Norwegian University
of Life Sciences

Master's Thesis 2021 30 ECTS
Faculty of Science and Technology

Medical image representations in cancer segmentation

Markus Ola Holte Granheim
Data Science

This page is intentionally left blank.

Acknowledgments

Firstly, I want to thank my supervisors Profs. Kristian Liland and Oliver Tomic for the subject of this thesis, their enthusiasm, interesting conversations, and feedback as well as their dedication in lecturing and teaching, preparing me for this semester.

There are also several others at the Norwegian Unversity of Life Science I am grateful for. A thanks to Prof. Cecilia M. Futsæther who has included me in the physics master students discussions and given me constructive feedback. Ass. Prof. Aurora Rosvoll Grøndahl and PhD. Bao Ngoc Hyuhn for assisting me both on possible ways to analyze the data and answering questions regarding the dataset, Deoxys software, and the Orion cluster.

Moreover, I want to thank my fellow study hall mates, teaching me how to make "proper" coffee, and keeping me company several late nights.

The last thanks, go to my family always supporting me, and my friends making my time at the university joyful.

Abstract

Purpose

Cancer is the second leading cause of death globally and was responsible for an estimated 9.6 million deaths in 2018 [1]. One of the treatments used to cure cancer is radiotherapy, where a precise delineation of the cancer is crucial. The radiation therapy needs to be precise not to damage any surrounding tissues or, in the worst-case organs. However, the delineation process is both time-consuming and affected by inter-observer variability. Automating this process will free up time for the radiologist, fasten the treatment time, and remove the inter-observer variability.

Theory and Method

Before the machine learning model can assist with medical delineations, it must perform well. In this thesis, use of U-Net architecture is used to train the model on CT (Computed Tomography), and PET (Positron Emission Tomography) scans gathered from the University Hospital in Oslo. The dataset consists of 197 patients and is divided into three sets; 142 in the training set, 40 in the test, and 15 in the validation set. New aspects of this research are implementing two types of image representations, histogram of oriented gradients (HOG) and local binary pattern (LBP). The U-Net is run on different models using combinations of the already existing medical images and the new image representations. Several model parameters have also been tested, such as augmentation, windowing, and the use of dilated convolutions.

The HOG images are created by using local contrast normalized blocks of histograms of oriented gradients. The different image representations take multiple parameters into account when created. This thesis has focused on the descriptor and normalization blocks' different sizes when creating the HOG images. In contrast to earlier work, this thesis uses the image representation of HOG and not its descriptor vector, which is commonly used in earlier studies.

LBP uses local structural changes to represent each pixel. Different radius sizes and number of descriptive points have been tested to find the optimal LBP - parameters.

Results

The result found that augmentation is vital for a precise delineation, increasing all the different model's performance. The results also show that the image representations perform best when the descriptor blocks and radius are small. HOG is the best performing of the two image representations. However, they do not beat the medical images alone (0.666). The highest performance was reached when combining the medical images and HOG image representation using both the CT and PET channels (0.675).

Conclusion

The thesis has shown that adding new image representation to the model can improve performance. However, the score has only increased the performance by one percent, and there is a significant computational cost creating these image representations. There are still multiple ways to improve the image representations, and HOGs full potential has not been exploited. This thesis, therefore, leaves room for numerous future works regarding the usage of the HOGs image representation and possibilities for implementing other image representations for segmentation tasks.

Contents

List of Figures

List of Tables

Abbreviations

Abbreviation	Meaning
AI	Artifical intelligence
CNN	Convolutional Neural Network
CP	Center Pixel
CPB	Cells Per Block
df	Degrees of Freedom
DL	Deep Learning
hdf5	Hierarchical Data Format version 5
HOG	Histogram of Oriented Gradients
LBP	Local Binary Patterns
MAE	Mean Absolute Error
ML	Machine Learning
MSE	Mean Squared Error
NP	Neighbouring Pixel
PPC	Pixel Per Cell
ReLU	Rectified Linear Units
WC	Window Center
WW	Window Width

Nomenclature

Symbols	Meaning
σ	The linear combination of the input nodes in a layer
ω_i	Weights for the i'th layer
Θ	Orientation of the pixels gradient
$\phi(..)$	Activation function
$\phi_{ReLU}(z) = max(0, z)$	The ReLU activation function
$\phi_{sigmoid}(z) = \frac{1}{1+exp(-z)}$	The sigmoid activation function
f	CNN filter size
G	magnitude of the pixel
$J(w)$	Cost function
P	LBP number of points
R	LBP Radius
s	CNN stride
W	Histograms vector output

Chapter 1

Introduction

1.1 Motivation

There are more than 200 types of cancer [2]. The common denominator is that some of the body's cells start to divide without stopping and spreading into surrounding tissues [3]. Cancer is the second leading cause of death globally and was responsible for an estimated 9.6 million deaths in 2018 [1]. An estimated 300 000 cases of head and neck cancer in 2014 [4]. In 2019, head and neck cancers accounted for 4% of new cancer cases in the USA [5]. The most significant risks for evolving cancer in the head and neck region are tobacco use and alcohol consumption [6].

The treatment for head and neck cancer differs, but the main types are medical, radiation, and surgical. Radiation therapy, also known as radiotherapy destroying cancer cells by using high-intensity radiation. It can be both the primary and secondary treatment for head and neck cancer. The treatment could be used after surgery to destroy the remaining cancer cells, mainly if the tumor is located in a difficult spot that cannot be removed surgically [3].

1.1.1 Effective delineation using deep learning

Before the treatment of head and neck cancer, an CT/PET scan is taken. The scan is interpreted by radiologists and further used to remove tumors. This process is time-consuming, and the demand for radiologists is higher than ever. With an increasing number of imaging machines, there is also a growing demand to interpret

images. This makes implementing machine learning to automate the delineation process both important and exciting. Automating this process may reduce the radiologist's load and make the process more efficient so more people can get help. Before this happens a precise delineation of the tumor is vital.

1.1.2 Implementation of new image representation

To achieve a good delineation of the tumor, the computer must get as much helpful information as possible, such as the PET/CT images are known to have and is the reason these images have been used for a long time. This thesis will be trying to make use of image representations to unlock further information and help the model see new patterns that might improve the performance of the segmentation.

1.2 Aims of the master thesis

This thesis will focus on radiation therapy and cancer segmentation using machine learning on medical images (PET-CT). The aims of the thesis are to see if implementing new image representations such as; histogram of oriented gradients (HOG) and Local Binary Patterns (LBP) can improve the delineation, and see what information they bring. This is done by adding them as new channels in the model both alone as separate channels and together with the medical images.

In Chapter 2, this thesis explains head- and neck cancer and the theory behind the HOG images. It will also deal with the methods used in machine learning to predict cancer tumors. Chapter 3 will explain the methodology used before the results are presented in Chapter 4 and discussed in Chapter 5. Chapters 6 and 7 contain respectively further work and conclusion.

1.3 Previous work

The image representations used are HOG and LBP, both of which have been used for over a decade. HOG got its breakthrough in Dalal and Triggs Paper in 2005 on detecting standing people in images with almost perfect accuracy. [7], where a lot of their theory regarding contrast normalization was based on Lowe's paper "Distinctive image features from scale-invariant key points" presented in 2004 [8].

Dalal and Triggs used the HOGs vector output, together a Support vector machine (SVM) and a sliding window for segmentation purposes, which still is the most common way to use the HOG descriptor, done by several people after them [9] [7] [10].

Later a combination of HOG and LBP have been tested, where information from both the LBP and HOG values were fed into an SVM classifier [10] [11]. Following Reddy tested out using the sliding window for image segmentation purposes [12]. Where there have been tested out numerous different parameters, both the R-HOG and C-HOG [7] and the segmental-HOG descriptor block [13]. These descriptor blocks will be discussed in the theory chapter. But the use of the image representation of HOG is not common.

This thesis will give an overview of how the image representations were implemented and the results of it. The thesis is built on the earlier research by former students at the Norwegian university of life science (NMBU), where some have tried out using radiomics for segmentation [14] [15], while others have used U-Net structures, which has lead to this thesis [16] [17].

Chapter 2

Theory

Section 2.1, and 2.2 provide some background on cancer and how the medical images are constructed. Further, in sections 2.3 and 2.4, the theory behind creating HOG- and LBP -image representations will be explained. The last sections, 2.5 and 2.6, will present the theory behind the segmentation. Section 2.5 gives an introduction to deep learning, followed up in section 2.6 by how the convolutional network works.

2.1 Cancer

Cancers can vary in behavior, how they spread and grow [18]. Some can be found in a solid form, made up of a mass of tissues, while other types, such as leukemia, will not have a tangible form. Malignant tumors can spread to other places, while benign tumors do not spread and are not considered cancer tumors [3].

Our body is continuously changing the cells. New cells replace old and dying cells. Each cell contains genetic material (DNA). DNA can be described as the recipe of the cell and controls how it evolves. It tells the cell what types of action it should perform and how to divide and grow [19]. A genetic mutation occurs if there is an error in the genetic material. These errors may entail that healthy cells act differently, and in the worst case, lead to cancer. The most typical genetic mutation leading to cancer are; allowing for rapid cell-division, failing to stop cell growth, and making mistakes when repairing DNA errors [19]. If the cells start to divide uncontrollably, it will lead to an overflow of cells in that area. If this cell

growth is restricted, in a lump, encapsulated, its called a tumor.

The genetic mutations can be inherited from parents or developed independently on it is own. The majority of cancers are caused by errors in the body's cell division, leading to genetic mutations [18]. Throughout blood veins and lymph, the cancer cells can spread to new places, making the treatment harder [20]. One of the most common cancers is head and neck cancer.

2.1.1 Head- and neck cancer

Head- and Neck cancer (HNC) makes up around 550,000 annual cases worldwide each year. It is the 6th leading cancer by incidents and is responsible for approximately 300 000 deaths annually [4]. HNC describes several different cancers in the head and neck region. The common denominator is that they all are malignant tumors located in the region around the throat, larynx, nose, sinuses, and mouth [21]. Over 90% of HNC originate from squamous cell carcinomas [4]. The squamous cell makes up the outer layers of the skin and cancers originating here can spread further into the body [22]. The squamous cells are the outer part of the "epithelium"-layer. Cancers started in these cells can often be seen on the skin.

The location of the cancer often tells us what kind of cancer it is. An example is cancers originating from the buccal mucosa area. These are the cells in the squamous layer making up the lining of the inner cheeks and the back of the lips, and it is called "carcinoma buccal mucosa" [23].

Cancers in the squamous cell layer, that have not spread further, are called "carcinoma in situ". If the cancer has grown beyond this cell layer an spread deeper into the body, it is called "invasive squamous cell carcinoma" (SCC) [24]. The symptoms can be all from the loosening of teeth to sore throat.

2.1.2 Risk factors and prevention

A risk factor is anything that increases the risk of getting cancer. The terms of risks are many, but the most prominent factors are age, gender, tobacco- and alcohol consumption. Men are more likely to get HNCs than women [6]. The different causes for this are not entirely clear. However, some scientists have presented hypotheses where lifestyle factors are considered the main reason, while others believe lower willingness to seek the necessary treatment are more decisive [25].

Age is a risk factor for many different types of cancer. Cancers including HNCs is a conseqence of errors in the cell division. This is because with time there comes the body have done more cell division, an therefore increasing the chance for a error in the division leading to a cancer [26]. However this research have the last years been questioned in multiple reports points, some claiming the lifestyle of the elders and not the necessarily the age [27] [28]. The general denominator for all the risk factors is increasing the chance for errors and creating a mutation in cell division mutation.

The most significant factor that can be easily influenced is the consumption of alcohol and usage of tobacco. The use of tobacco is the single most considerable risk of cancer and can link to around 85% of the cases [6]. Alcohol consumption increases the risk for cancer in the region between the mouth and esophagus. Combining them makes the risk even higher [6]. Other risk factors are prolonged sun exposure, Human Papillomavirus (HPV), and Epstein Barr Virus (EBV).

2.1.3 Treatments

This section will provide a brief overview of different treatments and present a detailed description of the term "standard of care." The "standard of care" is the treatments that are currently state of the art for a specific cancer indication. The standard of care consists of different therapies depending on the cancer type, size, and place. The standard treatments are; surgery, radiation therapy, chemotherapy, and targeted therapy [3]. Surgery removes the tumor and some surrounding tissues by hand and is applied if the cancer is easily accessible. Radiation therapy uses a high level of radiation, usually X-rays, but protons or other types of energy could also be used. The radiation after-effects can last for a long time, which will keep killing cancer cells up to several months after the intervention. It is applied when the tumor is hard to access [3]. Target Therapy targets the protein that controls how a cell grows, divide and spread [29]. Chemotherapy is the last of the commonly used methods. It involves the use of medication to treat the cancer [30]. The medicine may also give severe side effects, such as hair loss and reduced immune system, and others [31]. This thesis will focus on radiation therapy. Here, it is essential to determine where the tumor's location, size, and shape.

A team of oncologists analyses various factors that are influential on the treatment, such as; the tumor's location, accessibility, cancer type, and whether it could be treated with radiation, medication, or target therapy. The common denominator is that the treatments need a precise detection of the specific location and size

of the tumor. Medical images can obtain this information. PET-CT is the most commonly used type of medical imaging to acquire this information [32].

2.2 PET and CT -images

PET and CT are the two most used medical imaging techniques to detect cancer. PET is an abbreviation for Positron Emission Tomography, and CT stands for Computed Tomography. From these images, it is possible to locate the tumor, see how large the tumor is, and if the tumor is malignant or not. The images have been used for over a decade and are still state of the art. The two methods differ significantly, and combining the images, creates a good understanding of the tumor's location [32].

In the last years, new scanners have been introduced that can simultaneously generate CT and PET scans. These machines have made the imaging process less time-consuming and can handle more patients than before. Earlier, these were two separate machines which made it less effective and also more costly [33].

2.2.1 Positron Emission Tomography

Positron Emission Tomography (PET) uses radio-labeled glucose known as tracers. These tracers could be made from molecules in the body and could be all from sugar, protein to a hormone. The tracers are designed depending on what part of the body is being inspected. For the brain, glucose is often used as the radioactive molecule injected.

The molecule is made radioactive from a biological synthesizer, which makes the molecule give out positrons for the following hours, so this must be done straight before the PET scan. The molecules are then injected into the body's bloodstream. The tracers are gathered where there is a high number of cell activity going on, often around a tumor.

When the positron is let out, it will fly around in the brain until it crashes into an electron. These two particles destroy each other, sending two gamma-rays in the exact opposite directions. When the PET machine detects two gamma rays on opposite sides, the doctors can calculate the position of the tracers. The gamma rays are captured in the PET scan by using a ring of Geiger-counters. These

machines can detect thousands of gamma rays each second. With the help of some computations, it can get an accurate estimate and an image of where the tracers are located [34].

After a few hours, the molecules have sent out all the positrons, and the body is back to the normal state.

