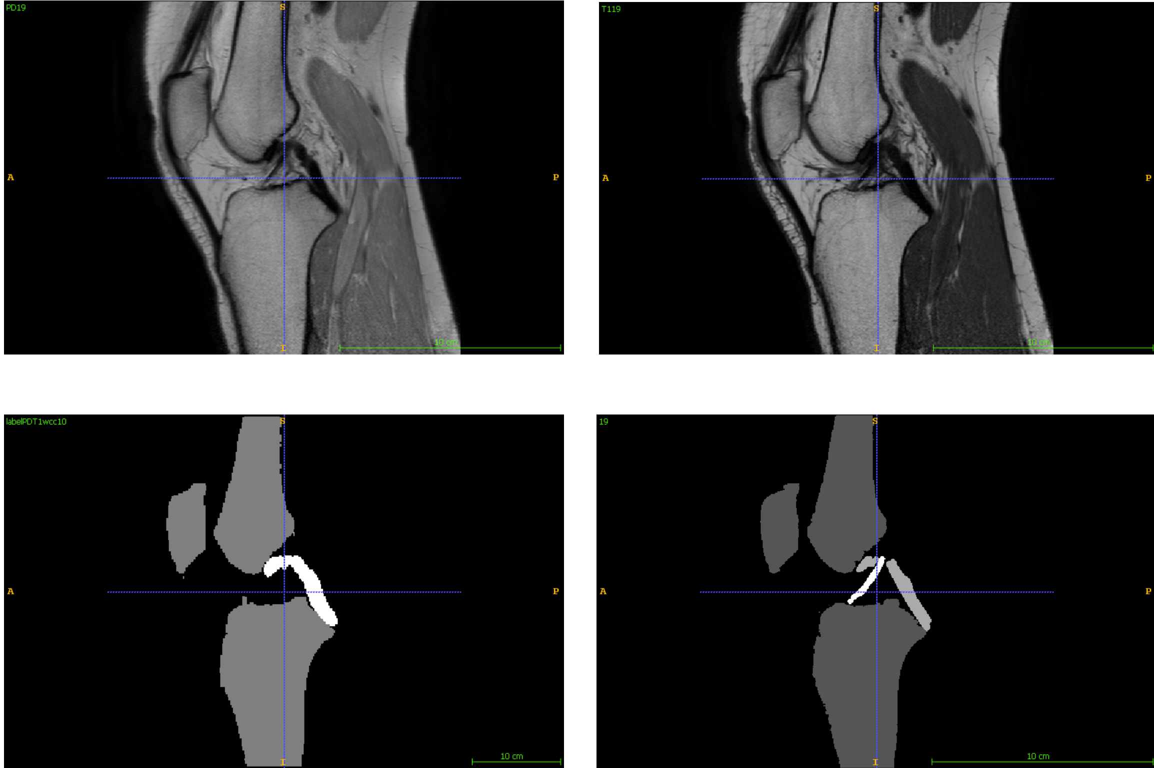

Figure 4.6: *Raw PD, Raw FS, Predicted output, Ground truths*

Table 4.12: *PD, T1 10*

Class	True Positive	Recall	Precision	F-Score
0	39813380	0.997109679	0.99858805	0.997848317
1	3979720	0.987904517	0.975905694	0.981868449
2	34614	0.922547974	0.661203438	0.770312674
3	0	0	Nan	Nan

4.2.3 Three inputs

This is the most common way to combine the image-types to represent all three inputs and make the network look at all the image-types in one. Since this image contains the most of information there is easy to conclude that this is the best input for the network. Although there seems to be some imperfections also with this input.

$$Average = 0.798264327$$

$$Weighted = 0.995903632$$

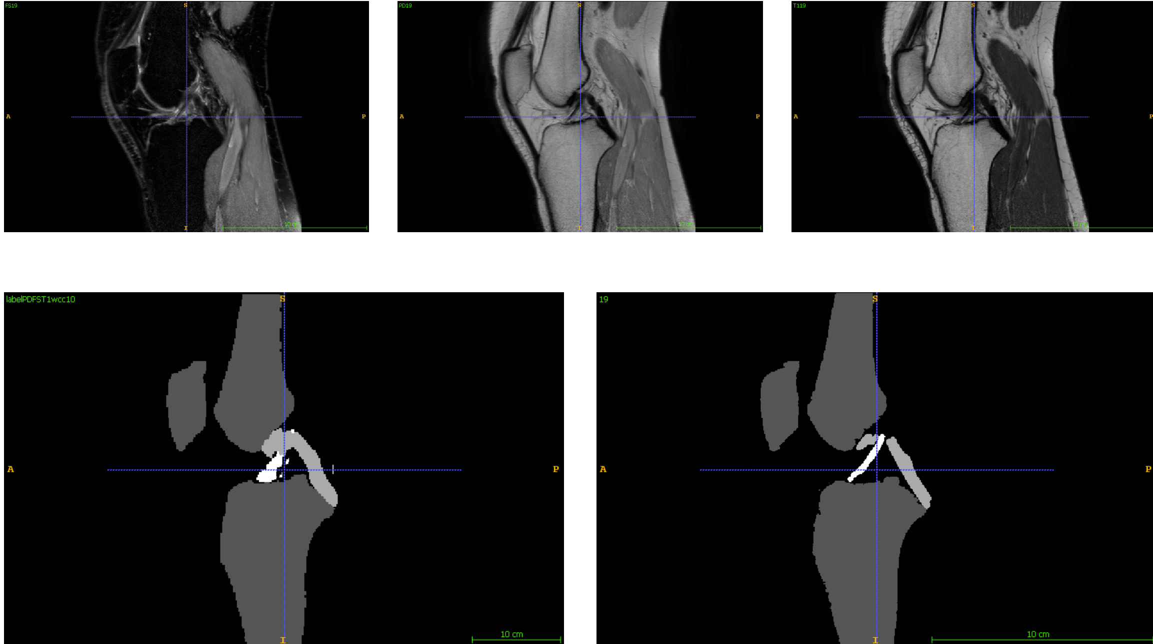

Figure 4.7: *Raw FS, Raw PD, Raw T1, Predicted output, Ground truths*

Table 4.13: *PD, FS, T1, 10*

Class	True Positive	Recall	Precision	F-Score
0	39789588	0.996513818	0.998841493	0.997676298
1	3984817	0.989169769	0.973869288	0.981459901
2	34937	0.931156716	0.628216424	0.75026038
3	3901	0.74347246	0.230664617	0.352091701

4.3 Accuracy over Iterations

Show the results of the classification as a 3D image of the entire image. The model reduces its loss within a small number of iterations. It seems to get too detailed and starts introducing some errors. Although it scores better in f1-score the visualization looks more wrong.

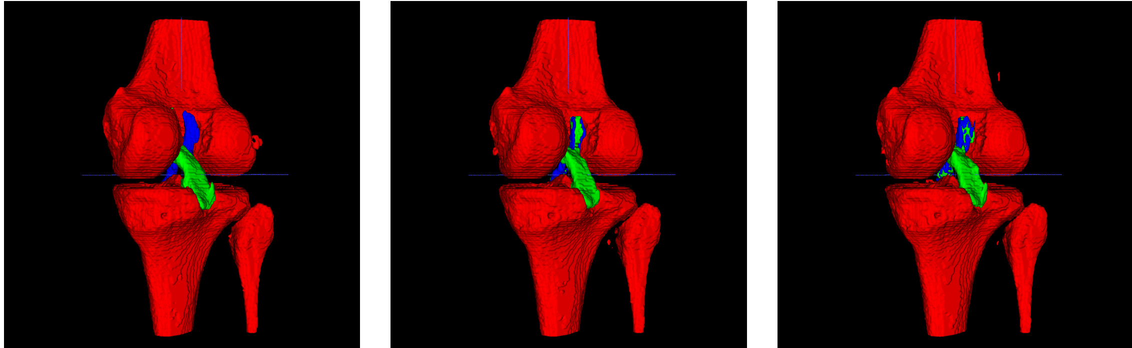

Figure 4.8: *10, 100, 1000*

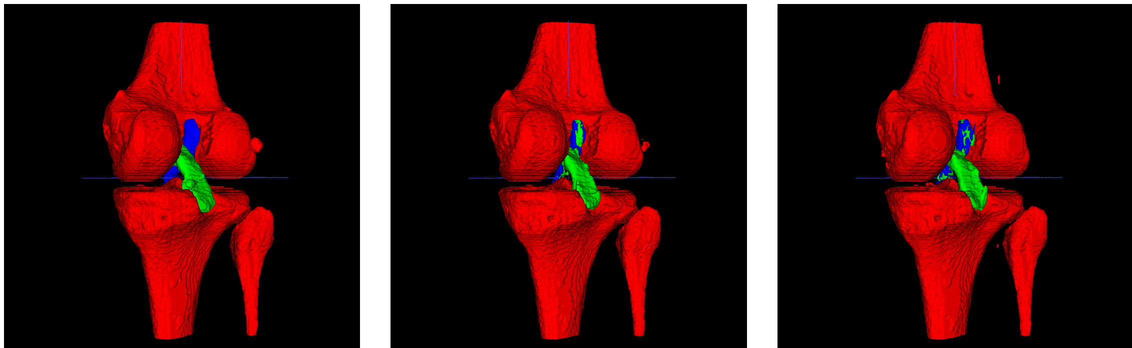

Figure 4.9: *Caption for this figure with two images*

Chapter 5

Discussion

5.1 Loss-Function with Weights 55

5.2 Best Input to Build a Model 55

5.3 Best Model 56

5.4 Model error vs. Human error 58

5.1 Loss-Function with Weights

The categorical cross-entropy (CC) scores great values on the knee, looks realistic with no obvious fails to recognize visually on its own and not comparing it to the labeled. Overall the categorical cross-entropy maybe the best-looking knee segmentation. But it does not score a good blue value (class 3). It does a god segmentation elsewhere and does not detect tissue remote from where it should be. When training the model for fewer iterations there is a significant issue in the CC's way to identify the lower classes and it regularly does not score any positives in class 2 and 3.

The first weighted categorical cross-entropy seems to be performing somewhere in between CC and the second version of weighted categorical cross-entropy. It does not achieve the best results at any of the classes but always makes a decent detection of the small classes and a satisfying segmentation on detecting the bone tissue values 1.

The second is the best model to predict value 3 and does that one properly. It does, however, find points outside what seems to be a reasonable area to detect the low weighted classes. This loss function had the best performance of them all when training for only 10 iterations. It detected all classes early, already after a small number of iterations with all inputs.

5.2 Best Input to Build a Model

In the earliest stages of training, the PD seems to contain less information than the other two imaging types. It has problems detecting the ACL and PCL, also when combined with other imaging types.

When training with only 10 or 100 iterations, this seems to be the best results combining all of the outputs. Although when training over 1000 iterations the differences seams to even out and there is no big differentiation anymore.

It seems that the 3 inputs introduces more noise than information to the model when

training the model for many iterations. This can bee seen when looking at the best models from 1000 iterations. When studying this network the best evaluation seems to come from the segmentation from models created only whit T1 images.

5.3 Best Model

There are a couple ways to explain which model is the best. The first is an overall accuracy. This is model is made by combining the categorical cross-entropy and T1 image over 1000 iterations. This gives a clear and realistic outcome that does score as high as 99,638 % accuracy. Witch is a satisfying accuracy to have on a CNN. Out of a total of 44 000 000 pixels, there are 43 840 759 correct classified pixels. Only missing on 159 241 of the pixels. But there is a problem with this model as well. Although it scores high in accuracy it has a problem detecting the smaller classes. The smallest class contains 5247 pixels, and this model only scores 3050 correct classifications in this class.

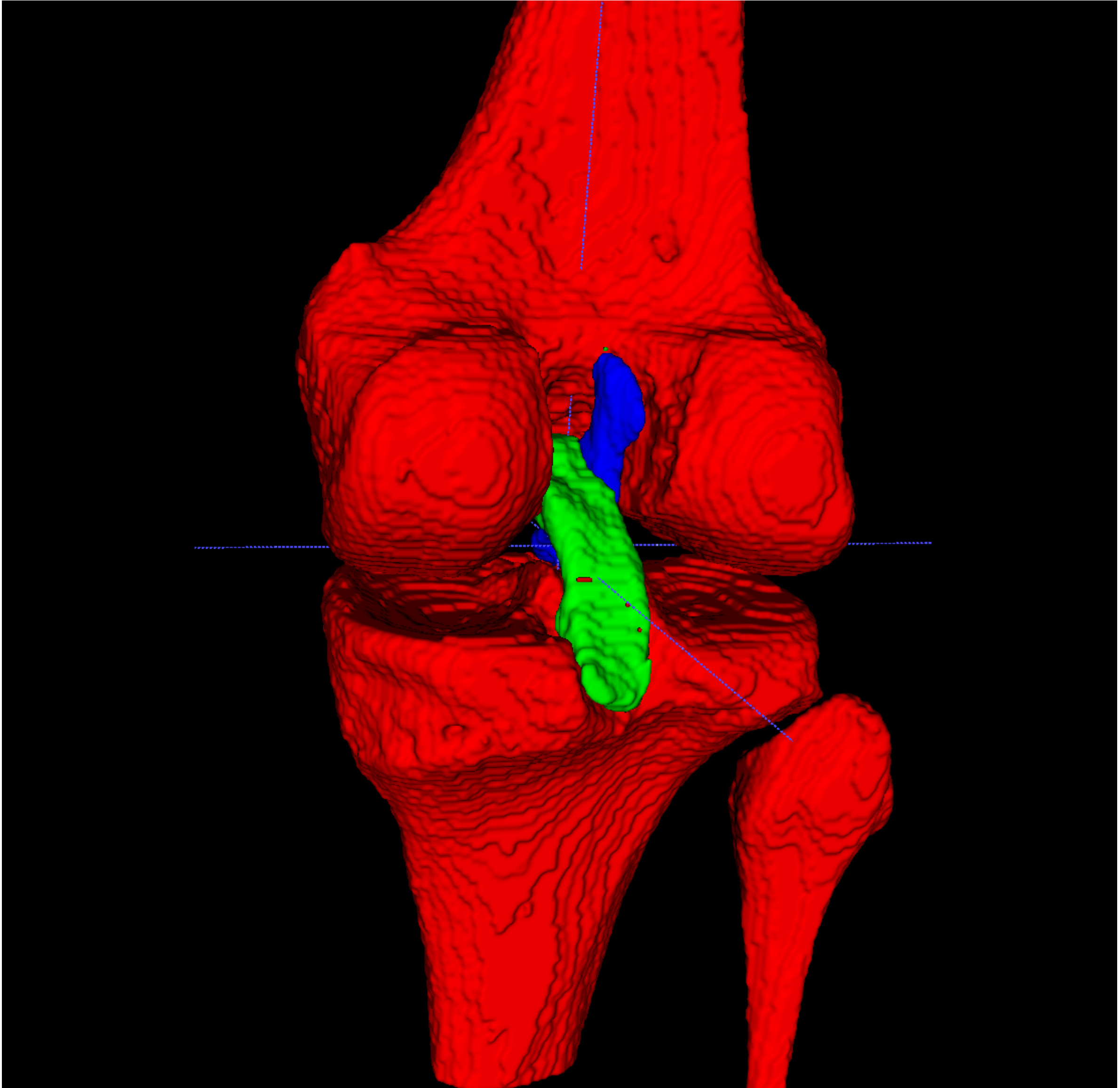

Figure 5.1: *Best Result CC*

The second way to score the best model is to look at the model that scores the highest accuracy on each of the classes combined. This model is created using one of the inputs. It uses T1 images and have the second weighted cross-entropy trained over 1000 iterations. It gives a realistic outcome with some missed boundaries in the ACL and PCL. It still has a good accuracy with 99,60 % and scoring 43 825 612 out of 44 000 000 leaving only 174 388 missed pixels. This model scores way better on the lower weighted classes and hits on 4436 of the total 5247 pixels in the class with value 3.

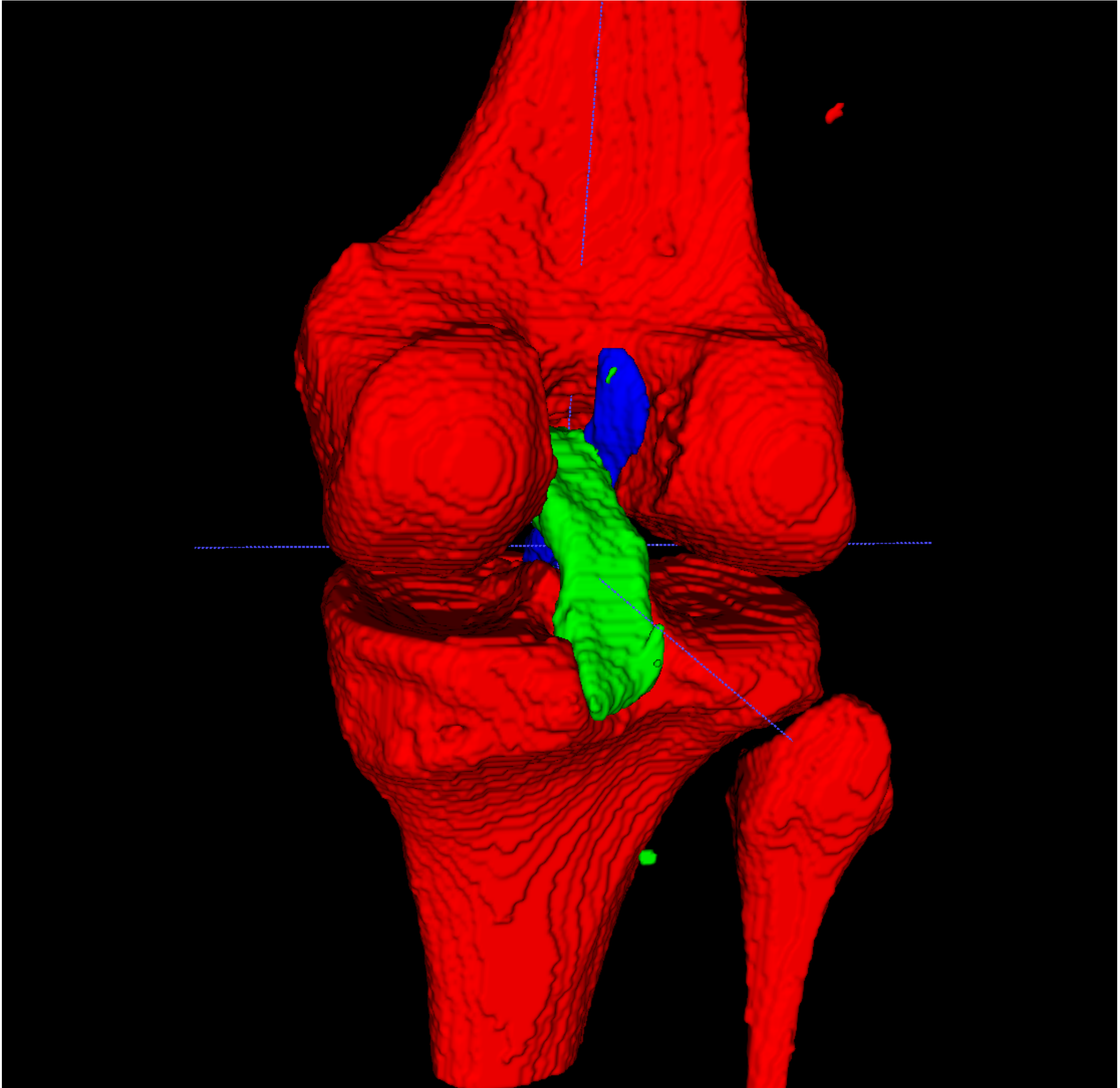

Figure 5.2: *Best Result WCC2*

5.4 Model error vs. Human error

We are scoring the accuracy of the predictions and on most of the cases, it is better then 99%. This brings up the question: How accurate are these predictions against the actual true values and does the old techniques also have some error when classifying the ground truth.

The ACL and PCL are often drawn by hand by a human that is qualified to know where it starts, ends and where it should go in between these points. But the pixels that are marked are pixels that often are dark and other pixels surrounding the ligaments are also dark. This makes it hard for a human to see the difference and it can introduce errors to the labeled ground truth. By looking at the ground truth you can also detect some unrealistic pixels and shapes in the way the ligaments look. After viewing some of the results and applying the new predicted segmentation's on top of the original image, the output from this model seems to be as possible as the ground truth.

Chapter 6

Conclusion

6.1 Conclusion and Future Work

The segmentation of bodyparts are a valuable tool to both understand and examen the human body. You can add a segmented bodyparts in a 3-dimensional world, look at it from different angles and discover something you could not have seen from a 2-dimensional view.

The designed python script named MartiNet is a Convolutional Neural Network (CNN) using Keras with TensorFlow as backend. It is a CNN created to make detailed segmentation of the knee joint. It is a U-Net inspired network that can classify each pixel in an MRI scan as a tissue.

This CNN is a tool performs this task very well. It can make a precise prediction and outputs a segmentation that is over 99% accurate and looks as real as the ground truth. It is probably not a perfect segmentation and it will require few corrections. These corrections can be performed by a professional in a short amount of time. This is a significant decrease in workload taken into a count that the CNN segments the entire knee in only seconds compared with the old method that used hours to complete.

The biggest issue was when detecting small tissues that do not have a lot of pixels in the images. This was solved by adding a weighted loss function. This had a big impact on how it performed on the smaller classes an this could be considered further when segmenting more labels on the knee. The loss function named WCC2 sowed a promising result on classifying these lower weighted classes. For further work, there should be a task finding a ratio or an optimal description for the weights which could help a lot when classifying more labels.

When looking at the different input images and how they performed, there were no significant differences. The finding in this thesis shows results that can point in the direction of using only one input image (T1) when classifying just three labels as the bone, PCL and ACL.

This could also work on several other joints on the human body and a CNN like this

can almost without any modifications be trained to classify other bodyparts.

As further work, this CNN could detect more labels. The rate of learning in this CNN was fast and it had an accuracy of better than 95 % after only a few iterations. This shows that there is space for more complex problems and it is possible to add more labels to the segmentation.

Bibliography

[1] JOHN P. GOLDBLATT, MD, and JOHN C. RICHMOND, MD (2003): ANATOMY AND BIOMECHANICS OF THE KNEE

[2] https://www.healthpages.org/anatomy-function/knee-joint-structure-function-problems/ [Downloaded 04.12.18]

[3] Zhaoye Zhou, Gengyan Zhao, Richard Kijowski, Fang Liu (2018): Deep convolutional neural network for segmentation of kneejoint anatomy

[4] Eli Gibson, Wenqi Li, Carole Sudre, Lucas Fidon, Dzhoshkun I. Shakir, Guotai Wang, Zach Eaton-Rosen, Robert Gray, Tom Doel, Yipeng Hu, Tom Whyntie, Parashkev Nachev, Marc Modat, Dean C. Barratt, Sébastien Ourselin. M. Jorge Cardoso, Tom Vercauteren (2018): NiftyNet: a deep-learning platform for medical imaging

[5] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015) : TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems

[6] `https://radiopaedia.org/articles/mri-sequences-overview` [Downloaded 03.12.2018]

[7] `https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1`[Downloaded 01.09.2019]

[8] Olaf Ronneberger, Philipp Fischer, Thomas Brox (2015): U-Net: Convolutional Networks for Biomedical Image Segmentation

[9] FRANÇOIS CHOLLET (2018): Deep Learning with Python

[10] Matthew Kirk (2017):Thoughtful Machine Learning with Python

Appendix A

All Results

cc

FS		10	cc					
	0	1	2	3	support	Recall	Precision	f-score
0	39703283	225504	0	0	39928787	0.99435235	0.994514392	0.994433362
1	183125	3845321	0	0	4028446	0.95454202	0.943007843	0.948739879
2	30632	6888	0	0	37520	0	0	0
3	5241	6	0	0	5247	0	0	0
						0.48722359	0.484380559	0.48579331
						0.989741	0.988832034	0.989283301