2.2.2 Computed Tomography

Computed Tomography (CT) is an X-ray technique. The CT instrument fires a large number of X-rays into the body. Depending on how many of the X-rays are absorbed by each tissue, it is possible to tell what tissue it is. Some tissues like bone and more rigid material in our body absorb more X-rays and will be shown as a bright pixel in an image; softer tissues tend to absorb less and appear darker. This may provide information on what type of tissue it is and if something is unusual. Each image represents a thin slice of the body. They are taken from multiple different angles. The images are computed to a 3d image [35]. In some cases, adding a contrast fluid makes room for a higher difference in the intensity values in the image and possibly easier to detect organs, bones, etc. The three-dimensional image given out is super detailed. Instead of using a color spectrum with 256 values, it uses a spectrum that reaches several thousands of different intensities to visualize the image.

Windowing

The spectrum of color intensities in a CT-SCAN is vast and can range over several thousand different intensity values. However, it is not always something that is possible in an image with an 8-bit color spectrum.

It is usually not crucial for cancer detection how much the bone structure absorbs since most cancer occurs in softer tissues where there is a higher amount of cell division. When the spectrum is vast, and the intensity change between cancer tissue and normal tissue can be minor, applying a windowing operation could help. This operation enhances the part of the spectrum that is of interest and changes the values outside this part to maximum or minimum in our new spectrum [36]. This means if a spectrum is between 0 and 1000, but the cancer is known to be in the spectrum range 250-750, a windowing operation can scale up this part and put the values ≤ 250 to 250 and the values ≥ 750 to 750. Figure 2.1 illustrates

how windowing works, where the left line represents 250 and the right line 750.

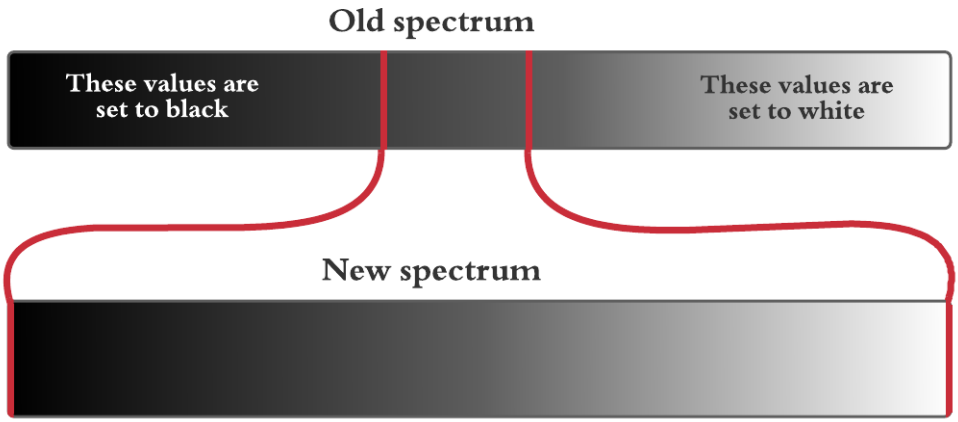


Figure 2.1: Illustration of how windowing creates a new spectrum from a section of the old one.

2.3 Histogram of Oriented Gradients

The following chapter is inspired by Granheim's semester assignment, written as a preparation for this thesis, and can be found in appendix A.

2.3.1 Overview of Method

One of the image representations presented in this paper is the Histogram of Oriented Gradients (HOG). This section gives an overview of the theory behind manufacturing the HOG images. It will go through each step of the creation separately. The idea involves using locally contrast normalized blocks in a dense grid of oriented gradients. The oriented gradients are calculated from edge-detection filters and are then placed into a histogram that compresses the data.

HOG-images have been used for over a decade, used for fast image segmentation tasks such as self driving cars and facial recognition. The method got its breakthrough in 2005 after Dalal and Triggs used HOG and a sliding window to detect standing people in images with almost perfect accuracy [7]. After this, it has been

frequently used in face detection and application where a fast decision is crucial, such as self-driving cars.

HOG transforms the image to a descriptor vector, capturing the entire vectors information. This vector can be fed into a classifier, such as the Support Vector Machines. Which is the same as Dalal and Triggs used back in 2005 [7]. This thesis will use the vector's image representation as an input channel in the U-Net model, which will be explained later in this chapter.

A digital image is created by several pixels placed in a grid-like structure. Each pixel contains information about what color and light intensity lies in that specific pixel. A black and white image uses only one number to represent each pixel. The number reaches between zero and 255. Where zero means the pixel is black, and 255 is white, the rest represents the range of grey levels. In this thesis, all images can be assumed to be digital.

To define a colored image, each pixel would be represented by multiple channels. These channels combined represent a color given from a color map, the most frequently used color map is RGB, using red, green, and blue to represent all colors. An RGB image is usually made up of 3 channels, where each channel represents one of the mentioned colors. However, there are other color maps where there can be used other color combinations to represent the images. Sometimes there is also added an extra channel to identify the intensity of the color.

PET and CT scans create one channeled images (Black and White), which will be used as the starting image for creating HOG. With a colored image, HOG would look at the highest number in each pixel and treat it as a one-channeled image. Merging the CT and PET scans down to one channel is not wanted, so for the next chapters, it is assumed that the functions are applied on them separately and they are both handled as one-channeled images.

2.3.2 Edge detection

The largest amount of information is usually gathered around the edges of an image. From edges, it is possible to identify objects' shapes and sizes and capture a sense of the texture in the material. An edge operation simplifies an image down to its edges. This can remove wanted information, but also potentially remove unnecessary noise, and capture the vital parts of an image. There are several edge detections that can be used. The one used in this thesis is a simplified Sobel filter.

Sobel filter

The edge detector used in this thesis is a variant of the Sobel filter. The Sobel filter is a simple matrix that iterates over the image, calculating the edges in both horizontal and vertical directions in two steps, with the eight neighboring pixels help. The Sobel filter uses a 3x3 matrix that iterates over the image, calculating the gradients of the central pixels. The most common Sobel filters are:

For horizontal calculation: $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

For vertical calculation: $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$

The Sobel filter iterates over the pixels covering the entire image. If the change is prominent in the horizontal or vertical direction, the respective filters will give a significant value. The output is provided into two new matrices, one matrix for the horizontal edges and one for the vertical edges.

The difference between the two examples above is that the right side values the corner pixels the same as the closest pixels. In contrast, the right side's filters value the nearest pixels two times more than the corner pixels.

The last one, is a simplified version. It uses a 1x3 and 3x1 filter instead, making the computational cost less. This means that it only looks at the X-axis's closest pixels when calculating the horizontal edges and the Y-axis pixels when calculating the vertical edges.

The simplified sobel filter: $\begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$

After running these two filters you are left with two new matrices that represent the edges in the horizontal and vertical directions.

2.3.3 Creation of Gradients

Two matrices are given from the edge detection, one for the horizontal edges and the other for the vertical edges. The edges' magnitude is provided by the size of the color/intensity-change in the filter; however, only two directions are calculated. It is possible to combine these to matrices using simple trigonometry to calculate each pixel gradient's magnitude and direction.

Using arctangent and the vertical and horizontal magnitude, you can calculate the direction of the edge for each pixel by:

$$\Theta = \arctan(\frac{G_y}{G_x}) \tag{2.1}$$

With Pythagoras's Theorem, you can calculate the total magnitude G by:

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.2}$$

Here G_y is the magnitude of the edge in the vertical direction, and G_x is the magnitude of the edge in the horizontal direction, both are given from the Sobel filter.

The new information is transformed into two new matrices; one magnitude and one orientation matrix. An example is visualized in Figure 2.2

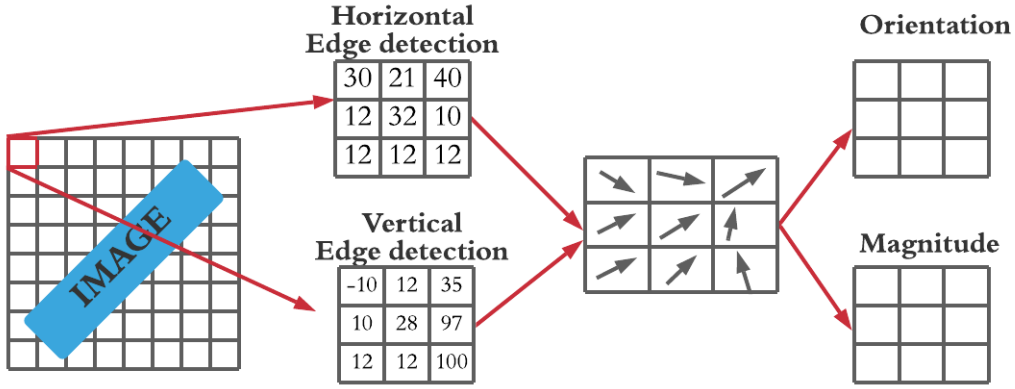


Figure 2.2: The figure shows how the image magnitude and orientation function is created, starting with finding the vertical and horizontal edges, these two matrices are combined to find the orientation and magnitude of each pixel

2.3.4 Creation of histogram

From the magnitude and gradient matrices, it is possible to divide the data into histograms. This is done in several ways, but all of them distribute the magnitude values into bins corresponding to their orientations.

The first factor when creating the histograms is how each orientation in the bin is represented. Creating a 360 bin histogram will make one bin for each degree. This has been tried out but has shown to give almost the same result, with an increase in computational cost [7]. Later research from Dalal and Triggs found a small increase in performance when using signed gradients on detection of e.g. cars and motorbikes. Dividing the orientation into larger bin sizes has proven to give better results [7]. The most used bin size is 20 degrees in each bin. This makes 18 bins if you use a signed spectral with 360 degrees or nine bins if you are using unsigned degrees. Unsigned degrees use 180-degree spectral and negative magnitudes to represent the last 180 degrees.

The magnitude is placed into the bin where the orientation fits the most. If the orientation is between two of the containers, the magnitude is divided into two. How much of the magnitude that goes to which of the bins is determined by how close the orientation was to the bins. If the direction were located precisely in the middle of the bins, the magnitude would be divided 50/50 between them. Figure 2.3 shows an example of how the magnitude is distributed.

This operation compresses the image a lot. Compressing the image down to only 9 or 18 values makes little sense. Since so much of the data will be lost if one tries to capture the essence in 9 values. Therefore the image is divided into blocks and creating multiple histograms. This will still compress the image, however, it will leave room for capturing more of the information. The size of these blocks varies depending on how detailed the image should be. Standard values are dividing 8x8 blocks into nine bin-histograms [37].

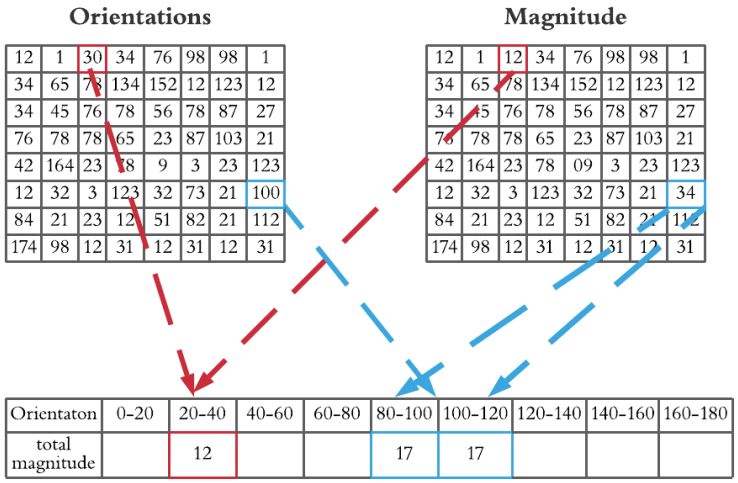


Figure 2.3: Orientation and magnitude matrix to vector, the red lines shows how a magnitude is directly placed into a bin if the orientation is in the middle of the bin. The blue line shows an example of how the magnitude is divided if the orientation is exactly in the middle of two bins.

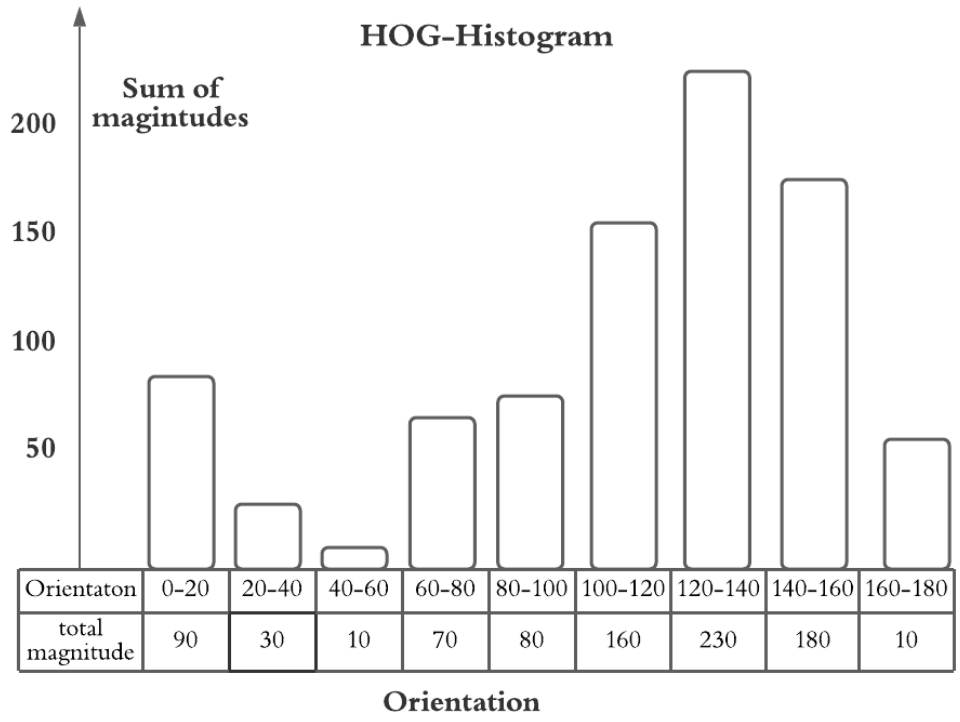


Figure 2.4: A visualization of how one descriptor blocks HOG histogram can be seen after the vector is created.

2.3.5 Descriptor blocks

The edge gradients in an image can be transparent or stretch over multiple pixels. These scenarios are essential to catch and gather the most amount of information in the shot possible. This is also why a contrast normalization across multiple histogram blocks has proven to give the best results. Which descriptor block used, decides what blocks, and how many that is taken into the contrast normalization. The normalization process will be explained in the next section. There have mainly been tested out two different descriptor block shapes on HOG, the R-HOG and the C-HOG.

It is possible to choose a step length on the iterating descriptor block so that each histogram is counted even more times or only once. There are two common different descriptor blocks sizes; R-HOG and C-HOG.

R-HOG

R-HOG is the most common descriptor block used in the creation of HOG images. This is also the simplest of the two explained in this thesis. It uses a rectangular receptive field, which fits very well with the rectangular histogram blocks. The most common R-HOG uses a three-by-three receptive field. This means that all the cells gather information from their eight adjacent cells, merging all the cells vectors down to one large vector for each cell. If there were used a nine bin histogram and a three-by-three descriptor block each cell would consist of a histogram of 81 bins after the cross normalization. This large histogram can be seen as a vector, which will then be normalized.