PD		10	cc					
	0	1	2	3	support	Recall	Precision	f-score
0	39769067	147818	11902	0	39928787	0.99599988	0.997355704	0.99667733
1	93922	3934524	0	0	4028446	0.9766853	0.96356618	0.970081388
2	8831	931	27758	0	37520	0.73981876	0.657788099	0.696396091
3	2687	21	2539	0	5247	0	0	0
						0.67812599	0.654677496	0.665788702
						0.9938943	0.993853591	0.993867411

T1		10	cc					
	0	1	2	3	support	Recall	Precision	f-score
0	39847412	81375	0	0	39928787	0.997962	0.997456958	0.997709413
1	59015	3969431	0	0	4028446	0.98535043	0.979865445	0.982600284
2	37336	184	0	0	37520	0	0	0
3	5241	6	0	0	5247	0	0	0
						0.49582811	0.494330601	0.495077424
						0.99583734	0.994876851	0.995356337

FS,PD		10	cc					
	0	1	2	3	support	Recall	Precision	f-score
0	39780681	148106	0	0	39928787	0.99629075	0.996938789	0.996614662
1	80119	3948327	0	0	4028446	0.98011169	0.963672224	0.971822437
2	36803	717	0	0	37520	0	0	0
3	5229	18	0	0	5247	0	0	0
						0.49410061	0.490152753	0.492109275
						0.99384109	0.992924047	0.993376109

FS,T1		10	cc					
	0	1	2	3	support	Recall	Precision	f-score
0	39858451	70336	0	0	39928787	0.99823846	0.997300778	0.997769401
1	65333	3963113	0	0	4028446	0.98378208	0.982507745	0.983144502
2	37304	216	0	0	37520	0	0	0
3	5241	6	0	0	5247	0	0	0
						0.49550514	0.494952131	0.495228476
						0.99594464	0.994977039	0.9954606

PD,T1		10	cc					
	0	1	2	3	support	Recall	Precision	f-score
0	39827330	99815	1642	0	39928787	0.99745905	0.998088118	0.997773486
1	47701	3980745	0	0	4028446	0.98815896	0.975031376	0.981551276
2	23794	2119	11607	0	37520	0.30935501	0.847535597	0.453265645
3	4796	5	446	0	5247	0	0	0
						0.57374325	0.705163773	0.608147602
						0.99590186	0.995729742	0.99570495

FS,PD,T1		10	cc					
	0	1	2	3	support	Recall	Precision	f-score
0	39818777	110010	0	0	39928787	0.99724484	0.997809963	0.997527324
1	44850	3983596	0	0	4028446	0.98886667	0.97307385	0.980906699
2	37306	214	0	0	37520	0	0	0
3	5240	7	0	0	5247	0	0	0
						0.49652788	0.492720953	0.494617091
						0.99550848	0.994575385	0.995036039

wcc

FS 10 wcc

	0	1	2	3	support	Recall	Precision	f-score
0	39669131	222629	18391	18636	39928787	0.993497023	0.994866632	0.994181355
1	200707	3825669	600	1470	4028446	0.949663716	0.944144318	0.946895974
2	2552	3692	31276	0	37520	0.83358209	0.611851243	0.705709805
3	1428	6	850	2963	5247	0.56470364	0.128440765	0.209280972
						0.835361617	0.669825739	0.714017027
						0.989296341	0.989792791	0.989512527

PD 10 wcc

	0	1	2	3	support	Recall	Precision	f-score
0	39777731	139826	11230	0	39928787	0.996216865	0.997099323	0.996657899
1	92616	3935749	81	0	4028446	0.97698939	0.965525878	0.971223809
2	18382	692	18446	0	37520	0.49163113	0.609261461	0.544161897
3	4720	8	519	0	5247	0	0	0
						0.616209346	0.642971666	0.628010901
						0.993907409	0.993758974	0.993824558

T1 10 wcc

	0	1	2	3	support	Recall	Precision	f-score
0	39789051	113445	18025	8266	39928787	0.99650037	0.998898098	0.997697793
1	40891	3987212	339	4	4028446	0.989764291	0.97229485	0.9809518
2	2140	163	34902	315	37520	0.930223881	0.649436195	0.764874756
3	861	6	476	3904	5247	0.744044216	0.312595084	0.440234551
						0.915133189	0.733306057	0.795939725
						0.995797023	0.996082585	0.995899592

FS,PD 10 wcc

	0	1	2	3	support	Recall	Precision	f-score
0	39796975	131812	0	0	39928787	0.996698823	0.996539267	0.996619038
1	95767	3932679	0	0	4028446	0.976227309	0.967491549	0.971839799
2	37225	295	0	0	37520	0	0	0
3	5213	34	0	0	5247	0	0	0
						0.493231533	0.491007704	0.492114709
						0.993855773	0.992911172	0.993381669

FS,T1 10 wcc

	0	1	2	3	support	Recall	Precision	f-score
0	39795367	93188	24736	15496	39928787	0.996658551	0.99865021	0.997653386
1	52676	3975564	191	15	4028446	0.986872854	0.977049596	0.981936658
2	698	190	36510	122	37520	0.973081023	0.5853682	0.730996786
3	414	6	934	3893	5247	0.74194778	0.199375192	0.314293788
						0.924640052	0.690110799	0.756220155
						0.995712136	0.996224822	0.995905556

PD,T1		10	wcc					
	0	1	2	3	support	Recall	Precision	f-score
0	39813380	98102	17305	0	39928787	0.997109679	0.99858805	0.997848317
1	48513	3979720	213	0	4028446	0.987904517	0.975905694	0.981868449
2	2757	149	34614	0	37520	0.922547974	0.661203438	0.770312674
3	5024	5	218	0	5247	0	0	0
						0.726890543	0.658924296	0.68750736
						0.996084409	0.996104575	0.996072251

FS,PD,T1		10	wcc					
	0	1	2	3	support	Recall	Precision	f-score
0	39789588	106659	19705	12835	39928787	0.996513818	0.998841493	0.997676298
1	43528	3984817	92	9	4028446	0.989169769	0.973869288	0.981459901
2	2157	259	34937	167	37520	0.931156716	0.628216424	0.75026038
3	465	2	879	3901	5247	0.74347246	0.230664617	0.352091701
						0.915078191	0.707897955	0.798264327
						0.995755523	0.9961475	0.995903632

FS		10	wcc2					
	0	1	2	3	support	Recall	Precision	f-score
0	39678430	212172	18515	19670	39928787	0.993729912	0.995062584	0.994395802
1	190505	3834083	1747	2111	4028446	0.951752363	0.94689197	0.949315945
2	3755	2866	30845	54	37520	0.822094883	0.598885523	0.692959202
3	2621	3	397	2226	5247	0.424242424	0.092514858	0.151903917
						0.797954896	0.658338734	0.697143716
						0.989672364	0.990206835	0.989910978

PD		10	wcc2					
	0	1	2	3	support	Recall	Precision	f-score
0	39821088	89203	0	18496	39928787	0.997302723	0.995490417	0.996395746
1	142531	3885744	0	171	4028446	0.964576415	0.977504839	0.970997595
2	37030	148	0	342	37520	0	0	0
3	829	71	0	4347	5247	0.828473413	0.186119198	0.303954131
						0.697588138	0.539778613	0.567836868
						0.993435886	0.992898337	0.993138175

T1		10	wcc2					
	0	1	2	3	support	Recall	Precision	f-score
0	39820557	66461	26086	15683	39928787	0.997289424	0.998165827	0.997727433
1	71267	3956394	781	4	4028446	0.982114195	0.983431476	0.982772394
2	1820	189	35418	93	37520	0.943976546	0.565936436	0.70763114
3	85	6	298	4858	5247	0.925862398	0.235391026	0.375352521
						0.962310641	0.695731191	0.765870872
						0.995846068	0.99635728	0.996036625

FS,PD		10	wcc2					
	0	1	2	3	support	Recall	Precision	f-score
0	39689988	200406	24678	13715	39928787	0.994019378	0.998451281	0.9962304
1	58982	3969054	263	147	4028446	0.985256846	0.951771568	0.968224778
2	1740	697	35083	0	37520	0.935047974	0.573786043	0.71116832
3	842	18	1119	3268	5247	0.622832095	0.190776416	0.292085624
						0.884289073	0.678696327	0.741927281
						0.993122568	0.993719053	0.993339279

FS,T1		10	wcc2					
	0	1	2	3	support	Recall	Precision	f-score
0	39769603	111674	30139	17371	39928787	0.996013302	0.998911786	0.997460438
1	42075	3986144	115	112	4028446	0.989499177	0.972686932	0.98102103
2	1018	251	36251	0	37520	0.966178038	0.53496746	0.688639192
3	232	6	1258	3751	5247	0.714884696	0.176650655	0.283297459
						0.916643803	0.670804208	0.73760453
						0.995357932	0.996017081	0.995606814

PD,T1		10	wcc2					
	0	1	2	3	support	Recall	Precision	f-score
0	39741163	116756	38606	32262	39928787	0.995301034	0.998824192	0.997059501
1	46515	3980308	1330	293	4028446	0.988050479	0.971487819	0.979699152
2	185	56	36989	290	37520	0.985847548	0.475602073	0.641652138
3	83	6	848	4310	5247	0.821421765	0.116000538	0.203292298
						0.947655207	0.640478655	0.705425772
						0.994608409	0.995769951	0.995072341

FS,PD,T1		10	wcc2					
	0	1	2	3	support	Recall	Precision	f-score
0	39775864	110154	22720	20049	39928787	0.996170107	0.998921414	0.997543863
1	41659	3986218	485	84	4028446	0.989517546	0.973081108	0.981230501
2	1207	113	36055	145	37520	0.960954158	0.603774533	0.741597762
3	82	6	456	4703	5247	0.896321708	0.18826308	0.311168453
						0.960740879	0.691010034	0.803851939
						0.995519091	0.996121966	0.995750181

T1 100 cc

	0	1	2	3	support	Recall	Precision	f-score
0	39834825	80632	6807	6523	39928787	0.99764676	0.998202401	0.997924503
1	63581	3964825	40	0	4028446	0.984207061	0.979798457	0.981997811
2	6110	923	30431	56	37520	0.811060768	0.802568769	0.806792423
3	2045	192	639	2371	5247	0.451877263	0.264916201	0.334014228
						0.811197963	0.761371457	0.780182241
						0.996192091	0.99626315	0.996224171

PD,T1 100 cc

	0	1	2	3	support	Recall	Precision	f-score
0	39828632	88115	5397	6643	39928787	0.997491659	0.998433636	0.997962425
1	52615	3975822	9	0	4028446	0.986936898	0.978245128	0.982571792
2	7940	301	28950	329	37520	0.771588486	0.836632662	0.802795225
3	1929	1	247	3070	5247	0.585096245	0.305715993	0.401595919
						0.835278322	0.779756855	0.79623134
						0.9962835	0.996364687	0.996315786

FS, T1 100 cc

	0	1	2	3	support	Recall	Precision	f-score
0	39827545	86802	8257	6183	39928787	0.997464436	0.998333985	0.997899021
1	56912	3971504	29	1	4028446	0.985865021	0.978490719	0.982164028
2	6576	499	30223	222	37520	0.805517058	0.782776483	0.793983975
3	2976	1	101	2169	5247	0.413379074	0.252944606	0.31384749
						0.800556397	0.753136448	0.771973628
						0.996169114	0.996244524	0.996202938

PD,FS,T1 100 cc

	0	1	2	3	support	Recall	Precision	f-score
0	39841868	80724	128	6067	39928787	0.997823149	0.997533524	0.997678316
1	58267	3970175	4	0	4028446	0.985535117	0.980019348	0.982769493
2	37009	216	196	99	37520	0.005223881	0.428884026	0.010322037
3	3236	4	129	1878	5247	0.357918811	0.233465937	0.282597246
						0.586625239	0.659975709	0.568341773
						0.995775386	0.995353985	0.99538611