C-HOG

C-HOG uses a circular receptive field instead of R-HOG's rectangular. The idea behind C-HOG is that it only uses the closest pixels. To gather the information that lies closest to the center pixel, a circle should be used. The distance between the center and the delimiter of a circle is constant, while the distance between the center point and the delimiter in the rectangular is not. Here the corners are the furthest away from the center, while the middle point on the long sides is the closest.

Dalal and Triggs tested this idea [7], however they did not reach as good a score

as with R-HOG.

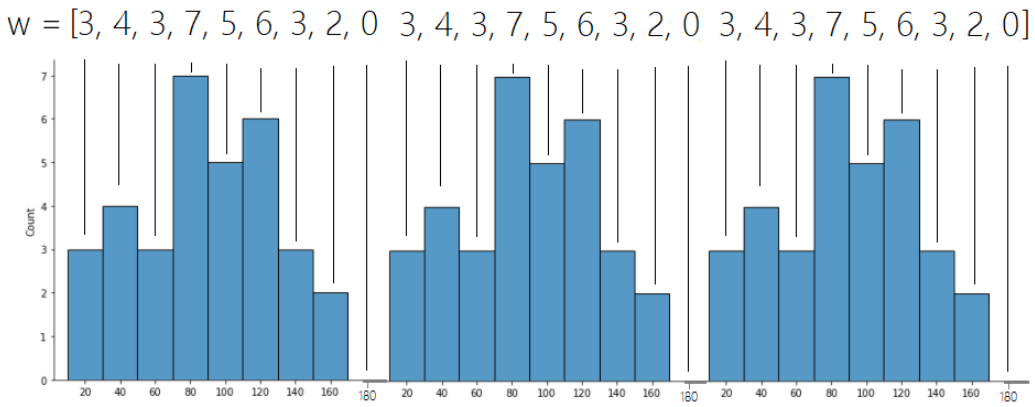


Figure 2.5: A visualisation of how a vector with illustration of histogram could look after using a normalization block, covering three block. Here all of the three blocks histograms are identical, which is usually not the case.

2.3.6 Normalization in HOG

There can be used different normalizations in HOG. The four most commonly used methods are offspring from the L1 and L2 regularization. These are the same methods implemented in the SciKit-HOG function package and what is frequently tested out when researching this subject [7] [37].

The concept of normalization is to re-scale the values down to the same scale. Usually this scale is between zero and one. There are multiple reasons for normalizing the data, most important for the machine learning is that the values are easily comparable and that they match the weight initialisation. This will save run-time and epoch by avoiding that the weights have to learn the class imbalance first. An other important thing is that the gradients in the image is sensitive for the overall lighting. For instance, dividing all the intensity values by two, the image will get darker and the gradients will change by half. Ideally the histogram is not affected by the background lighting, and the descriptor is independent. Normalization helps to achieve this. With a high span in values, it can be challenging to weigh them correctly. This is because the higher values easily can be more strongly weighted. How you want to re-scale them is different depending on what normalization and regularization used. The common denominator is that they find the total vector sum and divide each value by it. The total weight varies depending on the ”penalty” that is added. The most commonly used is L2-norm.

The bins in the histogram result in the vector. The size of the normalization blocks tells us how many of the histogram that goes into making one vector. With a 3-by-3 descriptor block and a nine bin histogram, we would end up with a vector "W" with 81 elements(3x3x9). A figure with a three structured normalization block can be seen in Figure 2.5.

This vector then adds a normalization parameter before it gets normalized. What is common for all of the methods is that they add a penalty term ϵ to the vectors sum, and divide each element in the original vector with this sum. Vector $\mathbf{W}_j$ is the original vector $\mathbf{W}$ with the added the penalty ϵ.

$$\mathbf{W}_j = \mathbf{W} + \epsilon = [W_0, W_1, ..., W_N, \epsilon] \tag{2.3}$$

L2 uses the squared value when calculating the sum of the vector. However the related L1-approach uses the absolute value instead.

Definition 2.3.1. Definition of L1 norm and L2 norm

$$L2 : \|W_j\|_2 = \sqrt{\sum_{j=1}^{m} \mathbf{W}_j^2} \tag{2.4}$$

$$L1 : \|W_j\|_1 = \sum_{j=1}^{m} |\mathbf{W}_j| \tag{2.5}$$

In these functions $\|W_j\|_2$ is the sum using the L2-normalisation, $\|W_j\|_1$ is the sum using the L1-normalisation.

The normalized vector is then calculated by taking the original vector and dividing it by the sum of W_j.

Definition 2.3.2.

$$L2 : \mathbf{W}_N = \frac{\mathbf{W}}{\|W_j\|_2} \tag{2.6}$$

$$L1 : \mathbf{W}_N = \frac{\mathbf{W}}{\|W_j\|_1} \tag{2.7}$$

The normalized vector W_N, is calculated by taking the original output-vector W and dividing it by the sum. The sum is given from the calculation in definition 2.3.1 depending on what normalization used, where $\|W\|_2$ is the L2s sum and $\|W\|_1$ is L1s.

The main difference between these are that the L1 penalty can tend to place the weights closer to the end of the specter, while the round penalty term L2 is better at keeping the features. The other two outsprings from these methods are the L1-sqrt and L2-Hys -normalization.

L2-Hys is the second method and uses the same normalization as L2 but limits the maximum values to not be higher than 0.2.

Definition 2.3.3. Definition of L1-sqrt, using the same as L1 but adding a square root.

$$\mathbf{W} = \sqrt{\frac{\mathbf{W}}{\|W_j\|_1}} \tag{2.8}$$

2.3.7 Creation of vector

From the normalization, multiple larger histograms are created. How large these histograms are depend on numerous factors, the most important: the size of the bins, descriptor blocks and the normalization blocks. The values can be folded out, creating a long array of numbers representing the image, then further fed into an machine learning model such as a support vector machine. This is what is mainly used in earlier researches regarding HOG [7]. This thesis will not use the vector but an image representation of the vectors.

2.3.8 Creation of image

Each histogram consists of several bins. Each bin has an orientation and a value representing the magnitude. From these two numbers, a pixel representation of the star-histogram can be presented. In the star-histogram, each bin will be rotated in the same orientation that it corresponds to, and the length of the bin is dependent on the magnitude: the higher the magnitude, the longer the bin.

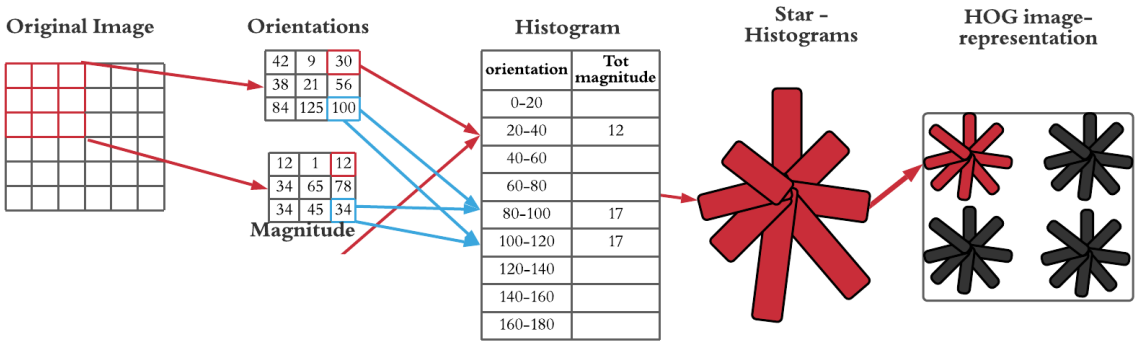


Figure 2.6: A figure showcasing the process of creating one of the star histograms in HOG, without the use of a normalization block

One star histogram will be made for every block in the image. The image representation is then made by adding all the star histograms to a black surface in their respective places. The process can be seen in the end of the HOG template shown in Figure 2.6. The figure shows how the image is divided into cells of 3x3 pixels, creating the magnitude and orientation matrices. Thereafter the vector is calculated, before showing the image representations of the star histogram for the same grid like structure it was divided in at the beginning of the process. The figure does not show the normalization process which would take place after placing the magnitude into bins.

2.4 Local Binary Pattern

Local binary pattern (LBP) is the second image representation used in this master thesis. It has been frequently used in face detection earlier and has made room for the algorithm to find new patterns. This method is often seen as a method to study local properties in an image and identify these parts' characteristics [38].

2.4.1 Overview of the method

The concept behind LBP is to describe the neighboring pixel using binary codes [38]. LBP creates a new value for each pixel. The value is generated by creating a binary vector combining the center cell with its surrounding eight neighboring cells. The vector is generated using a threshold function, where the center value is set as the threshold and compared with its surrounding pixels.

2.4.2 Creation of the Binary vector

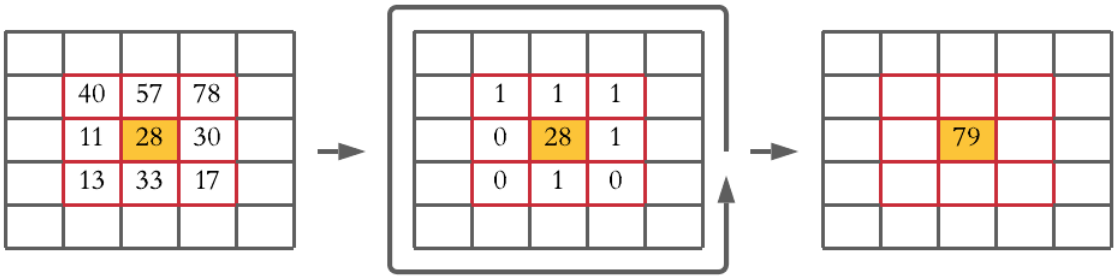


Figure 2.7: LBP-template, here the yellow pixel is the center pixel (CP) and the red box marks the neighbouring pixels (NP), the second image shows the neighbouring values after threshold the image, before the output is shown in the last part.

First, it will look into a section of the image to see what is being gathered. From Figure 2.7, the yellow pixel is being calculated, and the red box marks the neighboring pixels. The threshold is set to the center pixels value, in this case, 28. The middle matrix shows the image after the threshold, where the values above the center value are set to one and the values below zero. The number is then put into a vector gathered in a counter-clockwise direction. With the pixel to the right being the first. This order can also be changed, as long as it is the same order for the entire picture. In this example, the vector would be: [1, 1, 1, 1, 0, 0, 1, 0]

2.4.3 From vector to number

The vector is transformed as if it were in the binary system to the decimal system. Here the first number in the vector is multiplied by 2^0, the second with 2^1, and so on until the last digit with 2^7. The LBP equation can be seen as:

$$C(x) = \begin{cases} 1, & \text{if x} \geq 0 \\ 0, & \text{if x} < 0 \end{cases} \tag{2.9}$$

C gives out 1 or 0 depending on the difference between the center value and neighboring values.

$$LBP(CP, NP) = \sum_{p=0}^{P-1} C(NP - CP) \times 2^p \tag{2.10}$$

Here CP is the Center pixels value/threshold, the NP is the neighbor pixels, and C is the function 2.9 seen above. This output will represent the center value. Then it will iterate over the entire picture, calculating the LBP values for all the pixels.

For the example given above the calculation would be:

$$LBP = 2^0*1+2^1*1+2^2*1+2^3*1+2^4*0+2^5*0+2^6*1+2^7*0 = 1+2+4+8+64 = 79 \tag{2.11}$$

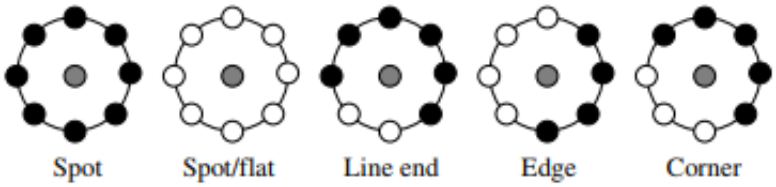


Figure 2.8: LBP-texture analysis, here the neighbouring values are drawn black or white depending on whether they are above the threshold or not.

How the LBP numbers can show information is visualized in Figure 2.8. These patterns are represented by a number, which can decide whether it is a corner, a spot, or line end, etc. Together with the rest of the image, this information can give a way of visualizing and hopefully find new patterns in the data.

2.4.4 Different parameters

Two main components can be tweaked when working with the LBP function. These are the number of neighboring points (P) and the radius (R) from the center value to the adjacent pixel. Figure 2.9 explains where the neighboring pixels would have been located depending on what parameter values were used.

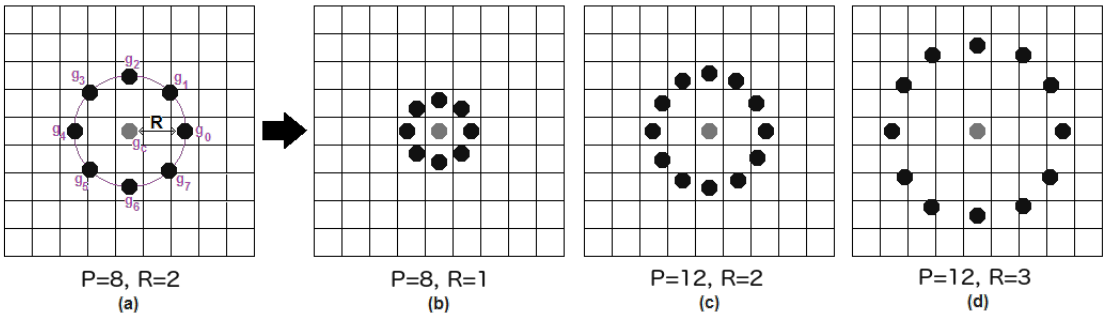


Figure 2.9: LBP-experimental setups, here it can be seen how the radius (R) and the number of points (P) affect the LBP setup.

The images created from the medical images are then fed into a machine learning algorithm which will try to exploit new patterns in the data set and make the best possible cancer segmentation. How the patterns are found will be explained in the next two subchapters.

2.5 Deep Learning

The term Artificial intelligence, known as AI, is defined in many different ways. One of the term's founders, John McCarthy defined it as "the science and engineering of making intelligent machines [39]." It refers to all types of simulations that mimic human intelligence. Machine Learning, known as ML, is considered a subset of artificial intelligence and refers to the concept that computers can learn and adapt to new data without human assistance. This thesis uses deep learning, known as DL, which again is a subset of machine learning. DL is a set of algorithms, often referred to as a deep artificial neural network [40]. DL uses deeper neural networks while ML usually uses more classic model-based heuristic methods for pattern detection and prediction. DL contains a more complex structure with a deeper network, containing a higher number of trainable parameters.

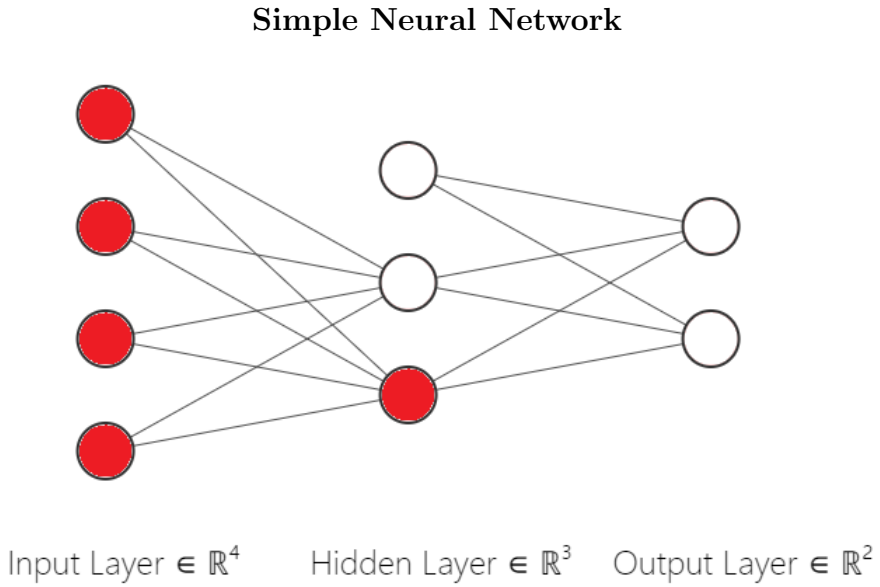


Figure 2.10: Example of a neural network diagram structure, showcasing which parts of the network are involved in the computations of one node.

2.5.1 Introduction to deep artificial neural networks

Neural networks are built up of three main types of layers; an input layer, hidden layers, and an output layer. These layers are used to mimic the human brain. The input layer takes in the information given, passing it on to the hidden layers to train up different scenarios before the output layer provides us with a prediction. Each layer contains nodes that hold the information and make room for finding combinations in the data. In Figure 2.10, a diagram of a simple neural network is presented. The figure starts with an input layer containing four nodes, where the top node is the "bias node." The input layer then goes into a hidden layer. The hidden layer has three nodes, including the bias node. It ends with the output layer, where the number of nodes represents the number of possible outcomes and is therefore called the output layer. There is some computations between each of the layers to generate as much information as possible. The lines in the diagram represent the weights between each layer. The neural network's purpose is to find the combination of weights that give the best possible prediction.

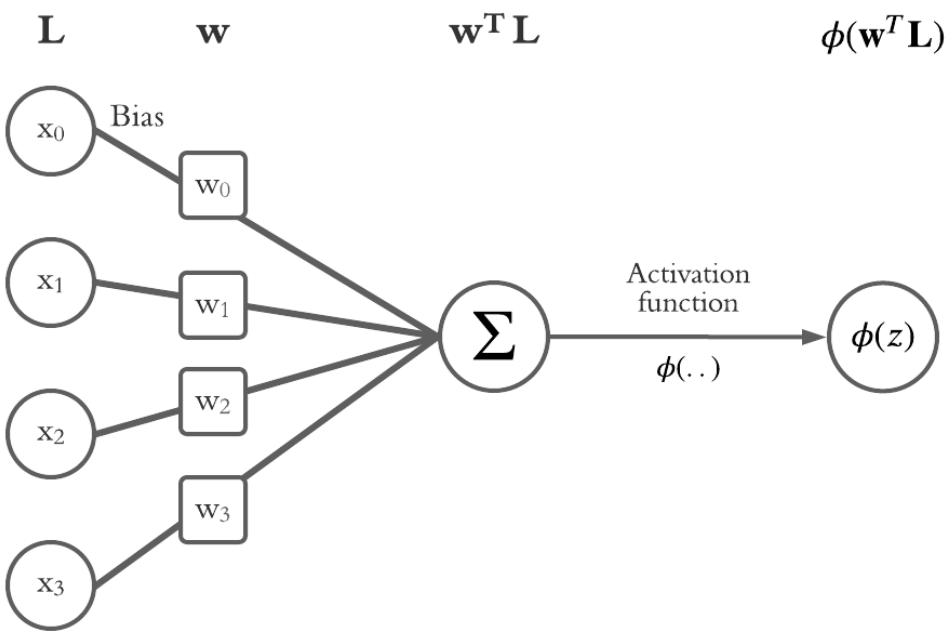


Figure 2.11: Example of a artificial neuron, this figure represent the red markings in Figure 2.10

The data fed into the computer is the starting information in the node and will be called L. Here x_0, x_1, x_2, x_3 will represent the four nodes in the input layer. x_0 is called the bias and is not gathered from our data, but is added in all layers. The bias allows for the activation function to be shifted in one direction or the other. This will help move the decision boundary and make it possible to change the boundary away from the origin, which it could not have done without the bias. The rest of the input nodes are given from our data.

Each of the nodes has weights w corresponding to them, these weights are the trainable parameters that is being optimised to get the model to perform as good as possible. The optimising of the weights happens during the back propagation, explained later in this chapter. The weights are used to calculate the next layers input. The input layer $\mathbf{L}$, and the transposed weights $\mathbf{w^T}$ could be written as:

$$\mathbf{w^T} = \begin{bmatrix} w_0 & w_1 & w_2 & w_3 \end{bmatrix} \text{ and } \mathbf{L} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Each node is calculated by summing these two matrices as follows:

$$\mathbf{z} = w_0 \cdot x_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 = \mathbf{w^T} \cdot \mathbf{L} \tag{2.12}$$

A visualization of this calculation can be seen in Figure 2.11, and what parts of the network going into one calculation marked red in Figure 2.10.

The sum is fed into an activation function ($\phi(z)$), which will add non-linearity to the network. Activation functions will be explained later in this chapter. The activation will give us an output that will be used as input in one of the following layer nodes. This process is marked in red in figure 2.10. The process will be done for both of the bottom nodes in the hidden layer, and a new bias is added to the hidden layer before the same calculation from the hidden layer to the output layer is computed [41].

The previous example shows how one of the nodes in the next layer is computed. For the rest of this thesis, all of the nodes in the next layer will be calculated at the same time, combining all the matrices in one larger matrix calculation. A formula can be seen below, where the bias is not implemented as x_0 it is added separately, both ways can be used to describe the calculation.

$$\phi(\mathbf{w^T} \cdot \mathbf{L}) = \phi(\mathbf{z}) \tag{2.13}$$

$$\phi\left(\begin{bmatrix} w_{00} & w_{01} & w_{13} & \dots & w_{0n} \\ w_{10} & w_{11} & w_{23} & \dots & w_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{m0} & w_{m1} & w_{m2} & \dots & w_{mn} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}\right) = \phi\left(\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}\right) \tag{2.14}$$

Here $\mathbf{z}$ represents the nodes in the next layer, where Z_1 is the first, Z_2 the second, and so on... x_1 and x_2 are the first and second node in the current layer. X_0 represents the bias, and $w_{00}, ..., w_{0n}$ its weights. $\phi(..)$ is the activation function.

2.5.2 Activation functions

An activation function is added to the end of each layers calculation, deciding the next layers input. It takes the information from the previous layer and decides what to forward into the next layer's nodes. One of the most significant points of having an activation function is that it adds a non-linearity to the network, allowing the network to learn much more complex tasks. Converting **z** too different functions, using the non linear functions. Without the activation functions, the entire network could be refactored down to a simple linear operation. How the activation functions add that non-linearity depends on the activation functions. They have distinct advantages and disadvantages. In this thesis there are mainly focuses on two activation functions; the Rectified Linear Units function (ReLU) and the sigmoid activation function.

Sigmoid

Definition 2.5.1. Sigmoid activation function definition [42] [43]:

$$\phi_{sigmoid}(z) = \frac{1}{1 + exp(-z)} \tag{2.15}$$

Definition 2.5.2. Derived sigmoid activation function definition:

$$\frac{\mathrm{d}\phi_{sig}}{\mathrm{d}z} = \frac{exp(-z)}{(1 + exp(-z))^2} \tag{2.16}$$

The sigmoid function is a smooth, continuous, S-shaped function. Making both the sigmoid function and its derived continuously differential. This makes the computation simpler which can help in the back-propagation, explained later in the chapter. The function and its derivative can be seen in figure 2.12.

When training the weights, the derivative of the activation function is used. This makes it possible to encounter what is known as the vanishing gradient problem when the derivative function is small. The vanishing gradient problem occurs during the back-propagation, which will be discussed later [44]. Many algorithms have started to use activation functions where the derived function does not set to approximately zero to avoid this problem.

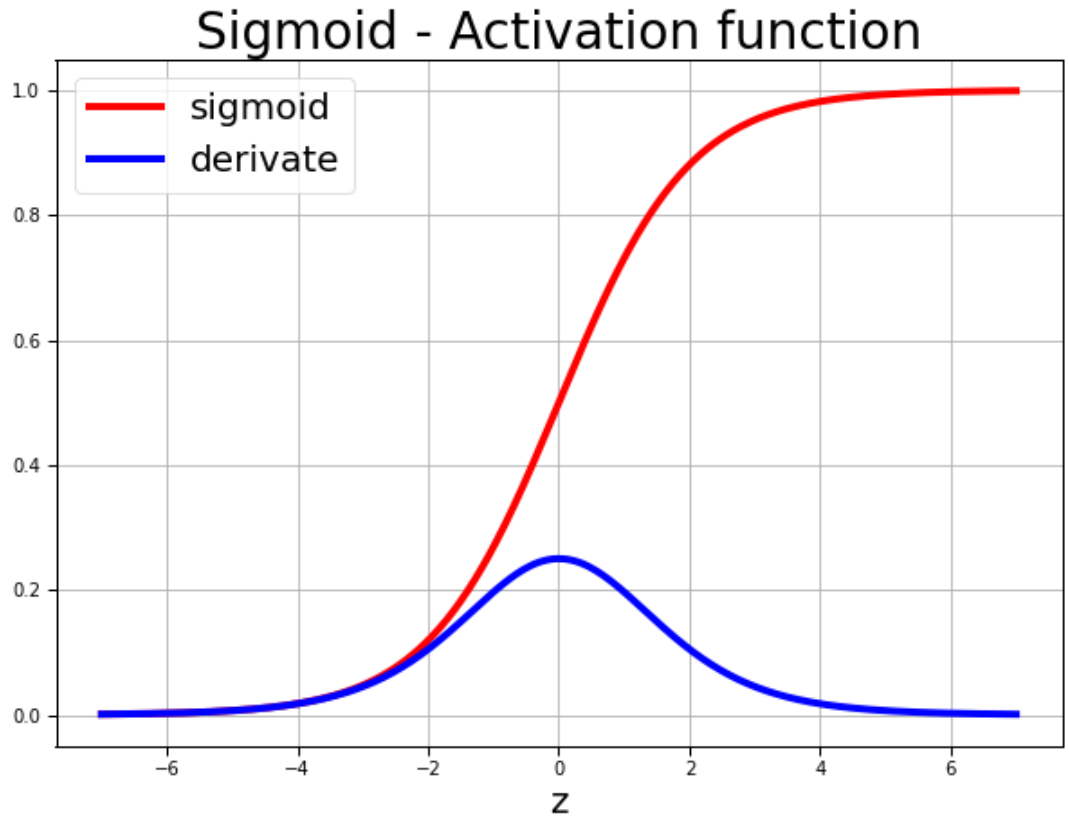


Figure 2.12: In red is the Sigmoid activation function, and its derived function in orange

Rectified Linear Units

The last activation function used is Rectified Linear Units, also known as ReLU. It is the most used activation function in machine learning [45]. And is only used in the hidden layers of neural networks.

Definition 2.5.3. ReLUs definition [42][46]:

$$\phi_{ReLU}(z) = max(0, z) \tag{2.17}$$

Definition 2.5.4. ReLUs derived activation function:

$$\frac{\mathrm{d}\phi(z)}{\mathrm{d}z} = \begin{cases} 0, & \text{for z} < 0 \\ 1, & \text{for z} \geq 0 \end{cases} \tag{2.18}$$

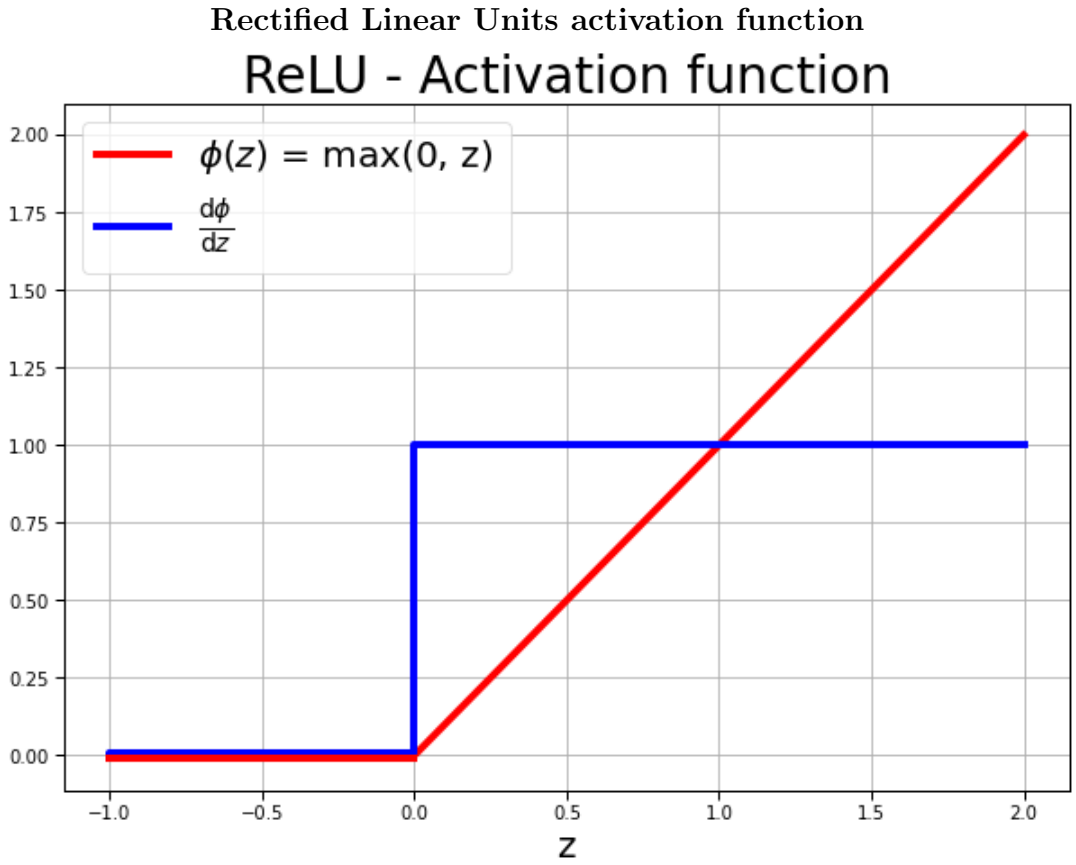


Figure 2.13: ReLU activation function, with its derivate

ReLU is also a non-linear function but it is better suited for back-propagating than the sigmoid (or tanh) activation functions. It was found to increase the learning rate of the weights compared to many of the S-shaped activation functions. The problem with the S-shaped activation functions, such as sigmoid is that it derivate of the function almost go to zero when Z is either really large or small. The derivate is used in the back-propagation, training the weights, and this can lead to dead neurons, where the weights do not improve or update. The ReLU derivate is one for all z-values above zero, solving this problem.