FS 100 cc

	0	1	2	3	support	Recall	Precision	f-score
0	39686334	231360	4946	6147	39928787	0.993927865	0.995086117	0.994506654
1	180667	3847679	29	71	4028446	0.955127362	0.942035039	0.948536025
2	11193	5369	20883	75	37520	0.556583156	0.798218791	0.655852517
3	4117	25	304	801	5247	0.152658662	0.11291232	0.129811198
						0.664574261	0.712063067	0.682176599
						0.989902205	0.989955921	0.989905891

T1		100	wcc					
	0	1	2	3	support	Recall	Precision	f-score
0	39825430	83229	13269	6859	39928787	0.997411467	0.998381749	0.997896372
1	58084	3970016	345	1	4028446	0.985495648	0.979425003	0.982450948
2	5408	164	31866	82	37520	0.849307036	0.684819049	0.758244896
3	1060	6	1052	3129	5247	0.596340766	0.310694072	0.408538974
						0.857138729	0.743329968	0.786782797
						0.996146386	0.996296763	0.996207619

PD,T1		100	wcc					
	0	1	2	3	support	Recall	Precision	f-score
0	39814900	94311	11727	7849	39928787	0.997147747	0.998501805	0.997824316
1	53403	3974876	166	1	4028446	0.986702068	0.976777526	0.981714715
2	4778	183	32469	90	37520	0.865378465	0.724334092	0.788599325
3	1559	7	464	3217	5247	0.613112255	0.288339159	0.392221409
						0.865585134	0.746988146	0.790089941
						0.996033227	0.996194348	0.996098762

FS, T1		100	wcc					
	0	1	2	3	support	Recall	Precision	f-score
0	39810128	99234	13549	5876	39928787	0.997028234	0.998609101	0.997818041
1	49716	3978670	60	0	4028446	0.987643871	0.975570943	0.981570285
2	4337	389	32762	32	37520	0.873187633	0.690584094	0.771224444
3	1396	6	1070	2775	5247	0.528873642	0.319590003	0.398420675
						0.846683345	0.746088535	0.787258361
						0.996007614	0.996156194	0.996065768

PD,FS,T1		100	wcc					
	0	1	2	3	support	Recall	Precision	f-score
0	39822307	84467	15296	6717	39928787	0.997333252	0.998486385	0.997909486
1	54357	3973751	337	1	4028446	0.986422804	0.979139378	0.982767597
2	4957	187	32355	21	37520	0.862340085	0.659767537	0.747573937
3	1053	7	1052	3135	5247	0.597484277	0.317500506	0.414655115
						0.860895105	0.738723452	0.785726534
						0.996171545	0.996345016	0.99624014

FS		100	wcc					
	0	1	2	3	support	Recall	Precision	f-score
0	39670670	232467	13345	12305	39928787	0.993535566	0.995097491	0.994315915
1	186796	3841219	256	175	4028446	0.953523766	0.941836513	0.947644106
2	5748	4735	27017	20	37520	0.720069296	0.662165143	0.689904368
3	2900	14	183	2150	5247	0.409757957	0.146757679	0.216112982
						0.769221646	0.686464207	0.711994343
						0.989569455	0.989836085	0.989690469

T1		100	wcc2					
	0	1	2	3	support	Recall	Precision	f-score
0	39816573	85681	17751	8782	39928787	0.997189647	0.99841759	0.99780324
1	58436	3969809	198	3	4028446	0.985444263	0.978838081	0.98213006
2	3992	140	33343	45	37520	0.888672708	0.637533461	0.74244044
3	678	4	1008	3557	5247	0.677911187	0.287155889	0.4034252
						0.887304451	0.725486255	0.78144974
						0.995983682	0.996232423	0.99607964

PD,T1		100	wcc2					
	0	1	2	3	support	Recall	Precision	f-score
0	39808425	99315	15291	5756	39928787	0.996985583	0.998569538	0.99777693
1	51792	3976521	133	0	4028446	0.987110414	0.97558846	0.98131562
2	4337	181	32989	13	37520	0.87923774	0.661102204	0.75472432
3	897	6	1487	2857	5247	0.54450162	0.331207976	0.41187919
						0.851958839	0.741617045	0.78642401
						0.995927091	0.996098142	0.99599268

FS, T1		100	wcc2					
	0	1	2	3	support	Recall	Precision	f-score
0	39807808	96036	15699	9244	39928787	0.996970131	0.9985613	0.99776508
1	53009	3975313	124	0	4028446	0.986810547	0.976357792	0.98155634
2	3711	219	33271	319	37520	0.886753731	0.671774991	0.7644376
3	634	6	433	4174	5247	0.795502192	0.303850914	0.43973873
						0.91650915	0.737636249	0.79587444
						0.995921955	0.996166941	0.99601557

PD,FS,T1		100	wcc2					
	0	1	2	3	support	Recall	Precision	f-score
0	39811619	96749	15339	5080	39928787	0.997065576	0.998606423	0.9978354
1	49595	3978806	45	0	4028446	0.987677631	0.976209649	0.98191016
2	4998	212	32231	79	37520	0.859035181	0.661094474	0.7471776
3	965	3	1139	3140	5247	0.598437202	0.378358838	0.46360549
						0.860553897	0.753567346	0.79763216
						0.996040818	0.996194102	0.99609991

FS		100	wcc2					
	0	1	2	3	support	Recall	Precision	f-score
0	39669658	234912	15025	9192	39928787	0.993510221	0.994821866	0.99416561
1	197730	3830104	498	114	4028446	0.950764637	0.941020732	0.94586759
2	6400	5140	25961	19	37520	0.691924307	0.612865911	0.65
3	2354	3	876	2014	5247	0.383838384	0.177617074	0.24285542
						0.755009387	0.681581396	0.70822216
						0.98926675	0.989472916	0.98936058

cc

FS,PD,T1	0	1	2	3	support	Recall	Precision	f-score
0	39826899	83182	8889	9817	39928787	0.99744826	0.998432	0.997940119
1	54245	3974149	52	0	4028446	0.9865216	0.979428	0.982961931
2	6088	281	31011	140	37520	0.82651919	0.763216	0.793607329
3	2195	11	680	2361	5247	0.44997141	0.191671	0.26883006
						0.81511512	0.733187	0.76083486
						0.99623682	0.996396	0.996307596

FS, T1	0	1	2	3	support	Recall	Precision	f-score
0	39822537	91080	9205	5965	39928787	0.99733901	0.998106	0.997722209
1	67336	3961046	64	0	4028446	0.98326898	0.977429	0.980340212
2	6331	375	30688	126	37520	0.81791045	0.764658	0.790388008
3	1912	15	176	3144	5247	0.59919954	0.340444	0.434194172
						0.8494295	0.770159	0.80066115
				43817415		0.99585034	0.995935	0.99588679

T1	0	1	2	3	support	Recall	Precision	f-score
0	39840970	73521	8501	5795	39928787	0.99780066	0.998237	0.998018753
1	62198	3966179	68	1	4028446	0.98454317	0.981687	0.983113203
2	6227	446	30560	287	37520	0.81449893	0.776304	0.794943163
3	1941	19	237	3050	5247	0.58128454	0.333954	0.424200278
						0.84453183	0.772546	0.800068849
				43840759		0.99638089	0.996453	0.996412471

wcc

	0	1	2	3	support	Recall	Precision	f-score
0	39821311	87671	12952	6853	39928787	0.99730831	0.998482868	0.99789524
1	53799	3974473	174	0	4028446	0.98660203	0.978368421	0.98246798
2	4661	193	32523	143	37520	0.8668177	0.701349953	0.77535403
3	2046	11	723	2467	5247	0.47017343	0.260699567	0.33541808
						0.83022537	0.734725202	0.77278383
						0.99615395	0.996299924	0.99621402

	0	1	2	3	support	Recall	Precision	f-score
0	39827414	81804	14408	5161	39928787	0.99746116	0.998388161	0.99792444
1	57210	3971015	221	0	4028446	0.98574363	0.979776125	0.98275082
2	5170	157	32178	15	37520	0.8576226	0.670948102	0.75288667
3	1919	6	1152	2170	5247	0.41356966	0.295398857	0.34463591
						0.81359926	0.736127811	0.76954946
						0.99619948	0.996321077	0.99624836

	0	1	2	3	support	Recall	Precision	f-score
0	39822228	84083	15074	7402	39928787	0.99733127	0.998225854	0.99777836
1	63932	3964347	141	26	4028446	0.98408841	0.979197336	0.98163678
2	6386	127	30804	203	37520	0.82100213	0.650587142	0.72592732
3	458	11	1329	3449	5247	0.657328	0.311281588	0.42249035
						0.86493745	0.73482298	0.7819582
						0.99592791	0.996105328	0.99600009

wcc2

	0	1	2	3	support	Recall	Precision	f-score
0	39819616	90841	13400	4930	39928787	0.99726586	0.99846848	0.99786681
1	53742	3974554	150	0	4028446	0.98662214	0.97759879	0.98208974
2	5360	224	31909	27	37520	0.85045309	0.6875606	0.7603808
3	1976	10	950	2311	5247	0.44044216	0.31796918	0.36931682
						0.81869581	0.74539926	0.77741354
						0.99609977	0.99621148	0.99614486

	0	1	2	3	support	Recall	Precision	f-score
0	39808156	96839	15501	8291	39928787	0.99697885	0.99850318	0.99774043
1	51741	3976530	175	0	4028446	0.98711265	0.97615442	0.98160295
2	6188	294	30966	72	37520	0.82531983	0.65461695	0.73012355
3	1746	6	662	2833	5247	0.53992758	0.2530368	0.34458432
						0.83733473	0.72057784	0.76351281
						0.99587466	0.99607489	0.99595686

	0	1	2	3	support	Recall	Precision	f-score
0	39818221	81858	18464	10244	39928787	0.99723092	0.99842119	0.9978257
1	60349	3967726	368	3	4028446	0.98492719	0.97975825	0.98233592
2	2155	109	35229	27	37520	0.93893923	0.6475324	0.76647267
3	461	6	344	4436	5247	0.84543549	0.30156356	0.44455579
						0.94163321	0.73181885	0.79779752
				43825612		0.99603664	0.99633018	0.99614426