The disadvantage of using ReLU is that it is not differentiable at zero. The function is unbounded. The dead neuron caused by values below zero can lead to the neurons not getting updated during the backpropagation. This is called the dying ReLU problem [46]. The dead neurons can be avoided by using a reduced learning rate and a bias, or using an offspring of the ReLU function such as the "leaky ReLU" [47]. Leaky ReLU is an outspring from the ReLU. Instead of setting

values to zero, it adds a mild slope on the negative values of z, keeping the same advantages as ReLU but being updated in the backpropagation. The disadvantage is that it lets large negative values still affect the predictions.

2.5.3 Loss functions

The loss function tells us how far the predicted output is from the base truth. There are different ways to compute the loss, and therefore multiple loss functions. Some of the most common ones are using the mean absolute error (MAE) or the root means squared error (RMSE) [42]. It is possible to add in regularization similar to the normalization in the HOG chapter to generalize the data. The cost function gives the sum across several samples from the loss functions. The target is to minimize the cost function since this will give the lowest sum of errors. This is done by using the cost function gradient to find its minimum using gradient descent.
The function for the root mean squared error is:

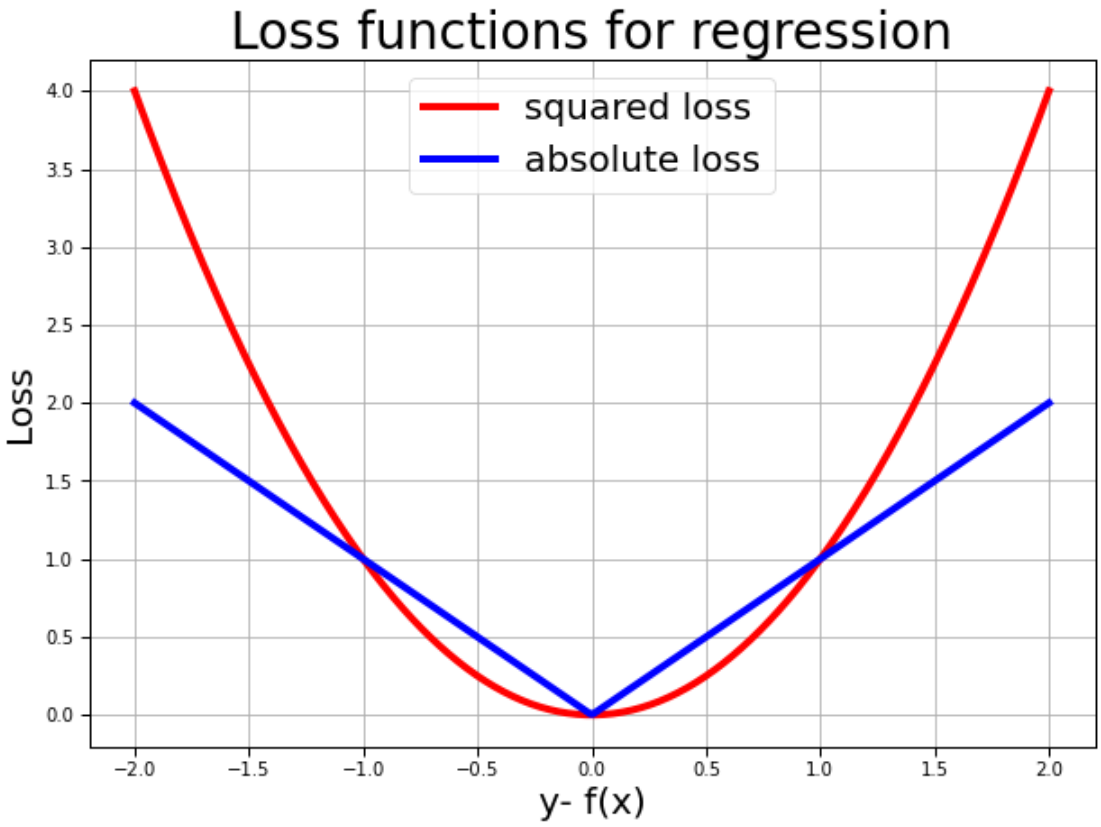


Figure 2.14: Loss functions, where the square loss is marked in red and the absolute loss in blue. Showing how the loss is calculated depending on far the prediction is from the ground truth.

$$J(w) = \sqrt{\frac{\sum_n (y^{(n)} - \phi(z)^{(n)})^2}{N}} \tag{2.19}$$

Here *J(w)* is the cost function, depending on the weights, y is the base truth, $\phi(z)$ is the activation function of the net output, and N is the total number[48].

$$w := w + \Delta w \tag{2.20}$$

The weight is updated incrementally for each of the epochs in the process.

$$\Delta w = -\eta \frac{\partial J}{\partial w_j} \tag{2.21}$$

In what direction the weights should be updated is determined by using the negative cost function gradient. This value will always look for the weight direction that decreases the cost the most. How much the weights learn for each epoch is controlled by a learning rate (η). A too big learning rate can lead to "uncontrolled" big updates and weights never settling towards a minimum. At the same time, a too-small weight update can lead to the weights settling too fast and maybe just finding a local minimum.

This process shows only how one neuron is updated. With multiple layers, the weights are updated from the output layer of the neural network and working their way back into the input layer. An example is shown in Figure 2.15. This is called back-propagation and is how all of the layers get updated in a neural network [49].

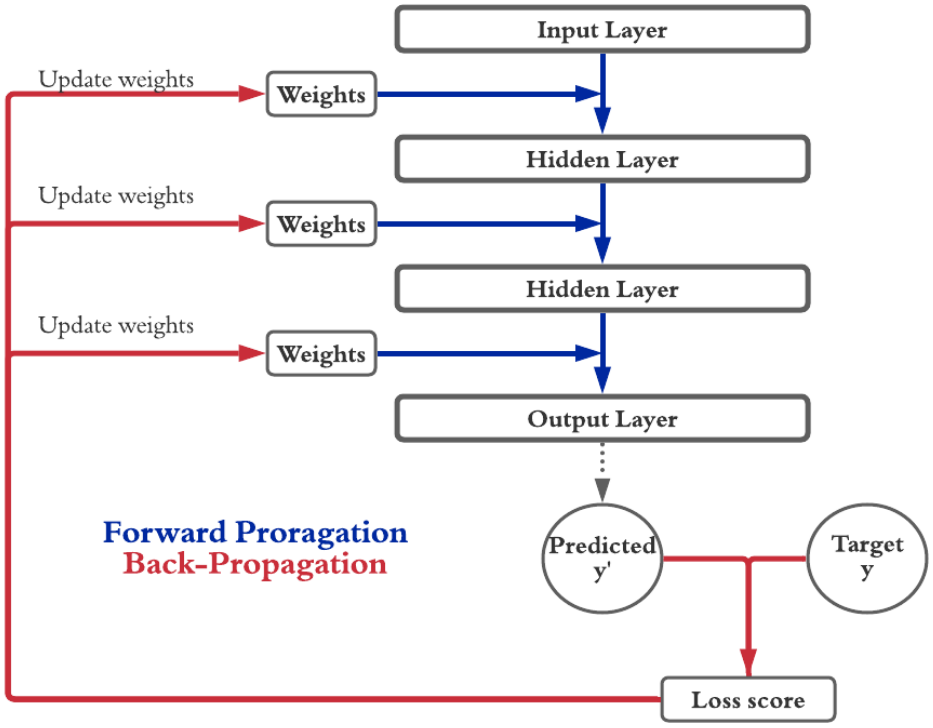


Figure 2.15: A figure visualizing the total process of forward and backward propagation. The blue arrows symbolizes the forward propagation and thee red arrows the back propagation

2.6 Convolutional Neural Network

The convolutional neural network is the most used method for machine learning on images and has become one of the most used techniques within deep learning [50]. A convolutional neural network (CNN), also known as ConvNet, is a network containing the same principles as the deep neural network, but its architecture is different. CNN's are usually used on data with a grid-like topology, such as a digital image. It then uses convolutional matrix transformations on the pictures to gather patterns and structures in the image. This matrix transformation is what characterizes the neural network and where it has its name from. What type of convolutional transformation used can vary, and the structure in the network can be varied. CNN's are usually built up of three main layers; a convolutional layer, a pooling layer, and a fully connected layer. The critical factor with a convolutional network is that the convolutions make it possible to catch up on the spatial relationship between pixels/data [51], making up an image. This chapter will introduce the matrix transformations used in the different layers, how they are used, and the network structure. For this thesis, CNN will be used as semantic segmentation. Meaning it will label each pixel in the image.

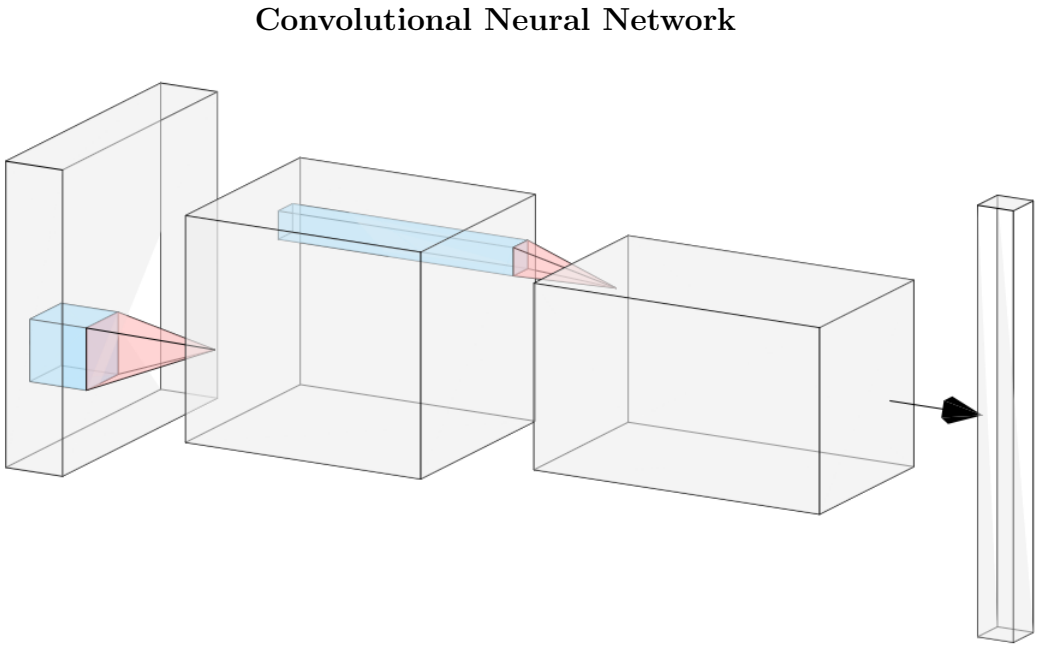


Figure 2.16: Example of a Convolutional neural network diagram structure created by LeNail visualization tool, 2019 [52] [53]

In this thesis, multiple images will serve as input to the model. A multidimensional image contains multiple layers of information. These data structures will be called "tensors." A tensor represents a multidimensional vector space that can hold information about both the image shape, which is the pixel-height $\times$ pixel-width, and the number of layers/dimensions. An image containing four layers, with the image height equal to 100 pixels and width equal to 200 pixels, can be written as a tensor with the shape (100, 200, 4). All of the images are digital images and will be called images for the rest of the thesis.

There are three main types of layers in the convolutional network; i) the convolutional layer, holds the weights of the filter and are responsible for learning the patterns; ii) the pooling layer; responsible for smoothing or sharpening of the image; and iii) the fully connected layer; which is often used in the end to make a prediction. In this thesis there is not used a fully connected layer in the end, instead, the data are transposed back to their original shape.

The neurons in a CNN have their weights built differently from the models discussed in the deep learning chapter. This is to capture the distance between the pixels and reduce the total number of trainable parameters, so the computational less expensive. Instead of putting the weights into an array, they are placed in a matrix and uses convolutions transformations to catch the changes in the data. This can be followed by a pooling layer, which uses a form of non-linear down-sampling, progressively reducing the spatial size of the representation, thus reducing the computation and parameters in the network.

Between the convolutional layer and the pooling layer, there is an activation function. The activation functions used in this thesis are the ReLU and Sigmoid, which are explained in the previous chapter. The models contain multiple layers with convolutions and pooling. Adding more layers makes the network deeper and detects more complex patterns in the data, making room for over-fitting.

2.6.1 Filter operations

Before the layers are explained, the thesis will take a look into the filter operations that can be used before the CNN. They are usually used for image sharpening, image smoothing, edge detection, and generally enhancing the image [54]. This is done using an iterating filter. This filter iterates over the image and performs a matrix transformation creating a new image. What features get enhanced or smoothed out depends on what type of filter that iterates over the image. This type of filter operation can be compared to the calculation of HOG gradients,

explained earlier in this chapter. The iterating filter can be thought of as a sliding window, sliding over the image and taking the dot product between the sliding window and the values under it.

Figure 2.18 shows one step of the calculation. The function in the figure can do multiple different operations depending on the size of the different weights in the filter. If you choose a filter like the one in the HOG chapter, you will enhance the edges in the image. An example of these filter types can be seen to the left in the figure below. These filters are called Sobel filters and are used in edge detection [54]. The same is done for the convolutional layer, however, in the convolutional layers, the weights are updated in the backpropagation. Which will be explained later in this chapter.

Horizontal

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

Vertical

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

Mean

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Figure 2.17: An illustration of different filters used in filter operations, the two furthest to the left can be used for edge enhancing, while the right one for smoothing out the image.

The edge detection filter will give a high value if there is a big intensity difference between the pixels on the opposite sides of the center pixel. This will enhance the area with high-intensity changes, and therefore enhance the edges since edges can be seen as intensity changes in an image. The two matrices to the left in the figure above enhance the edges in their respective direction. If you want the total edge, you can add the two new matrices created on top of each other.

Another filter operation often used is smoothing. These filters can be seen to the right in figure 2.17. This filter will take the average of the center pixel and its eight surrounding neighboring pixels. This will make the image more toned down and smoothed since the high intensities will get scaled down by their surrounding lower values. The lower intensities will increase from their higher intensity neighboring pixels. Creating the span of intensity smaller and also reducing the amount of noise. After the filter operations, they are fed into the neural network.

2.6.2 The Convolutional layer

The convolutional layer is essential and is responsible for most of the computational cost in the network [55]. These layers use a dot product between a matrix with trainable parameters/weights and the receptive part of the image. The weights are stored in a grid-shaped order and called the kernel. The kernel moves across the image, calculating the dot product with each receptive part of the image until the entire image is calculated. The number of weights is dependent on the rank of the image and how many parameters there are in one filter.

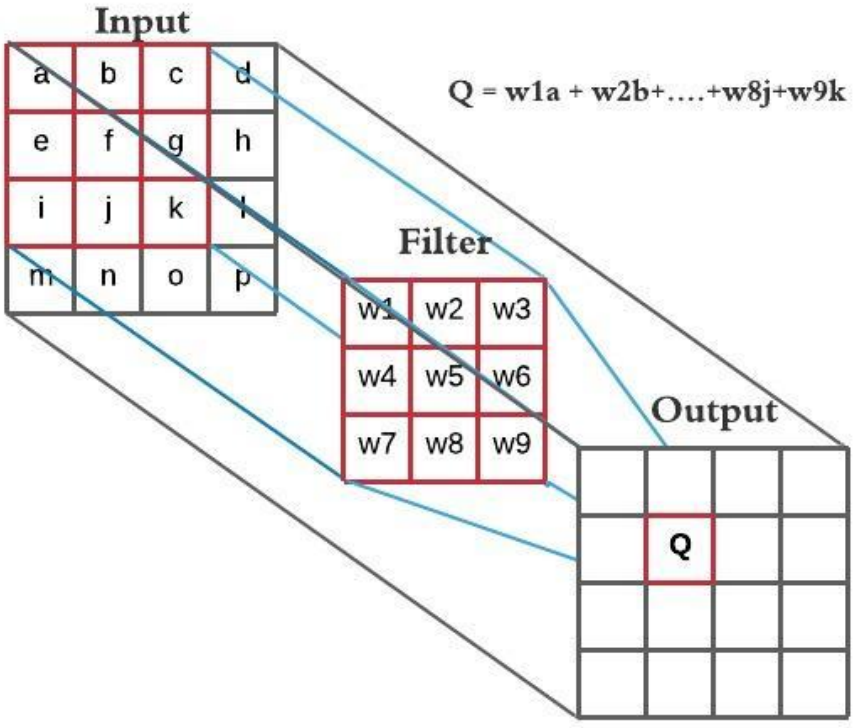


Figure 2.18: An example of a convolutional calculation, the layer to the left is the input layer, the middle layer is the filter containing all the weights W1-W9, the output layer shows the output Q given the formula in the upper-right corner

The kernel matrix contains all the trainable parameters, the weights. These weights get trained to recognize the image patterns. The kernel matrix iterates

over the image with the same depth/number of channels as the image. This means that if you have a filter with the size 3 x 3 and a rank of 4 with a bias, there will be in total 37 trainable parameters ((3x3) x 4 + 1) for one layer. The calculation between a rank one image and its weight can be seen in Figure 2.18. After the convolutional layer, the information is fed through an activation function. In this thesis, it is used ReLU in the network and a sigmoidal function for the output. Both are explained earlier in the Deep learning chapter. After the convolutional layer, the data is fed into a pooling layer.

2.6.3 Downsampling

Pooling layer

The pooling layer is the second and last layer discussed in this thesis and is one of the building blocks in CNN. The pooling layers task is to reduce the dimensions of the feature maps [56]. This means that it is trying to summarise the image's pixel values in a smaller dimension space. The pooling operation uses the same type of iteration kernel over the image as the other layers and filter operations. Different types of pooling operations could be used, but the most common functions are max-, min-, and mean-pooling. It is possible to run a CNN without using down- and up-sampling. However, using up and downsampling will reduce the computational cost. The reason why is visualized and explained in the segmentation chapter. The reduced computational cost makes it possible to use more layers and deeper structures and hopefully detect more complex patterns.

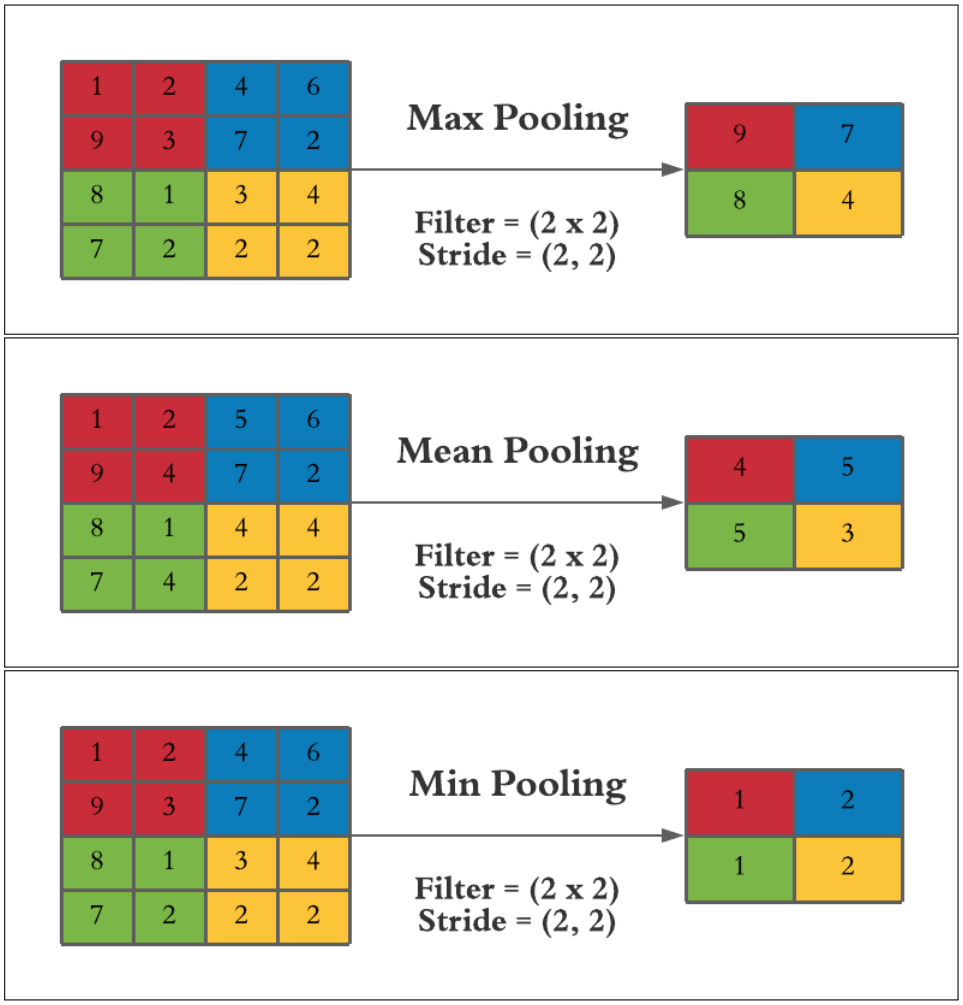


Figure 2.19: A visualization of the three common pooling operations

The pooling iterates over the image. For the max-pooling, the image extracts the maximum value for each filter operation. Then it iterates to the next value, with a given stride. The stride decides how close each operation should be to the other. With a more significant stride, there will be fewer iterations and a bigger down-scaling. The min-pooling extracts the minimum value for each operation, and the mean-pooling gathers the mean for each process, as you can see in Figure 2.19. The size of the image after the down-scaling is given from the definition2.6.1.

Definition 2.6.1. The output shape of an image after a pooling operation can be illustrated as:

$$\left\lfloor \frac{(n_h - f + 1)}{s} \right\rfloor \times \left\lfloor \frac{(n_w - f + 1)}{s} \right\rfloor \times n_c \tag{2.22}$$

Here, n is the image, where n_h is the height, n_w the width and n_c the number of channels. f is the filter size and is assumed to be square, and s is the stride of the iteration.

The stride, filter size, and padding are what decides the image output shape. The padding adds information to the edges of the image to control the dimension of the image and will be explained later in this chapter. The task of down-sampling is to keep the information while reducing the image. Earlier research has shown that there can occur discarding of the high-frequency information when down-sampling [57]. Therefore it is essential to find the best possible way to scale down the data. A second way is to use dilated kernels.

Dilated Kernel

A dilated kernel increases the size of the filter without increasing the number of parameters of the filter. The advantage of using a dilated kernel compared with a pooling layer is the possibility to catch up with new spatial relationships, that can go over further distances.

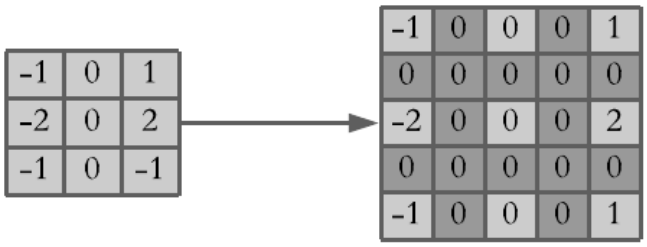


Figure 2.20: An illustration of how a dilated kernel can be extracted from a regular kernel

Dilated Convolutional process

The process of implementing dilated convolutional process has been used by several others, including in medical image segmentation [58] The dilation process works in the same way as the convolutional layer, but here there is added spatial distance to the weights, shown in figure 2.21. The similarities between the dilation and convolutional operation can be seen by looking into the computation of the two of them, shown in the Definition (2.6.2).

Definition 2.6.2. Let $F : \mathbb{Z}^2 \to \mathbb{R}$ be a discrete function. $\Omega_r = [-r, r]^2 \cup \mathbb{Z}^2$ and let $k : \Omega_r$ be a discrete filter of size $(2r + 1)^2$. The discrete convolution operator $*$ can be defined as [59]:

$$(F * k)(p) = \sum_{s+t=p} F(s)k(t) \tag{2.23}$$

Here are the following generalization operator for dilation factor, using l as the dilation factor and $_l$ be defined as:

$$(F *_l k)(p) = \sum_{s+lt=p} F(s)k(t) \tag{2.24}$$

The difference between the computations is the dilated term "l" is added.

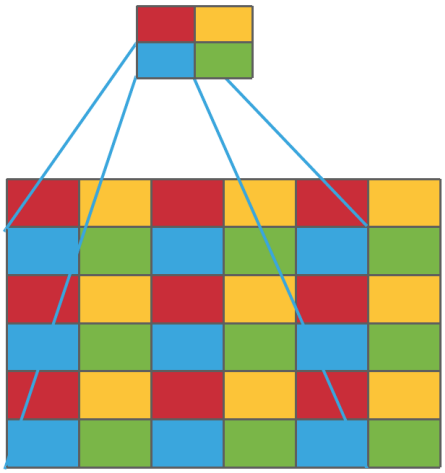


Figure 2.21: An illustration of how a dilated kernel gathers information in an image with a stride of one and a dilation rate = 2×2

2.6.4 Upsampling

After using downsampling, an up-sampling has to be used to get the input and output data on the same form. This needs to be done for the semantic segmentation, where the task is to label each pixel of the image. For image classification the output from the downsampling could be flatten and feed into a dense layer. The downsampling frees up memory, since the convolutional layer works on a smaller set of the data with fewer parameters for each downsampling, and the computational cost is reduced.

There are several up-sampling methods. The one used in this thesis is transposed convolution layer and can also be found under the name "up convolutions" or "upconv" but is also often confused with "deconvolution" in other papers, which is miss-leading [60]. Deconv refers to when you are trying to recreate the exact image after the downsampling, which will not leave room for improving the weights. While the upconv uses convolutions that makes room for learning the weights.

The difference between upsampling and up convolutional layers are that up-sampling only uses simple operation comparing it to its nearest neighbor. An example can be seen in the figure below. At the same time, the transposed convolution uses more convolutions and more computation. The advantage of upsampling is that it cost less computational power.

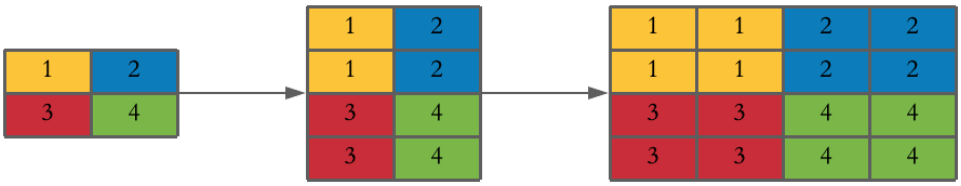


Figure 2.22: An illustration of how the upsampling works, where it first doubles the number of rows then the column, making the matrix 2 times bigger

The convolutional transpose uses both upsampling then convolutional calculations to get the best estimate when scaling up the data. The upconv adds zero to the matrix first to get the desired output shape.

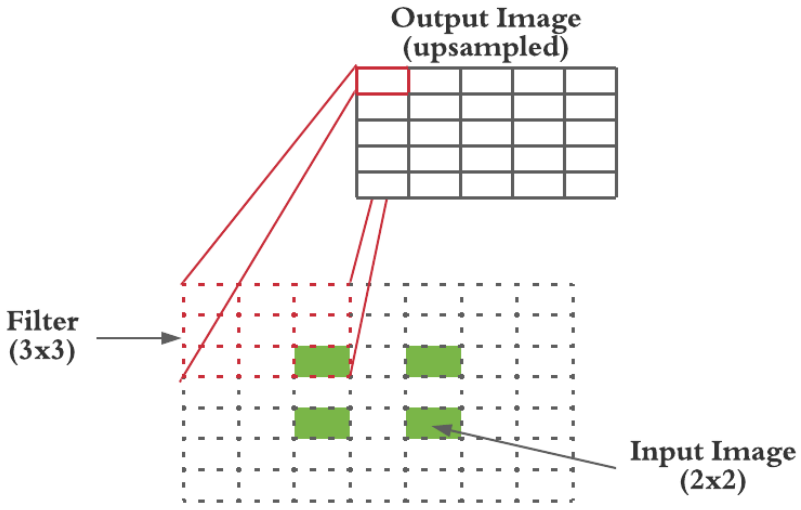


Figure 2.23: An illustration of how the transposed convolutional layer works, here the green squares are the input information. The rest of the matrix is zeros, to get the desired output shape. The red square represents the filter and the output matrix is the output after the transposed convolutional, illustrating the change in dimensions

With this information, it is possible to understand how the structure of the U-Net, and how the semantic segmentation works.

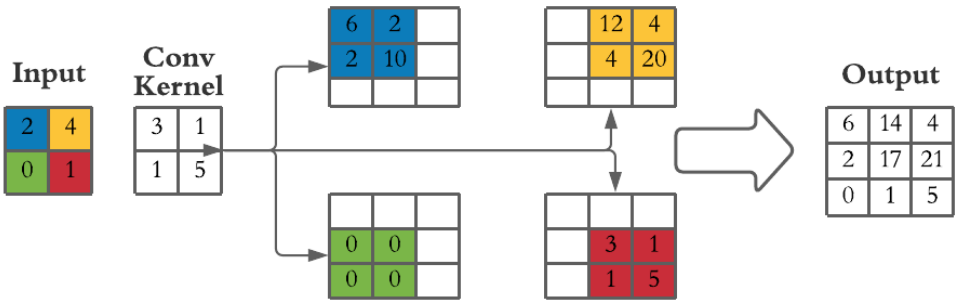


Figure 2.24: An illustration on how the transpose calculates the output matricies, the figure uses a padding = one and a stride = (2,2)

The figure above shows how the output image is calculated and how the images are increased in dimensions.

2.7 U-Net

2.7.1 Semantic image segmentation

The purpose of semantic segmentation is to label all pixels in an image to the correct class. This type of prediction can be called a dense prediction. It is crucial to notice that semantic segmentation does not separate subclass within the class. For this thesis, it will mean it treats cancers and lymphs as the same class. It will only detect if the pixel contains cancer/lymph or not, i.e., binary classification.

This type of segmentation is used everywhere, from autonomous self-driven cars to face detection. To detect pedestrian or to unlock your phone. In this thesis the segmentation is done using a U-Net.

This convolutional model architecture has the shape of the letter "U", because of the downsampling followed by an upsampling with the help of the skip-connections, which is the reason it is named "U-Net." Figure 2.25 shows an example of the "U" shape.