Appendix B

CNN Summary

File - C:\Users\Martin\Desktop\net structure.txt

```
 1 Layer (type)                     Output Shape         Param #     Connected to
 2 ==================================================================================================
 3 img (InputLayer)                 (None, 256, 256, 2)  0
 4 __________________________________________________________________________________________________
 5 conv2d_1 (Conv2D)                (None, 256, 256, 16) 304         img[0][0]
 6 __________________________________________________________________________________________________
 7 batch_normalization_1 (BatchNor (None, 256, 256, 16) 64          conv2d_1[0][0]
 8 __________________________________________________________________________________________________
 9 activation_1 (ReLu)              (None, 256, 256, 16) 0           batch_normalization_1[0][0]
10 __________________________________________________________________________________________________
11 conv2d_2 (Conv2D)                (None, 256, 256, 16) 2320        activation_1[0][0]
12 __________________________________________________________________________________________________
13 batch_normalization_2 (BatchNor (None, 256, 256, 16) 64          conv2d_2[0][0]
14 __________________________________________________________________________________________________
15 activation_2 (ReLu)              (None, 256, 256, 16) 0           batch_normalization_2[0][0]
16 __________________________________________________________________________________________________
17 max_pooling2d_1 (MaxPooling2D)   (None, 128, 128, 16) 0           activation_2[0][0]
18 __________________________________________________________________________________________________
19 dropout_1 (Dropout)              (None, 128, 128, 16) 0           max_pooling2d_1[0][0]
20 __________________________________________________________________________________________________
21 conv2d_3 (Conv2D)                (None, 128, 128, 32) 4640        dropout_1[0][0]
22 __________________________________________________________________________________________________
23 batch_normalization_3 (BatchNor (None, 128, 128, 32) 128         conv2d_3[0][0]
24 __________________________________________________________________________________________________
25 activation_3 (Relu)              (None, 128, 128, 32) 0           batch_normalization_3[0][0]
26 __________________________________________________________________________________________________
27 conv2d_4 (Conv2D)                (None, 128, 128, 32) 9248        activation_3[0][0]
28 __________________________________________________________________________________________________
29 batch_normalization_4 (BatchNor (None, 128, 128, 32) 128         conv2d_4[0][0]
30 __________________________________________________________________________________________________
31 activation_4 (ReLu)              (None, 128, 128, 32) 0           batch_normalization_4[0][0]
32 __________________________________________________________________________________________________
33 max_pooling2d_2 (MaxPooling2D)   (None, 64, 64, 32)   0           activation_4[0][0]
34 __________________________________________________________________________________________________
35 dropout_2 (Dropout)              (None, 64, 64, 32)   0           max_pooling2d_2[0][0]
36 __________________________________________________________________________________________________
37 conv2d_5 (Conv2D)                (None, 64, 64, 64)   18496       dropout_2[0][0]
38 __________________________________________________________________________________________________
39 batch_normalization_5 (BatchNor (None, 64, 64, 64)   256         conv2d_5[0][0]
40 __________________________________________________________________________________________________
41 activation_5 (ReLu)              (None, 64, 64, 64)   0           batch_normalization_5[0][0]
42 __________________________________________________________________________________________________
43 conv2d_6 (Conv2D)                (None, 64, 64, 64)   36928       activation_5[0][0]
44 __________________________________________________________________________________________________
45 batch_normalization_6 (BatchNor (None, 64, 64, 64)   256         conv2d_6[0][0]
46 __________________________________________________________________________________________________
47 activation_6 (ReLu)              (None, 64, 64, 64)   0           batch_normalization_6[0][0]
48 __________________________________________________________________________________________________
49 max_pooling2d_3 (MaxPooling2D)   (None, 32, 32, 64)   0           activation_6[0][0]
50 __________________________________________________________________________________________________
51 dropout_3 (Dropout)              (None, 32, 32, 64)   0           max_pooling2d_3[0][0]
52 __________________________________________________________________________________________________
53 conv2d_7 (Conv2D)                (None, 32, 32, 128)  73856       dropout_3[0][0]
54 __________________________________________________________________________________________________
55 batch_normalization_7 (BatchNor (None, 32, 32, 128)  512         conv2d_7[0][0]
56 __________________________________________________________________________________________________
57 activation_7 (ReLu)              (None, 32, 32, 128)  0           batch_normalization_7[0][0]
58 __________________________________________________________________________________________________
59 conv2d_8 (Conv2D)                (None, 32, 32, 128)  147584      activation_7[0][0]
60 __________________________________________________________________________________________________
61 batch_normalization_8 (BatchNor (None, 32, 32, 128)  512         conv2d_8[0][0]
62 __________________________________________________________________________________________________
63 activation_8 (ReLu)              (None, 32, 32, 128)  0           batch_normalization_8[0][0]
64 __________________________________________________________________________________________________
65 max_pooling2d_4 (MaxPooling2D)   (None, 16, 16, 128)  0           activation_8[0][0]
66 __________________________________________________________________________________________________
67 dropout_4 (Dropout)              (None, 16, 16, 128)  0           max_pooling2d_4[0][0]
68 __________________________________________________________________________________________________
69 conv2d_9 (Conv2D)                (None, 16, 16, 256)  295168      dropout_4[0][0]
70 __________________________________________________________________________________________________
71 batch_normalization_9 (BatchNor (None, 16, 16, 256)  1024        conv2d_9[0][0]
72 __________________________________________________________________________________________________
73 activation_9 (ReLu)              (None, 16, 16, 256)  0           batch_normalization_9[0][0]
74 __________________________________________________________________________________________________
75 conv2d_10 (Conv2D)               (None, 16, 16, 256)  590080      activation_9[0][0]
76 __________________________________________________________________________________________________
77 batch_normalization_10 (BatchNo (None, 16, 16, 256)  1024        conv2d_10[0][0]
78 __________________________________________________________________________________________________
79 activation_10 (ReLu)             (None, 16, 16, 256)  0           batch_normalization_10[0][0]
80 __________________________________________________________________________________________________
81 conv2d_transpose_1 (Conv2DTrans (None, 32, 32, 128)  295040      activation_10[0][0]
82 __________________________________________________________________________________________________
83 concatenate_1 (Concatenate)      (None, 32, 32, 256)  0           conv2d_transpose_1[0][0]
84                                                                   activation_8[0][0]
85 __________________________________________________________________________________________________
86 dropout_5 (Dropout)              (None, 32, 32, 256)  0           concatenate_1[0][0]
87 __________________________________________________________________________________________________
88 conv2d_11 (Conv2D)               (None, 32, 32, 128)  295040      dropout_5[0][0]
89 __________________________________________________________________________________________________
90 batch_normalization_11 (BatchNo (None, 32, 32, 128)  512         conv2d_11[0][0]
```

```
 91 ____________________________________________________________________________________________________
 92 activation_11 (ReLu)            (None, 32, 32, 128)  0           batch_normalization_11[0][0]
 93 ____________________________________________________________________________________________________
 94 conv2d_12 (Conv2D)              (None, 32, 32, 128)  147584      activation_11[0][0]
 95 ____________________________________________________________________________________________________
 96 batch_normalization_12 (BatchNo (None, 32, 32, 128)  512         conv2d_12[0][0]
 97 ____________________________________________________________________________________________________
 98 activation_12 (ReLu)            (None, 32, 32, 128)  0           batch_normalization_12[0][0]
 99 ____________________________________________________________________________________________________
100 conv2d_transpose_2 (Conv2DTrans (None, 64, 64, 64)   73792       activation_12[0][0]
101 ____________________________________________________________________________________________________
102 concatenate_2 (Concatenate)     (None, 64, 64, 128)  0           conv2d_transpose_2[0][0]
103                                                                  activation_6[0][0]
104 ____________________________________________________________________________________________________
105 dropout_6 (Dropout)             (None, 64, 64, 128)  0           concatenate_2[0][0]
106 ____________________________________________________________________________________________________
107 conv2d_13 (Conv2D)              (None, 64, 64, 64)   73792       dropout_6[0][0]
108 ____________________________________________________________________________________________________
109 batch_normalization_13 (BatchNo (None, 64, 64, 64)   256         conv2d_13[0][0]
110 ____________________________________________________________________________________________________
111 activation_13 (ReLu)            (None, 64, 64, 64)   0           batch_normalization_13[0][0]
112 ____________________________________________________________________________________________________
113 conv2d_14 (Conv2D)              (None, 64, 64, 64)   36928       activation_13[0][0]
114 ____________________________________________________________________________________________________
115 batch_normalization_14 (BatchNo (None, 64, 64, 64)   256         conv2d_14[0][0]
116 ____________________________________________________________________________________________________
117 activation_14 (ReLu)            (None, 64, 64, 64)   0           batch_normalization_14[0][0]
118 ____________________________________________________________________________________________________
119 conv2d_transpose_3 (Conv2DTrans (None, 128, 128, 32) 18464       activation_14[0][0]
120 ____________________________________________________________________________________________________
121 concatenate_3 (Concatenate)     (None, 128, 128, 64) 0           conv2d_transpose_3[0][0]
122                                                                  activation_4[0][0]
123 ____________________________________________________________________________________________________
124 dropout_7 (Dropout)             (None, 128, 128, 64) 0           concatenate_3[0][0]
125 ____________________________________________________________________________________________________
126 conv2d_15 (Conv2D)              (None, 128, 128, 32) 18464       dropout_7[0][0]
127 ____________________________________________________________________________________________________
128 batch_normalization_15 (BatchNo (None, 128, 128, 32) 128         conv2d_15[0][0]
129 ____________________________________________________________________________________________________
130 activation_15 (ReLu)            (None, 128, 128, 32) 0           batch_normalization_15[0][0]
131 ____________________________________________________________________________________________________
132 conv2d_16 (Conv2D)              (None, 128, 128, 32) 9248        activation_15[0][0]
133 ____________________________________________________________________________________________________
134 batch_normalization_16 (BatchNo (None, 128, 128, 32) 128         conv2d_16[0][0]
135 ____________________________________________________________________________________________________
136 activation_16 (ReLu)            (None, 128, 128, 32) 0           batch_normalization_16[0][0]
137 ____________________________________________________________________________________________________
138 conv2d_transpose_4 (Conv2DTrans (None, 256, 256, 16) 4624        activation_16[0][0]
139 ____________________________________________________________________________________________________
140 concatenate_4 (Concatenate)     (None, 256, 256, 32) 0           conv2d_transpose_4[0][0]
141                                                                  activation_2[0][0]
142 ____________________________________________________________________________________________________
143 dropout_8 (Dropout)             (None, 256, 256, 32) 0           concatenate_4[0][0]
144 ____________________________________________________________________________________________________
145 conv2d_17 (Conv2D)              (None, 256, 256, 16) 4624        dropout_8[0][0]
146 ____________________________________________________________________________________________________
147 batch_normalization_17 (BatchNo (None, 256, 256, 16) 64          conv2d_17[0][0]
148 ____________________________________________________________________________________________________
149 activation_17 (ReLu)            (None, 256, 256, 16) 0           batch_normalization_17[0][0]
150 ____________________________________________________________________________________________________
151 conv2d_18 (Conv2D)              (None, 256, 256, 16) 2320        activation_17[0][0]
152 ____________________________________________________________________________________________________
153 batch_normalization_18 (BatchNo (None, 256, 256, 16) 64          conv2d_18[0][0]
154 ____________________________________________________________________________________________________
155 activation_18 (sigmoid)         (None, 256, 256, 16) 0           batch_normalization_18[0][0]
156 ____________________________________________________________________________________________________
157 conv2d_19 (Conv2D)              (None, 256, 256, 4)  68          activation_18[0][0]
158 ====================================================================================================
159 Total params: 2,164,500
160 Trainable params: 2,161,556
161 Non-trainable params: 2,944
162
```

Appendix C

Code

File - C:\Users\User\Desktop\Martin\DL\scripts\load.py

```
1 import random
2 import numpy as np
3 from scripts.unet import get_unet, get_unet2, get_unet3
4 from scripts.load_data import get_data, get_data2, get_test_data, get_test_data2, get_test_data3
5 from scripts.plot import plot_img, plot_categorical, plot_categorical2
6 from scripts.weighted_categorical_crossentropy import weighted_categorical_crossentropy
7 from scripts.save_data import save_array, save_array_from_categorical
8 from scripts.score import score
9
10 from sklearn.model_selection import train_test_split
11 from PIL import Image
12
13 from keras.models import Model
14 from keras.layers import Input, BatchNormalization, Activation, Dropout
15 from keras.optimizers import Adam
16 from keras.utils import multi_gpu_model
17 from keras.layers.convolutional import Conv2D, Conv2DTranspose
18 from keras.layers.pooling import MaxPooling2D
19 from keras.layers.merge import concatenate
20 from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
21 from keras.models import load_model
22
23 IMG_HEIGHT = 256
24 IMG_WIDTH = 256
25 IMG_DEPTH = 256
26 IMG_CHANNELS = 1
27 NR_CLASSES = 4
28 path_train2 = '/Users/User/Desktop/Martin/DL/input2/train/'
29 path_test2 = '/Users/User/Desktop/Martin/DL/input2/test/'
30
31 def load():
32     print("____________________RUNNING LOAD____________________")
33     X= get_test_data(path_test2)
34
35     # Split train and valid
36
37     input_img = Input((IMG_HEIGHT, IMG_WIDTH, 1), name='img')
38     model = get_unet(input_img, n_filters=16, dropout=0.02, batchnorm=True)
39     model = multi_gpu_model(model, gpus=2)
40
41     # Load best model
42     #keras.losses.custom_loss = weighted_categorical_crossentropy
43     #model = load_model('unet_model.h5', custom_objects={'weighted_categorical_crossentropy':
  weighted_categorical_crossentropy})
44     model.load_weights('unet_100_FS_wcc_weights.h5')
45
46
47     # Predict on train, val and test
48     preds_train = model.predict(X, verbose=1) #X_train)
49
50
51     save_array_from_categorical(preds_train,"labelFSwcc100")
52
53     model.load_weights('unet_100_FS_wcc2_weights.h5')
54     preds_train = model.predict(X, verbose=1)  # X_train
55     save_array_from_categorical(preds_train, "labelFSwcc2100")
56
57     model.load_weights('unet_100_FS_cc_weights.h5')
58     preds_train = model.predict(X, verbose=1)  # X_train
59     save_array_from_categorical(preds_train, "labelFScc100")
60
```

```
1 import random
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 IMG_HEIGHT = 256
6 IMG_WIDTH = 256
7 IMG_DEPTH = 256
8 IMG_CHANNELS = 1
9 NR_CLASSES = 4
10
11 def plot_categorical(nr,X_train, y_train):
12     y_train = y_train[nr]
13
14     y_train = y_train.reshape(IMG_WIDTH * IMG_DEPTH, NR_CLASSES)
15     y_train = np.argmax(y_train, axis=1)
16     y_train = y_train.reshape(IMG_WIDTH, IMG_DEPTH)
17
18     has_mask = y_train.max() > 0
19
20     fig, ax = plt.subplots(1, 2, figsize=(20, 10))
21
22     ax[0].imshow(X_train[nr, ..., 0], cmap='gray', interpolation='bilinear')
23     if has_mask:
24         ax[0].contour(y_train.squeeze(), colors='k', levels=[0.5])
25     ax[0].set_title('First')
26
27     ax[1].imshow(y_train.squeeze(), interpolation='bilinear', cmap='gray')
28     #ax[1].imshow(y_train[ix], interpolation='bilinear', cmap='gray')
29     ax[1].set_title('Second');
30     plt.show()
31
32 def plot_categorical2(nr, X_train, y_train, pred, pred2):
33
34     fig, ax = plt.subplots(1, 4, figsize=(20, 10))
35     y_train = y_train[nr]
36
37     y_train = y_train.reshape(IMG_WIDTH * IMG_DEPTH, NR_CLASSES)
38     y_train = np.argmax(y_train, axis=1)
39     y_train = y_train.reshape(IMG_WIDTH, IMG_DEPTH)
40
41     has_mask = y_train.max() > 0
42
43     ax[0].imshow(X_train[nr, ..., 0], cmap='gray', interpolation='bilinear')
44     if has_mask:
45         ax[0].contour(y_train.squeeze(), colors='k', levels=[0.5])
46     ax[0].set_title('Input + Countour')
47
48     ax[1].imshow(y_train.squeeze(), interpolation='bilinear', cmap='gray')
49     #ax[1].imshow(y_train[ix], interpolation='bilinear', cmap='gray')
50     ax[1].set_title('Ground Truth');
51
52
53     pred2 = pred2[nr]
54
55     pred2 = pred2.reshape(IMG_WIDTH * IMG_DEPTH, NR_CLASSES)
56     pred2 = np.argmax(pred2, axis=1)
57     pred2 = pred2.reshape(IMG_WIDTH, IMG_DEPTH)
58
59     has_mask = pred2.max() > 0
60
61     ax[2].imshow(pred[nr, ..., 0], cmap='gray', interpolation='bilinear')
62     if has_mask:
63         ax[2].contour(pred2.squeeze(), colors='k', levels=[0.5])
64     ax[2].set_title('Scores')
65
66     ax[3].imshow(pred2.squeeze(), interpolation='bilinear', cmap='gray')
67     ax[3].set_title('Result');
68     plt.show()
69
70
71 def plot_categorical_argmax(nr, X_train, y_train, pred, pred2):
72
73     fig, ax = plt.subplots(1, 4, figsize=(20, 10))
74     y_train = y_train[nr]
75
76     y_train = y_train.reshape(IMG_WIDTH * IMG_DEPTH, NR_CLASSES)
77     y_train = np.argmax(y_train, axis=1)
78     y_train = y_train.reshape(IMG_WIDTH, IMG_DEPTH)
79
80     has_mask = y_train.max() > 0
81
82     ax[0].imshow(X_train[nr, ..., 0], cmap='gray', interpolation='bilinear')
83     if has_mask:
84         ax[0].contour(y_train.squeeze(), colors='k', levels=[0.5])
85     ax[0].set_title('Input + Countour')
86
87     ax[1].imshow(y_train.squeeze(), interpolation='bilinear', cmap='gray')
88     #ax[1].imshow(y_train[ix], interpolation='bilinear', cmap='gray')
89     ax[1].set_title('Ground Truth');
90
```

```
 91
 92     pred2 = pred2[nr]
 93
 94     pred2 = pred2.reshape(IMG_WIDTH * IMG_DEPTH, NR_CLASSES)
 95     pred2 = np.argmax(pred2, axis=1)
 96     pred2 = pred2.reshape(IMG_WIDTH, IMG_DEPTH)
 97
 98     has_mask = pred2.max() > 0
 99
100     ax[2].imshow(pred[nr, ..., 0], cmap='gray', interpolation='bilinear')
101     if has_mask:
102         ax[2].contour(pred2.squeeze(), colors='k', levels=[0.5])
103     ax[2].set_title('Scores')
104
105     ax[3].imshow(pred2.squeeze(), interpolation='bilinear', cmap='gray')
106     ax[3].set_title('Result');
107     plt.show()
108
109
110 def plot_img(X_train, y_train):
111     ix = random.randint(0, len(X_train))
112     has_mask = y_train[ix].max() > 0
113
114     fig, ax = plt.subplots(1, 2, figsize=(20, 10))
115
116     ax[0].imshow(X_train[ix, ..., 0], cmap='seismic', interpolation='bilinear')
117     if has_mask:
118         ax[0].contour(y_train[ix].squeeze(), colors='k', levels=[0.5])
119     ax[0].set_title('Seismic')
120
121     ax[1].imshow(y_train[ix].squeeze(), interpolation='bilinear', cmap='gray')
122     ax[1].set_title('Salt');
123     plt.show()
124     print("Done! img show")
```

```
1 from keras.models import Model
2 from keras.layers import Input, BatchNormalization, Activation, Dropout
3 from keras.layers.convolutional import Conv2D, Conv2DTranspose, Conv3D, Conv3DTranspose
4 from keras.layers.pooling import MaxPooling2D, MaxPooling3D
5 from keras.layers.merge import concatenate
6
7
8 def conv2d_block(input_tensor, n_filters, kernel_size=3, batchnorm=True):
9     # first layer
10    x = Conv2D(filters=n_filters, kernel_size=(kernel_size, kernel_size), kernel_initializer="he_normal",
11               padding="same")(input_tensor)
12    if batchnorm:
13        x = BatchNormalization()(x)
14    x = Activation("relu")(x)
15    # second layer
16    x = Conv2D(filters=n_filters, kernel_size=(kernel_size, kernel_size), kernel_initializer="he_normal",
17               padding="same")(x)
18    if batchnorm:
19        x = BatchNormalization()(x)
20    x = Activation("relu")(x)
21    return x
22
23 def conv3d_block(input_tensor, n_filters, kernel_size=3, batchnorm=True):
24    # first layer
25    x = Conv3D(filters=n_filters, kernel_size=(kernel_size, kernel_size, kernel_size), kernel_initializer="he_normal",
26               padding="same")(input_tensor)
27    if batchnorm:
28        x = BatchNormalization()(x)
29    x = Activation("relu")(x)
30    # second layer
31    x = Conv3D(filters=n_filters, kernel_size=(kernel_size, kernel_size, kernel_size), kernel_initializer="he_normal",
32               padding="same")(x)
33    if batchnorm:
34        x = BatchNormalization()(x)
35    x = Activation("relu")(x)
36    return x
37
38
39 def get_unet2(input_img, n_filters=16, dropout=0.5, batchnorm=True):
40    # contracting path
41    c1 = conv2d_block(input_img, n_filters=n_filters * 1, kernel_size=3, batchnorm=batchnorm)
42    p1 = MaxPooling2D((2, 2))(c1)
43    p1 = Dropout(dropout * 0.5)(p1)
44
45    c2 = conv2d_block(p1, n_filters=n_filters * 2, kernel_size=3, batchnorm=batchnorm)
46    p2 = MaxPooling2D((2, 2))(c2)
47    p2 = Dropout(dropout)(p2)
48
49    c3 = conv2d_block(p2, n_filters=n_filters * 4, kernel_size=3, batchnorm=batchnorm)
50    p3 = MaxPooling2D((2, 2))(c3)
51    p3 = Dropout(dropout)(p3)
52
53    c4 = conv2d_block(p3, n_filters=n_filters * 8, kernel_size=3, batchnorm=batchnorm)
54    p4 = MaxPooling2D(pool_size=(2, 2))(c4)
55    p4 = Dropout(dropout)(p4)
56
57    c5 = conv2d_block(p4, n_filters=n_filters * 16, kernel_size=3, batchnorm=batchnorm)
58
59    # expansive path
60    u6 = Conv2DTranspose(n_filters * 8, (3, 3), strides=(2, 2), padding='same')(c5)
61    u6 = concatenate([u6, c4])
62    u6 = Dropout(dropout)(u6)
63    c6 = conv2d_block(u6, n_filters=n_filters * 8, kernel_size=3, batchnorm=batchnorm)
64
65    u7 = Conv2DTranspose(n_filters * 4, (3, 3), strides=(2, 2), padding='same')(c6)
66    u7 = concatenate([u7, c3])
67    u7 = Dropout(dropout)(u7)
68    c7 = conv2d_block(u7, n_filters=n_filters * 4, kernel_size=3, batchnorm=batchnorm)
69
70    u8 = Conv2DTranspose(n_filters * 2, (3, 3), strides=(2, 2), padding='same')(c7)
71    u8 = concatenate([u8, c2])
72    u8 = Dropout(dropout)(u8)
73    c8 = conv2d_block(u8, n_filters=n_filters * 2, kernel_size=3, batchnorm=batchnorm)
74
75    u9 = Conv2DTranspose(n_filters * 1, (3, 3), strides=(2, 2), padding='same')(c8)
76    u9 = concatenate([u9, c1], axis=3)
77    u9 = Dropout(dropout)(u9)
78    c9 = conv2d_block(u9, n_filters=n_filters * 1, kernel_size=3, batchnorm=batchnorm)
79
80    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)
81    model = Model(inputs=[input_img], outputs=[outputs])
82    return model
83
84 ## Unte for categorical data
85
86 def get_unet(input_img, n_filters=16, dropout=0.5, batchnorm=True):
87    # contracting path
88    c1 = conv2d_block(input_img, n_filters=n_filters * 1, kernel_size=3, batchnorm=batchnorm)
89    p1 = MaxPooling2D((2, 2))(c1)
90    p1 = Dropout(dropout * 0.5)(p1)
```

```
 91
 92     c2 = conv2d_block(p1, n_filters=n_filters * 2, kernel_size=3, batchnorm=batchnorm)
 93     p2 = MaxPooling2D((2, 2))(c2)
 94     p2 = Dropout(dropout)(p2)
 95
 96     c3 = conv2d_block(p2, n_filters=n_filters * 4, kernel_size=3, batchnorm=batchnorm)
 97     p3 = MaxPooling2D((2, 2))(c3)
 98     p3 = Dropout(dropout)(p3)
 99
100     c4 = conv2d_block(p3, n_filters=n_filters * 8, kernel_size=3, batchnorm=batchnorm)
101     p4 = MaxPooling2D(pool_size=(2, 2))(c4)
102     p4 = Dropout(dropout)(p4)
103
104     c5 = conv2d_block(p4, n_filters=n_filters * 16, kernel_size=3, batchnorm=batchnorm)
105
106     # expansive path
107     u6 = Conv2DTranspose(n_filters * 8, (3, 3), strides=(2, 2), padding='same')(c5)
108     u6 = concatenate([u6, c4])
109     u6 = Dropout(dropout)(u6)
110     c6 = conv2d_block(u6, n_filters=n_filters * 8, kernel_size=3, batchnorm=batchnorm)
111
112     u7 = Conv2DTranspose(n_filters * 4, (3, 3), strides=(2, 2), padding='same')(c6)
113     u7 = concatenate([u7, c3])
114     u7 = Dropout(dropout)(u7)
115     c7 = conv2d_block(u7, n_filters=n_filters * 4, kernel_size=3, batchnorm=batchnorm)
116
117     u8 = Conv2DTranspose(n_filters * 2, (3, 3), strides=(2, 2), padding='same')(c7)
118     u8 = concatenate([u8, c2])
119     u8 = Dropout(dropout)(u8)
120     c8 = conv2d_block(u8, n_filters=n_filters * 2, kernel_size=3, batchnorm=batchnorm)
121
122     u9 = Conv2DTranspose(n_filters * 1, (3, 3), strides=(2, 2), padding='same')(c8)
123     u9 = concatenate([u9, c1], axis=3)
124     u9 = Dropout(dropout)(u9)
125     c9 = conv2d_block(u9, n_filters=n_filters * 1, kernel_size=3, batchnorm=batchnorm)
126
127     outputs = Conv2D(4, (1, 1), activation='sigmoid')(c9)
128     model = Model(inputs=[input_img], outputs=[outputs])
129     model.summary()
130     return model
131
132 def get_unet3(input_img, n_filters=16, dropout=0.5, batchnorm=True):
133     # contracting path
134     c1 = conv3d_block(input_img, n_filters=n_filters * 1, kernel_size=3, batchnorm=batchnorm)