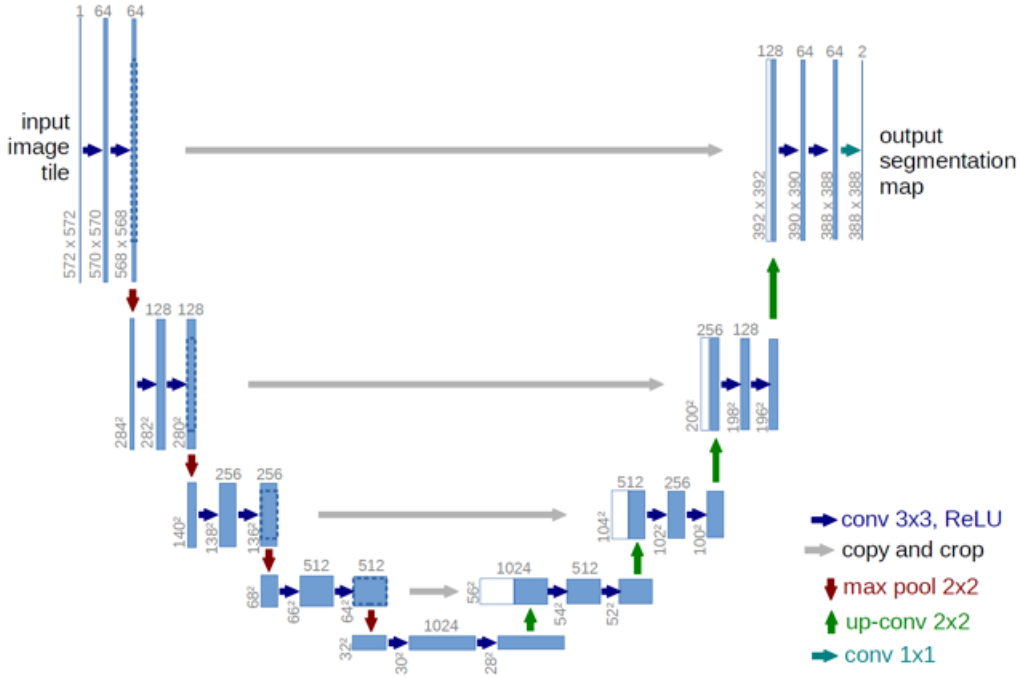


Figure 2.25: The image shows an example of a U-Nets structure [61]

2.7.2 Skip connections

U-Net is a type of CNN containing both convolutional up and down-scaling, which characterizes this model. An example of the structure can be seen in Figure 2.25. The idea that makes the U-Net different from other up and down-scaling models is that the output mapping is directly dependant on the input space. This means the up-scaling is dependant on the down-scaling, using a "skip"-connection.

The skip connection is used to carry information to other parts of the network. The means the information is not used in the next step, and the information skips on til another part of the network. Therefore a "skip connection". The skip connections allow for a different path for the gradients. What is special for the U-Net is the long symmetrical skip connections that can bee seen as the grey arrows from the encoder side to the decoder side of Figure 2.25.

The purpose of skipping the layers, is to reduce the number of layers to train. This speeds up the learning, and reduces the chance for a vanishing gradient problem because of the fewer numbers of layers to back propagate through.

Chapter 3

Material and Method

The goals of this master thesis are to (I) find out if new image representation can improve the cancer segmentation and (II) see how the predictions are affected using them. It was hypothesized that adding more channels could improve the model and performance because of the more extensive data. This chapter will give an overview of the approach taken when creating the new image representations, what type of data, prepossessing, software, and models used.

This chapter will use the same notations and definitions as the ones applied in Chapter 2.

3.1 Software

In this thesis, different modules and coding languages were implemented. The scripts are written in Python, which is an open software programming language available for everyone. The models parameters are stored JSON files. While the data, models, and predictions are in a hierarchical data format, also known as hdf5 or h5 -files. Both JSON and H5 files will be explained later in this chapter. Python contains multiple modules that can easily be integrated and let the system run more effectively [62]. The Deoxys framework, created by Ngoc Hyuhn, has been used to run the models [63] [64]. Orion is an external server and has been used to speed up the processes [65]. Both the Orion and Deoxys will be introduced in the following sub-chapters.

Scikit-learn, often called sklearn, is an additional package containing many tools regarding preprocessing and machine learning [66]. Skimage is a separate module, which enhances the functionalities of sklearn. This module includes the transformation function for both the LBP and HOG [67] [68]. Matplotlib is a library for creating static, animated, and interactive visualizations in Python [69] and visualizes the different images and models. Numpy and Pandas are used to structure the data and comparing the models [70] [71]. Following below is a table of versions of the various modules used for preprocessing the data.

Table 3.1: Table over the different modules versions used in this thesis.

Module	Version
Python	3.7.9
Numpy	1.17.0
Pandas	1.1.3
Matplotlib	3.3.2
seaborn	0.11.0
H5py	2.10.0
Sklearn	0.23.2
Skimage	0.17.2
Tensorflow	2.3.1
Keras	2.3.0

3.2 Data-structure

3.2.1 Hierarchical data format 5

All of the data in this thesis is stored in a hierarchical data format. This thesis uses the fifth version, known as hdf5 or h5. This format can keep huge, complex, and heterogeneous data. This is done by creating structures within the files, which can be thought of as a "file directory." This leaves room for creating organized designs from big datasets and is used for storing the dataset, predictions, and models in this thesis.

3.2.2 The dataset

The dataset used in this thesis combines CT and PET images and their segmentation masks. The data is collected from 197 patients that underwent treatments at the Norwegian Radium Hospital. The segmentation masks are drawn by radiologists at the hospital and will be used as the target values when predicting.

The data set is split into three groups for deep learning. The three sets are; (I) a training set, which will train our network; (II) a validation set to validate our models; and (III) a test set for testing the results from the model [16]. The patients are grouped as follows:

Table 3.2: The number of patients in each dataset. The same split is used in both Moe and Hyunh thesis which are using the same dataset [16] [17].

Datasets	No. patients
train	142
val	15
test	40

The sub-datasets are organized by dividing the information into four new keys; (I) input, where the images are stored, (II) target; where the segmentation masks from the radiologist are stored, (III) patient IDX: contains the patient No., indicating which person the slice is from, and (IV) slice IDX, indicating which slice from the number of images taken in a 3d stack of images.

Table 3.3: Overview of the sub-files structure.

Datasets	Shape	Datatype	Content
image	[No. images, x, y, c]	float32	The input images
target	[No. images, x, y, m]	float32	The target masks
patient idx	[No. images]	int32	Patient number
slice idx	[No. images]	int32	Slice index

This thesis uses multiple different datasets. All of them are combinations of the images and image representations presented in the theory section. Each of them follows the same file structure. There have been tested out combinations of the LBP-data, HOG-data, and MED-data. In this thesis, it is assumed that both the PET and CT channel is included if it is not specified anything else. This

means that the "MED" data will consist of both the medical PET and CT images (MEDPET and MEDCT). The same is true for the HOG and LBP datasets.

3.3 Deoxys software

This thesis has been using numerous different models. They all have been tested using the newly developed Deoxys framework, created by Hyunh, as a preparation for her master and further worked on during her master and doctorate [64] [63] [17] The goal of Deoxys is to make it possible to run multiple CNN models in order to find out which model that performs best and use this model further in the research [63]. The framework is built on Keras [72]. Keras is an open-source API written in Python and which can easily be integrated into other programming platforms. Deoxys makes it possible for users to create configuration files, and to choose their model's structure. There are multiple parameters that can be changed in the models structure, such as; loss function, size of the U-Net, number of layers, and many other hyperparameters, which made it possible to increase the amount of simulation in this thesis.

It also makes room for creating your own activation functions, loss functions, etc. Deoxys can be run on both computers but also external GPUs, such as Orion [73]. The minimum software requirement for the Deoxys framework to work correctly are Python 3.7 and Keras 2.3.0 [17].

3.4 Orion cluster

Orion is a server owned by the Norwegian University of Life Science (NMBU) and was used in this thesis. The Orion server is a high-performance computing (HPC) cluster infrastructure available to course participants, master students, doctoral students, and employees at NMBU [73]. Orion is a SLURM-based Linux cluster with high specs. It has 580 CPUs, 7 TB RAM, and 550 TB storage space [65]. This cluster made room for speeding up the simulation process and made the research run more effectively and efficiently.

3.5 Preprocessing

This chapter will look into what preprocessing methods are applied before and after creating the new image representations.

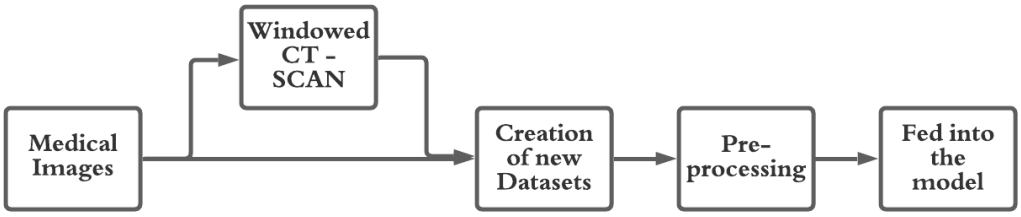


Figure 3.1: A visualization of the preprocessing template.

The medical data were usually not preprocessed before creating new image representations. The preprocessing was usually done after creating the dataset and before entering the model. The only prepossessing tried out before the data creation was windowing on the medical CT scan. Windowing is explained in chapter 2.2. Windowing enhances the desired spectral of the image, and an example of the transformation can be seen below.

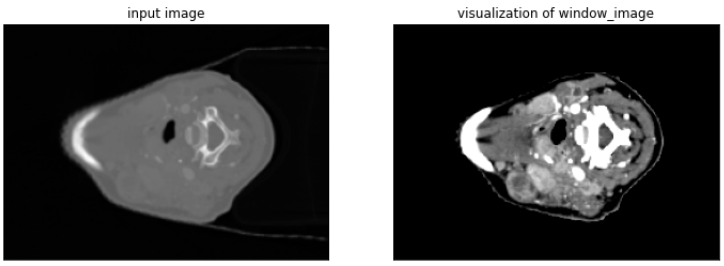


Figure 3.2: A visualization of before and after the windowing operation is used

3.6 Creation of the image representations

This thesis has used two new image representations, HOG, and LBP. The creation of them is explained in the theory chapter. This chapter will define the different parameters considered when creating the images and how they affect them. The various parameters in Table 3.4 are for both the creations of the LBP and HOG image representations.

Table 3.4: An overview of the different parameters taken into consideration creating the HOG and LBP images.

HOG-parameters	**standard size**
Orientations	9
Pixels Per Cell	8x8
Cells Per Block	3x3
Block Norm	L2-Hys
Visualize	True
Transform	False
Feature Vector	True
Multi Channel	None
LBP-parameters	**standard size**
radius	2
number of points	16

3.6.1 HOG-Parameters

There are multiple different HOG parameters. Only the first four in the Table 3.4 have been tested out. The last four are choosing what input and output wanted. (I) the visualization tells us if it should give out the image or not; (II) the transformation tells us if the image input should be normalized before transforming it; (III) the feature vector tells us if the function should give out the vector, something which is not necessary when this thesis is only using the image representations. It has only been used for looking into each of the bins in the histogram.

The last one is (IV) the multi-channel. It tells us whether the input is regarded as a multi-channel image such as a colored image, or as an image with one channel,

for instance, a black and white image. If there is a multi-dimensional image as input, this function will break down the image to a one-channel before running the HOG function. This is done by taking the highest value from each channel in their respective position. Our medical data scan consists of two tracks, the PET and CT. These channels are independent and do not complement each other the same way a colored image does. Therefore it is not possible to treat the PET/CT image as a multi channel image. The HOG function is done on each of the channels separately and then merged afterward.

Orientation

The orientation tells us the number of bins used in each of the histograms, which will determine how detailed each star histogram is.

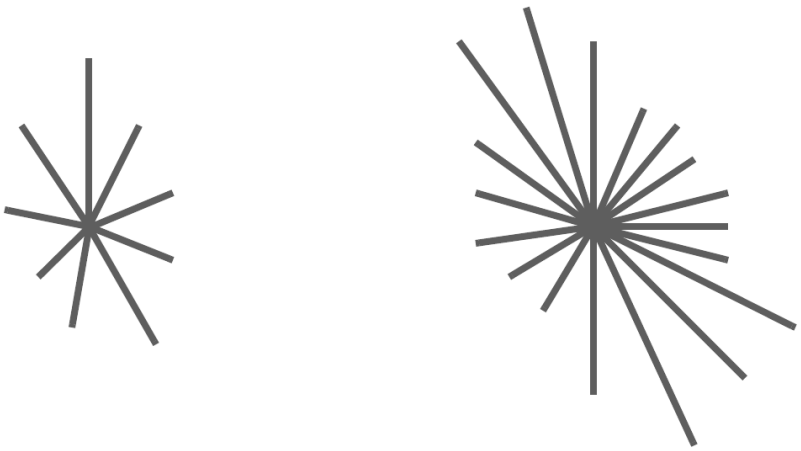
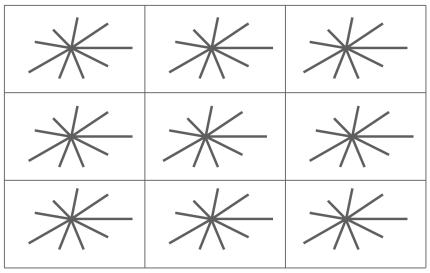


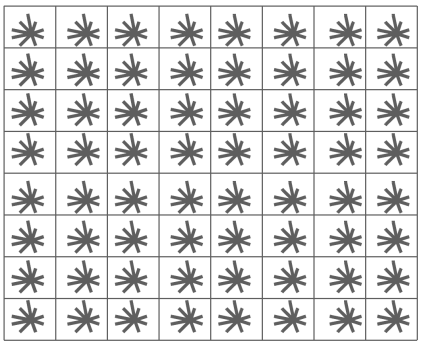
Figure 3.3: A representation showing the difference between using 9 and 18 orientations.

Pixel per cell

The Pixel Per Cell (PPC) parameter decides the size of each descriptor block. This means, if you have a large PPC, there would be a bigger descriptor block, creating bigger cells and decreasing the number of star histograms. This is because each cell contains one histogram. However, with a big descriptor block, there would be more information going into each of the histograms. An example of these two scenarios can be seen in Figure 3.4.



(a) A representation of an image divided into nine (3x3) descriptor-blocks.



(b) A representation of an image divided into 64 (8x8) descriptor-blocks.

Figure 3.4: Representation of how the size of the descriptor-blocks change the number of parameters.

This thesis tests out four different descriptor-blocks sizes; 2x2, 4x4, 8x8, 16x16.

Cells Per Block

Cells Per Block (CPB) controls how many of the surrounding cells are taken into account when normalizing each block. A cell has the same size as a descriptor-block seen from the figure above. If the size of CPB is 1x1, it will normalize each of the cells separately, not taking the neighboring cells into account. Increasing the CPB will increase the number of histograms/adjacent cells that are taken into account when normalizing. If you have a nine bin histogram in each cell and a 3x3 CPB, the normalization will use the surrounding eight neighboring cells. The normalized vector will have a total of 9 $\times$ (3x3) parameters, in total 81 different parameters.