135     p1 = MaxPooling3D((2, 2, 2))(c1)
136     p1 = Dropout(dropout * 0.5)(p1)
137
138     c2 = conv3d_block(p1, n_filters=n_filters * 2, kernel_size=3, batchnorm=batchnorm)
139     p2 = MaxPooling3D((2, 2, 2))(c2)
140     p2 = Dropout(dropout)(p2)
141
142     c3 = conv3d_block(p2, n_filters=n_filters * 4, kernel_size=3, batchnorm=batchnorm)
143     p3 = MaxPooling3D((2, 2, 2))(c3)
144     p3 = Dropout(dropout)(p3)
145
146     c4 = conv3d_block(p3, n_filters=n_filters * 8, kernel_size=3, batchnorm=batchnorm)
147     p4 = MaxPooling3D(pool_size=(2, 2, 2))(c4)
148     p4 = Dropout(dropout)(p4)
149
150     c5 = conv3d_block(p4, n_filters=n_filters * 16, kernel_size=3, batchnorm=batchnorm)
151
152     # expansive path
153     u6 = Conv3DTranspose(n_filters * 8, (3, 3, 3), strides=(2, 2, 2), padding='same')(c5)
154     u6 = concatenate([u6, c4])
155     u6 = Dropout(dropout)(u6)
156     c6 = conv3d_block(u6, n_filters=n_filters * 8, kernel_size=3, batchnorm=batchnorm)
157
158     u7 = Conv3DTranspose(n_filters * 4, (3, 3, 3), strides=(2, 2, 2), padding='same')(c6)
159     u7 = concatenate([u7, c3])
160     u7 = Dropout(dropout)(u7)
161     c7 = conv3d_block(u7, n_filters=n_filters * 4, kernel_size=3, batchnorm=batchnorm)
162
163     u8 = Conv3DTranspose(n_filters * 2, (3, 3, 3), strides=(2, 2, 2), padding='same')(c7)
164     u8 = concatenate([u8, c2])
165     u8 = Dropout(dropout)(u8)
166     c8 = conv3d_block(u8, n_filters=n_filters * 2, kernel_size=3, batchnorm=batchnorm)
167
168     u9 = Conv3DTranspose(n_filters * 1, (3, 3, 3), strides=(2, 2, 2), padding='same')(c8)
169     u9 = concatenate([u9, c1])#, axis=3
170     u9 = Dropout(dropout)(u9)
171     c9 = conv3d_block(u9, n_filters=n_filters * 1, kernel_size=3, batchnorm=batchnorm)
172
173     outputs = Conv3D(4, (1, 1, 1), activation='sigmoid')(c9)
174     model = Model(inputs=[input_img], outputs=[outputs])
175     return model
176
```

```
1
2 import os
3 import sys
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import nibabel as nib
7 plt.style.use("ggplot")
8
9 from tqdm import tqdm_notebook
10 from skimage.transform import resize
11 from nibabel.testing import data_path
12
13 IMG_HEIGHT = 400
14 IMG_WIDTH = 400
15 IMG_DEPTH = 275
16 IMG_CHANNELS = 1
17 NR_CLASSES = 4
18
19
20 def score():
21     print('score')
22     # Get train and test IDs
23     train_ids =  next(os.walk('/Users/User/Desktop/Martin/DL/input2/score/'))[2]
24
25     # Get and resize train images and masks
26     Y_train = np.zeros((len(train_ids), IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), dtype=np.float32)
27     print('Getting and resizing train images and masks ... ')
28     sys.stdout.flush()
29
30     for n, id_ in tqdm_notebook(enumerate(train_ids), total=len(train_ids)):
31         example_filename2 = os.path.join(data_path, '/Users/User/Desktop/Martin/DL/input2/score/' + id_)
32         print(example_filename2)
33         img = nib.load(example_filename2)
34         data = img.get_fdata()
35
36         data = resize(data, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
37         data = data.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH)
38         data = np.rint(data)
39         print(Y_train.shape)
40         print(data.shape)
41         Y_train[n] = data
42
43     score_value(0,Y_train[0],Y_train[1])
44     score_value(1, Y_train[0], Y_train[1])
45     score_value(2, Y_train[0], Y_train[1])
46     score_value(3, Y_train[0], Y_train[1])
47
48
49
50 def score_all_values(true_data, predicted_data):
51     print(true_data.shape)
52     print(np.unique(true_data))
53     print(predicted_data.shape)
54     print(np.unique(predicted_data))
55     print("All values equals:")
56     rigth_predictions = np.equal(true_data, predicted_data)
57     print(np.sum(rigth_predictions))
58     print("/")
59     print(true_data.size)
60     print('==============')
61     print(np.sum(rigth_predictions)/true_data.size)
62
63 def score_value(value, true_data, predicted_data):
64     true_values = np.isin(true_data, [value])
65     true_values_predicted=predicted_data[true_data == [value]]
66     predicted0=np.isin(true_values_predicted,[0])
67     predicted1=np.isin(true_values_predicted,[1])
68     predicted2=np.isin(true_values_predicted,[2])
69     predicted3=np.isin(true_values_predicted,[3])
70     all_prediction = np.sum(predicted0)+np.sum(predicted1)+np.sum(predicted2)+np.sum(predicted3)
71     print("   %i,  %i,  %i,  %i" % (np.sum(predicted0), np.sum(predicted1), np.sum(predicted2), np.sum(predicted3)))
72
```

```
1 import random
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 plt.style.use("ggplot")
6
7
8 from scripts.load_data import get_data, get_data2, get_data3
9 from scripts.unet import get_unet, get_unet2, get_unet3
10 from scripts.plot import plot_img, plot_categorical
11 from scripts.weighted_categorical_crossentropy import weighted_categorical_crossentropy,
  weighted_categorical_crossentropy2
12
13 from sklearn.model_selection import train_test_split
14
15 from keras.layers import Input, BatchNormalization, Activation, Dropout
16 from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
17 from keras.optimizers import Adam
18 from keras.utils import multi_gpu_model
19
20 # Set parameters for dimentions and data
21 IMG_HEIGHT = 256
22 IMG_WIDTH = 256
23 IMG_DEPTH = 256
24 IMG_CHANNELS = 1
25 NR_CLASSES = 4
26 path_train2 = '/Users/User/Desktop/Martin/DL/input2/train/'
27 path_test2 = '/Users/User/Desktop/Martin/DL/input2/test/'
28
29
30
31 def train():
32     print("______________________RUNNING TRAIN______________________")
33
34     #Get data from folder, see scipts\load_data.py for more
35     X, y = get_data(path_train2, train=True)
36
37     print(X.shape)
38     print(y.shape)
39
40     #Split the data into train and validation
41     X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.14, random_state=2018)
42
43     #Set image dimentions and size
44     input_img = Input((IMG_HEIGHT, IMG_WIDTH, 1), name='img')
45
46     #Create CNN unet model
47     model = get_unet(input_img, n_filters=16, dropout=0.02, batchnorm=True)
48
49     #RUN multiple GPUS, remove if only one gpu
50     model = multi_gpu_model(model, gpus=2)
51
52     #Compiile with loss function
53     model.compile(optimizer=Adam(), loss=weighted_categorical_crossentropy(), metrics=["accuracy"]) # '
  binary_crossentropy' , weighted_categorical_crossentropy()
54     #Print model summary
55     model.summary()
56
57     #MODEL SAVE does not work with multiple GPUS
58     #model.save('unet_100_PD_model.h5')
59
60     #Set early stopping, reduce learning rate, and save weights if better
61     callbacks = [
62         EarlyStopping(patience=30, verbose=1),
63         ReduceLROnPlateau(factor=0.1, patience=6, min_lr=0.000001, verbose=1),
64         ModelCheckpoint('unet_100_FS_wcc_weights.h5', verbose=1, save_best_only=True, save_weights_only=True)
65     ]
66
67     #Train model
68     results = model.fit(X_train, y_train, batch_size=32, epochs=100, class_weight='auto', callbacks=callbacks,
69                         validation_data=(X_valid, y_valid))
70
71     #####################################################
72
73     #Create CNN unet model
74     model1 = get_unet(input_img, n_filters=16, dropout=0.02, batchnorm=True)
75
76     #RUN multiple GPUS, remove if only one gpu
77     model1 = multi_gpu_model(model1, gpus=2)
78
79     #Compiile with loss function
80     model1.compile(optimizer=Adam(), loss=weighted_categorical_crossentropy2(), metrics=["accuracy"])  # '
  categorical_crossentropy' , weighted_categorical_crossentropy(),weighted_categorical_crossentropy()
81     #Print model summary
82     model1.summary()
83
84     #MODEL SAVE does not work with multiple GPUS
85     #model.save('unet_100_PD_model.h5')
86
87     #Set early stopping, reduce learning rate, and save weights if better
```

```
88      callbacks1 = [
89          EarlyStopping(patience=30, verbose=1),
90          ReduceLROnPlateau(factor=0.1, patience=6, min_lr=0.000001, verbose=1),
91          ModelCheckpoint('unet_100_FS_wcc2_weights.h5', verbose=1, save_best_only=True, save_weights_only=True)
92      ]
93
94      #Train model
95      results1 = model1.fit(X_train, y_train, batch_size=32, epochs=100, class_weight='auto', callbacks=callbacks1,
96                          validation_data=(X_valid, y_valid))
97
98      ######################################################################
99
100
101     plt.figure(figsize=(8, 8))
102     plt.title("Learning curve")
103     plt.plot(results.history["loss"], label="loss")
104     plt.plot(results.history["val_loss"], label="val_loss")
105     plt.plot( np.argmin(results.history["val_loss"]), np.min(results.history["val_loss"]), marker="x", color="r", label
    ="best model")
106     plt.xlabel("Epochs")
107     plt.ylabel("log_loss")
108     plt.legend
109     plt.show()
110
111     #Load best weights
112     model.load_weights('unet_100_FST1_wcc_weights.h5')
113
114
115     # Evaluate on validation set (this must be equals to the best log_loss)
116     model.evaluate(X_valid, y_valid, verbose=1)
117
118     print("DONE")
119
120
```

```
1 #####
```

```
1 import os
2 import sys
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import nibabel as nib
6
7 from keras.utils import to_categorical
8 from tqdm import tqdm_notebook, tnrange
9 from skimage.transform import resize
10 from nibabel.testing import data_path
11 from keras.preprocessing.image import img_to_array, load_img
12
13 plt.style.use("ggplot")
14
15 # Set some parameters
16 IMG_HEIGHT = 256
17 IMG_WIDTH = 256
18 IMG_DEPTH = 256
19 IMG_CHANNELS = 1
20 NR_CLASSES = 4
21 path_train = '/Users/User/Desktop/Martin/DL/input2/train/'
22 path_test = '/Users/User/Desktop/Martin/DL/input2/test/'
23
24
25 def get_data3(path, train=True):
26     # Get train and test IDs
27     train_ids1 = next(os.walk(path + "imagesPD"))[2]
28
29     # Get and resize train images and masks
30     X_train = np.zeros((len(train_ids1), IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, 3), dtype=np.float32)
31     Y_train = np.zeros((len(train_ids1), IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, NR_CLASSES), dtype=np.float32)
32     print('Getting and resizing train images and masks ... ')
33     sys.stdout.flush()
34
35     for n, id_ in tqdm_notebook(enumerate(train_ids1), total=len(train_ids1)):
36         example_filename1 = os.path.join(data_path, path + 'imagesPD/' + id_)
37         example_filename2 = os.path.join(data_path, path + 'imagesFS/' + id_)
38         example_filename3 = os.path.join(data_path, path + 'imagesT1/' + id_)
39         print(example_filename1)
40         print(example_filename2)
41         print(example_filename3)
42         img1 = nib.load(example_filename1)
43         data1 = img1.get_fdata()
44
45         img2 = nib.load(example_filename2)
46         data2 = img2.get_fdata()
47
48         img3 = nib.load(example_filename3)
49         data3 = img3.get_fdata()
50
51         data1 = resize(data1, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
52         data1 = data1.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
53
54         data2 = resize(data2, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
55         data2 = data2.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
56
57         data3 = resize(data3, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
58         data3 = data3.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
59
60         data4 = np.concatenate((data1, data2, data3), axis=3)
61
62         X_train[n] = data4
63         # ________________________________________________________________________________________
64
65         example_filename2 = os.path.join(data_path, path + 'masks/' + id_)
66         img = nib.load(example_filename2)
67         data = img.get_fdata()
68
69         data = resize(data, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
70         data = data.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
71         data = np.rint(data)
72         data = to_categorical(data)
73         data = data.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, NR_CLASSES)
74         Y_train[n] = data
75
76     X_train = X_train.reshape(len(train_ids1) * IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, 3)
77     Y_train = Y_train.reshape(len(train_ids1) * IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, NR_CLASSES)
78
79     return X_train, Y_train
80
81
82 def get_data2(path, train=True):
83     # Get IDs
84     train_ids1 = next(os.walk(path + "imagesFS"))[2]
85
86     # Get and resize train images and masks
87     X_train = np.zeros((len(train_ids1), IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, 2), dtype=np.float32)
88     Y_train = np.zeros((len(train_ids1), IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, NR_CLASSES), dtype=np.float32)
89     print('Getting and resizing train images and masks ... ')
90     sys.stdout.flush()
```

```
 91
 92     for n, id_ in tqdm_notebook(enumerate(train_ids1), total=len(train_ids1)):
 93         example_filename1 = os.path.join(data_path, path + 'imagesFS/' + id_)
 94         example_filename2 = os.path.join(data_path, path + 'imagesT1/' + id_)
 95         print(example_filename1)
 96         print(example_filename2)
 97         img1 = nib.load(example_filename1)
 98         data1 = img1.get_fdata()
 99
100         img2 = nib.load(example_filename2)
101         data2 = img2.get_fdata()
102
103         data1 = resize(data1, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
104         data1 = data1.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
105
106         data2 = resize(data2, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
107         data2 = data2.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
108
109         data3 = np.concatenate((data1, data2), axis=3)
110
111         X_train[n] = data3
112         # ________________________________________________________________________________________
113
114         example_filename2 = os.path.join(data_path, path + 'masks/' + id_)
115         img = nib.load(example_filename2)
116         data = img.get_fdata()
117
118         data = resize(data, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
119         data = data.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
120         data = np.rint(data)
121         data = to_categorical(data)
122         data = data.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, NR_CLASSES)
123         Y_train[n] = data
124
125     X_train = X_train.reshape(len(train_ids1) * IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, 2)
126     Y_train = Y_train.reshape(len(train_ids1) * IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, NR_CLASSES)
127
128     return X_train, Y_train
129
130 ### to categorical get data
131
132 def get_data(path, train=True):
133     # Get train and test IDs
134     train_ids =  next(os.walk(path + "imagesFS"))[2]
135
136     # Get and resize train images and masks
137     X_train = np.zeros((len(train_ids), IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS), dtype=np.float32)
138     Y_train = np.zeros((len(train_ids), IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, NR_CLASSES), dtype=np.float32)
139     print('Getting and resizing train images and masks ... ')
140     sys.stdout.flush()
141
142     for n, id_ in tqdm_notebook(enumerate(train_ids), total=len(train_ids)):
143         example_filename = os.path.join(data_path, path + 'imagesFS/' + id_ )
144         print(example_filename)
145         img = nib.load(example_filename)
146         data = img.get_fdata()
147
148         data = resize(data, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
149         data = data.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
150
151         X_train[n] = data
152         # ________________________________________________________________________________________
153
154         example_filename2 = os.path.join(data_path, path + 'masks/' + id_ )
155         img = nib.load(example_filename2)
156         data = img.get_fdata()
157
158         data = resize(data, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
159         data = data.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
160         data = np.rint(data)
161         data = to_categorical(data)
162         data = data.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, NR_CLASSES)
163         Y_train[n] = data
164
165     X_train = X_train.reshape(len(train_ids) * IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
166     Y_train = Y_train.reshape(len(train_ids) * IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, NR_CLASSES)
167
168     return X_train,Y_train
169
170
171 def get_test_data(path, train=True):
172     # Get IDs
173     test_ids =  next(os.walk(path + "imagesFS"))[2]
174
175     # Get and resize train images and masks
176     X_test = np.zeros((len(test_ids), IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS), dtype=np.float32)
177     print('Getting and resizing train images and masks ... ')
178     sys.stdout.flush()
179
180     for n, id_ in tqdm_notebook(enumerate(test_ids), total=len(test_ids)):
```

```
181         example_filename = os.path.join(data_path, path + 'imagesFS/' + id_ )
182         print(example_filename)
183         img = nib.load(example_filename)
184         data = img.get_fdata()
185
186         data = resize(data, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
187         data = data.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
188
189         X_test[n] = data
190
191     X_test = X_test.reshape(len(test_ids) * IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
192
193
194     return X_test
195
196 def get_test_data2(path, train=True):
197     # Get IDs
198     test_ids =  next(os.walk(path + "imagesFS"))[2]
199
200     # Get and resize train images and masks
201     X_test = np.zeros((len(test_ids), IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, 2), dtype=np.float32)
202     print('Getting and resizing train images and masks ... ')
203     sys.stdout.flush()
204
205     for n, id_ in tqdm_notebook(enumerate(test_ids), total=len(test_ids)):
206         example_filename = os.path.join(data_path, path + 'imagesFS/' + id_ )
207         example_filename2 = os.path.join(data_path, path + 'imagesT1/' + id_)
208         print(example_filename)
209         print(example_filename2)
210         img = nib.load(example_filename)
211         data = img.get_fdata()
212         img2 = nib.load(example_filename2)
213         data2 = img2.get_fdata()
214
215         data = resize(data, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
216         data = data.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
217
218         data2 = resize(data2, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
219         data2 = data2.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
220
221         data3 = np.concatenate((data, data2), axis=3)
222         data3 = data3.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, 2)
223
224         X_test[n] = data3
225
226     X_test = X_test.reshape(len(test_ids) * IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, 2)
227
228
229     return X_test
230
231 def get_test_data3(path, train=True):
232     # Get IDs
233     test_ids =  next(os.walk(path + "imagesPD"))[2]
234
235     X_test = np.zeros((len(test_ids), IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, 3), dtype=np.float32)
236     print('Getting and resizing train images and masks ... ')
237     sys.stdout.flush()
238
239     for n, id_ in tqdm_notebook(enumerate(test_ids), total=len(test_ids)):
240         example_filename = os.path.join(data_path, path + 'imagesPD/' + id_ )
241         example_filename2 = os.path.join(data_path, path + 'imagesFS/' + id_)
242         example_filename3 = os.path.join(data_path, path + 'imagesT1/' + id_)
243         print(example_filename)
244         print(example_filename2)
245         print(example_filename3)
246         img = nib.load(example_filename)
247         data = img.get_fdata()
248         img2 = nib.load(example_filename2)
249         data2 = img2.get_fdata()
250         img3 = nib.load(example_filename3)
251         data3 = img3.get_fdata()
252
253         data = resize(data, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
254         data = data.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
255
256         data2 = resize(data2, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
257         data2 = data2.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
258
259         data3 = resize(data3, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
260         data3 = data3.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
261
262         data4= np.concatenate((data, data2, data3), axis=3)
263
264         X_test[n] = data4
265
266     X_test = X_test.reshape(len(test_ids) * IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, 3)
267
268
269     return X_test
270
```

```
271
272 def get_data3D(path, train=True):
273
274     train_ids =  next(os.walk(path + "images"))[2]
275
276     X_train = np.zeros((len(train_ids), IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS), dtype=np.float32)
277     Y_train = np.zeros((len(train_ids), IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, NR_CLASSES), dtype=np.float32)
278     print('Getting and resizing train images and masks ... ')
279     sys.stdout.flush()
280
281     for n, id_ in tqdm_notebook(enumerate(train_ids), total=len(train_ids)):
282         example_filename = os.path.join(data_path, path + 'images/' + id_ )
283         print(example_filename)
284         img = nib.load(example_filename)
285         data = img.get_fdata()
286
287         data = resize(data, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
288         data = data.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
289
290         X_train[n] = data
291
292         # ______________________________________________________________________________________
293
294         example_filename2 = os.path.join(data_path, path + 'masks/' + id_ + ".gz")
295         img = nib.load(example_filename2)
296         data = img.get_fdata()
297
298
299         data = resize(data, (IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH), mode='constant', preserve_range=True)
300         data = data.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, IMG_CHANNELS)
301         data = np.rint(data)
302         data = to_categorical(data)
303         data = data.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, NR_CLASSES)
304         Y_train[n] = data
305
306     return X_train,Y_train
307
```

```
1 import numpy as np
2 import nibabel as nib
3 from skimage.transform import resize
4
5
6 ORG_HEIGHT = 400
7 ORG_WIDTH = 400
8 ORG_DEPTH = 275
9
10 IMG_HEIGHT = 256
11 IMG_WIDTH = 256
12 IMG_DEPTH = 256
13 IMG_CHANNELS = 1
14 NR_CLASSES = 4
15
16
17 def save_array_from_categorical(array, file_string):
18     save_data = np.argmax(array, axis=3)
19     save_data = save_data.reshape(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH, 1)
20     data = resize(save_data, (ORG_HEIGHT, ORG_WIDTH, ORG_DEPTH, 1), mode='constant', preserve_range=True, anti_aliasing=
   False, order=0)
21
22     empty_header = nib.Nifti1Header()
23
24     affine = np.array([[0, 0, 1, 0], [-1, 0, 0, 0], [0, -1, 0, 0], [0, 0, 0, 1]])
25     print(affine)
26     clipped_img = nib.Nifti1Image(data, affine, empty_header)
27     nib.save(clipped_img, '%s.nii' % (file_string))
28
29
30 def save_array(array, file_string):
31
32     empty_header = nib.Nifti1Header()
33     affine = np.diag([1, 1, 1, 1])
34
35     clipped_img = nib.Nifti1Image(array, affine, empty_header)
36     nib.save(clipped_img, '%s.nii' % (file_string))
```

```
1 from keras import backend as K
2 import numpy as np
3
4
5 def weighted_categorical_crossentropy():
6
7     weights = np.array([1.0, 1.05, 6.9, 11.8])
8
9     weights = K.variable(weights)
10
11     def loss(y_true, y_pred):
12         # Scale predictions so that the class probas of each sample sum to 1
13         y_pred /= K.sum(y_pred, axis=-1, keepdims=True)
14         #Clip
15         y_pred = K.clip(y_pred, K.epsilon(), 1 - K.epsilon())
16         #Calculate
17         loss = y_true * K.log(y_pred) * weights
18         loss = -K.sum(loss, -1)
19         return loss
20
21     return loss
22
23 def weighted_categorical_crossentropy2():
24
25     weights = np.array([1.0, 1.1, 12.0, 19.7])
26
27     weights = K.variable(weights)
28
29     def loss(y_true, y_pred):
30         # scale predictions so that the class probas of each sample sum to 1
31         y_pred /= K.sum(y_pred, axis=-1, keepdims=True)
32         # Clip
33         y_pred = K.clip(y_pred, K.epsilon(), 1 - K.epsilon())
34         # Calculate
35         loss = y_true * K.log(y_pred) * weights
36         loss = -K.sum(loss, -1)
37         return loss
38
39     return loss
```