This parameter will decide how much information from the surrounding cells should be captured. The different CPB sizes tested out in this thesis are 1x1, 3x3, and 5x5.

Normalization

There are mainly two different normalizations tested out. These are the L2- and the L2-Hys Normalization. These are explained in Chapter 2.3. Both of them use a squared penalty term, however L2-Hys sets a cap on the normalization, so it cannot go higher than 0.2, forcing it not to weigh some values too high.

Dataset-combinations

In total there have been created 12 HOG-datasets that have been run alone and with the other models.

Table 3.5: An overview of the name and parameters for the different HOG-datasets.

Name	Orientation	pixel per cell	cells per block	Block Norm
HOG(3-4)	9	4x4	3x3	L2-Hys
HOG(3-2)	9	2x2	3x3	L2-Hys
HOG(3-2)-L2	9	2x2	3x3	L2
HOG(3-16)	9	16x16	3x3	L2-Hys
HOG(1x2)	9	2x2	1x1	L2-Hys
HOG(1x4)	9	4x4	1x1	L2-Hys
HOG(1x8)	9	8x8	1x1	L2-Hys
HOG(1x16)	9	16x16	1x1	L2-Hys
HOG(5x2)	9	2x2	5x5	L2-Hys
HOG(5x4)	9	4x4	5x5	L2-Hys
HOG(5x8)	9	8x8	5x5	L2-Hys
HOG(5x16)	9	16x16	5x5	L2-Hys
WindowedHOG	9	8x8	3x3	L2-Hys

3.6.2 LBP-Parameters

There are two parameters tested out creating the LBP image representations in this thesis. These are the radius, controlling the distance between the center pixel and its surrounding pixels, and the number of points, controlling how many different points are taken into each calculation.

Radius and number of points

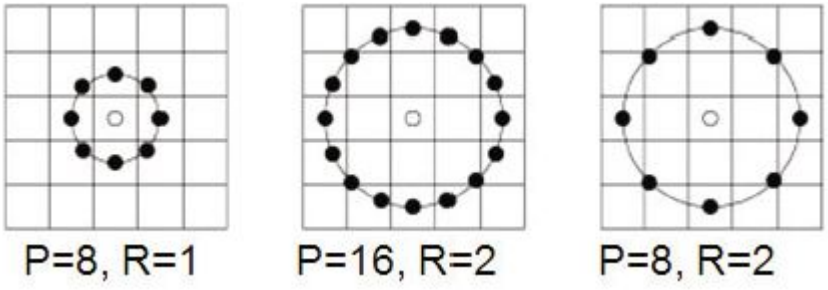


Figure 3.5: An image to show the different parameters used in LBP, Here R represents the radius and P the number of points.

The different radiuses used are one and two, which are the most common ones. Figure 3.5 above shows how the other parameters affect the LBP calculations. The most common number of points is eight times the radius, which is valid for both of the two LBP datasets used in this thesis, as can be seen in the following table.

Table 3.6: An overview of the name and parameters in the different LBP datasets.

Name	Radius	Number of Points
LBP	2	16
LBP1	1	8

3.7 Model Parameters

The primary preprocessing method is augmentation and windowing. Windowing was run on the medical CT-channel and has been tried out both before and after creating the new datasets. The theory is explained in Chapter 2.2.

Data augmentation

Data augmentation creates new slightly modified image representations, using simple transformations of the data. This makes it possible to create more images from the same number of data. This gives a regularisation effect reducing the chance of overfitting. The different methods could be for instance; mirroring the images, use different zooms, tilt and rotate the original images. The augmentations in this thesis consists of multiple factors, but there are mainly used two different augmentations setups in this thesis. The "def" augmentation, also called "A" is provided in a master thesis by earlier master students at NMBU [16][17]. The "A2" augmentation is provided by one of this year's master students, Maria Odegaard, working with the same dataset and optimization of the augmentation.

Table 3.7: A Table showing the different parameters in the two different augmentation setups

Augmentation	Affine Transformations			
Name	Rotation	Zoom	Shift	Flip
Def(A)	90	[0.8 - 1.2]	[10, 10]	0
A2	90	[0.5 - 1.5]	[10, 10]	0

Table 3.8: A table showing the different Points and filter operations parameters in the two augmentation setups

Augmentation	Points and filter operations					
Name	brigthness	contrast	noise variance	noise ch	blur range	blur ch
Def(A)	[0.8 -1.2]	[0.7 -1.3]	0,05	1	[0.5 -1.5]	1
A2	NONE					

When the numbers are in brackets, it symbolizes the upper and lower limit. It is then generated images that are rotated with a random angle between 0-90. There is no difference between the DEF(A) and A2 augmentation regarding the rotation. The zoom, zooms into the images with a random factor drawn from the zoom range. The same is done for the other parameters, augmenting more data.

3.8 Models used for analysis

In this thesis, there were tested out numerous hyperparameters, parameters, and datasets in the model. Where the augmentation, dilated convolutions, and what type of dataset used are the most tested. In this thesis, there have been focused on how the different datasets models perform and compared them against each other, with a main focus on the HOG dataset and whether it adds a benefit to the models' prediction.

All of the data is fed into a model with a U-Net shape and run on the Orion server to speed up the process with the help of the Deoxys framework Deoxys makes it possible to choose numerous different model-parameters, for instance, layer type, loss functions, optimizer, the architecture structure, and the run-time through the number of epochs. It can also generate architectures. There have been tested out two types of architectures, one regular seen in the figure below, and one using dilated kernels, with different dilation rates. The different dilation rates implemented are a 2x2 dilation, 3x3, and 5x5 dilation rate.

The run time is chosen by the number of iterations. However, it is controlled by an early stopping function, the early stopping makes it possible to stop the model earlier if it has not improved over the last number of iteration. The number of iteration before stopping can be chosen.

3.9 Analysis of the model performance

The performance is measured using dice score or F1-score, which for this thesis will be the same calculation. In this thesis, the dice score is used to measure the different models' performances. Then other statistical methods have been applied to find a significant change in the data. From the Deoxys framework, both the dice score per patient and the dice score per slice are given. For measuring the performance of the model, the average dice per patient is used.

Dice score

Definition 3.9.1. Definition of dice score:

$$Dice = \frac{2TP}{2TP + FN + FP} \tag{3.1}$$

Here TP is the True Positive, FN the False Negative, and the FP the False Positive. The terms are all explained in Figure 3.6.

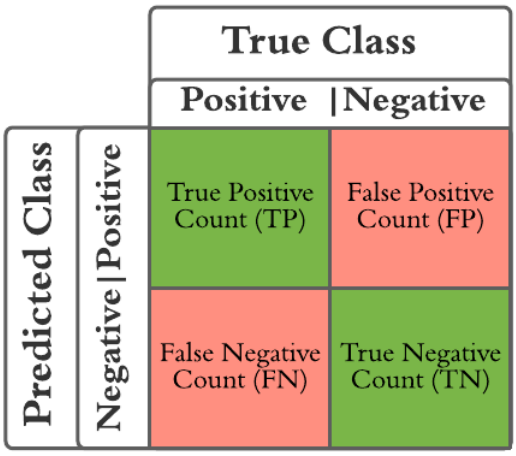


Figure 3.6: A figure showing what is meant of the different false positive terms. Here the true positive and negative are marked in green and the false positive and negative are drawn in red.

The equation takes two times the number of true positives which in this thesis will mean the correctly predicted cancer pixels. Then it is dividing it by the total number of predicted cancer pixels added with the total number of actual pixels containing cancer. This means that if all of the pixels are correctly classified the dice score will be equal to one (2TP / 2TP), if none is classified correctly the dice score equals zero (TP = 0).

3.10 Statistical methods

Multiple tests have been used to find out if there is a significant difference between the performance of models. This thesis looked into the dice score per patient and the dice score per slice to look for substantial differences in the different datasets.

First, tests are performed to see if the datasets are normally distributed. The dice score per patient is the average dice score for all the slices to the patients.

3.10.1 Tests for normality

To test if the data is normally distributed a Q-Q plot is used.

Quantile-quantile-plot

The quantile-quantile plots (Q-Q-plots) helps us visualize if the data is normally distributed. It tests this by taking the quantiles of your data and plotting it against the quantiles of a normal distribution. This is done by taking the lowest quantile of your data and plotting it against the lowest quantile of the normally distributed data. Then the process is repeated for the second-lowest until the highest quantile is plotted. If the plot is a straight line, the quantiles for the data follow identical distributions and are normally distributed.

3.10.2 Parametric tests

The following methods are for testing for a significant difference in results from various models. They all assume a null hypothesis (H_0) that says there is no difference between the models. Then, tests will be performed to prove a significant difference between the models. The parametric tests perform mathematical calculations on datasets assuming the data is normally distributed. One of these tests is the ANOVA test.

ANOVA

Analysis of variance is also known as ANOVA. ANOVA is used to measure variance in data. ANOVA tries to see if there is a significant difference between the means of each dataset. This is done by looking at the ratio of the between-group variance and the within-group variance.

3.10.3 Non-parametric tests

Non-parametric tests are used if the data used are not drawn from a particular distribution, such as a normal distribution. In this thesis, there have been used multiple non-parametric tests. This section looks into different tests, how the math is calculated, and the difference between them. Some differences are that some tests can use multiple datasets, while others are for separating two datasets. The tests are done in python using the Scipy module.

Wilcoxons test

Since the data used in the different datasets are from the same patients, "matched pair", it is possible to use the Wilcoxon signed-rank test. The test looks at the difference between the data and ranks them in order of their difference. This means that the highest differences are weighed the most. After this, all of the ranks get signed depending on if the difference is negative or positive. With the help of a table of critical values for Wilcoxon's Ranks Test and the number of samples (n) it is possible to determine if there is a significant difference. If the sample size is considerable, it could be approximated with z in a normal distribution table and read out if the difference is significant.

Definition 3.10.1. A definition is used to calculate the Z-Score when the sample size is large and can then use a standard table for z -score.

$$Z = \frac{\left|T - \frac{1}{4}n(n+1)\right|}{\sqrt{\frac{n(n+1)(2n+1)}{24}}} \tag{3.2}$$

Here n is the sample size, and T is the sum of the different ranks.

Mann-Whitney U test

Mann-Whitney U test, also known as Wilcoxon Rank Sum test, uses the same principles as the previous method. This method ranks the data depending on the size of the dice score. So the lowest dice scores are valued the smallest, while the highest the most. The test can also handle differences in sample size between the sets, but there will be the same size for all tests in this test. The sums of each

class are then calculated and compared using a Wilcoxon rank-sum test, which will give out the values that it has to be above to see a significant change in the data.

Definition 3.10.2. A definition used to calculate the Z-Score when the sample size is significant, using a normal distribution table for the Mann-Whitney U Test.

$$E(T_1) = \frac{1}{2} n_1 (n_1 + n_2 + 1) \tag{3.3}$$

$$Z = \frac{|T_1 - E(T_1)|}{\sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}}} \tag{3.4}$$

Here T is the sum of the ranks, E(T) is the function taking the sample size of both groups into account, n_1 and n_2 is the sample size for their respective groups.

The common denominator for these two functions is that they use ranked data. The Wilcoxon rank is given from the difference in performance, while the Mann-Whitney is provided from the actual dice-score. Then the sum of the different classes is used in their respective tables. If the sample size is too large for the tables, for instance, the normal distribution can be used when using the dice per slice. They can also only be used to compare a maximum of two datasets. The following method handles several datasets at once.

Kruskal Wallis H test

Kruskal Wallis uses the rank of the datasets, ranking the lowest to the highest. Then it calculates the value H given from the following definition and compares it in the Kruskal Wallis table depending on the number of degrees of freedom.

Definition 3.10.3. Kruskal Wallis definition:

$$H = \frac{12}{N(N+1)}(\sum_{n=i}^{N} \frac{T_i^2}{n_i}) - 3(N+1) \tag{3.5}$$

H is the Kruskal Wallis number. N is the total number of samples, n is the number of samples within each group, and T is the sum of the ranks within each group.

Friedman's test

Friedman does also used ordinal/ranked data. But it ranks each row separately. An example, if you have a row that is [23, 43, 12], the ranked column would now be [2, 3, 1]. This is done for all of the rows. A difference to the Kruskal Wallis definition is that Friedman has to have the same number of samples within each group.

Definition 3.10.4. Friedman test definition:

$$H = \frac{12}{nk(k+1)}(\sum T_i^2 - 3n(k+1)) \tag{3.6}$$

Here H is the output measured against the table, n is the number of samples in the groups, k is the number of groups, and T is the sum for each of the groups.

Both of the H values from the Kruskal Wallis test and Friedman are compared in a table to find significance using the degrees of freedom and alpha. Alpha controls how sure you want to be, the level of significance. Comparing your calculated score with the score from the table, it is possible to see if you can discard the null hypothesis, with a decided level of significance. The degrees of freedom (df) is calculated by taking the number of groups minus one; df = k-1.

Chapter 4

Results

These tests are done with numerous different datasets, augmentations, and hyperparameters. The performance of the models is measured using a dice score per patient for the patients in the validation set. The models have been conducted using the Deoxys framework and the external server "Orion" to speed up the process, both are explained in the method chapter. This chapter will present the results gathered throughout this master period, and what hyperparameters and augmentation affecting the different datasets. The thesis will further explore the other dataset's best model to find structural differences between them and use different statistical methods to look for significant changes between them. Firstly, this section will explain the different datasets. The best models will be compared from section 4.5.

4.1 Datasets

The performance of the models used is shown in the Table 4.1.

Table 4.1 shows that the medical images perform best, and implementing new channels only leads to lower scores. The table also indicates that the HOG and LBP images alone manage to capture a significant amount of the information. It also tells that HOG and LBP reduce the score of the image representations when implemented with the medical data. By implementing multiple image representations, the score decreases further.

Table 4.1: The different models performance, without any augmentation

Data-set	Dice Score
MED	0.60516
HOG	0.58591
LBP	0.55944
MED HOG	0.59737
MED LBP	0.58183
HOG LBP	0.55988
MED HOG LBP	0.56390

4.2 PET and CT

Table 4.2: Table showing what data goes into the different PET and CT models

Datasets	CT	PET	Both
MED	MEDCT	MEDPET	MED
HOG	HOGCT	HOGPET	HOG
MED HOG	MEDCT HOGCT	MEDPET HOGPET	MED HOG

Table 4.2 shows what datasets go into each model. Where the general assumption is a dataset that has not specified what channel uses both the CT and PET-channel.

Table 4.3: The table shows the performance of the models using the CT and PET channel separately, using the default augmentation and windowing (WC = 1073, WW = 200)

Datasets	No Augmentation	Aug	Wind	Wind and Aug
MEDCT	0.42854	0.53450	0.50228	0.54897
MEDCT HOGCT	0.41874	0.51922	0.44423	0.5062
MEDPET	0.58146	0.58717	-	-
MEDPET HOGPET	0.58085	0.58011	-	-

Before going into the different datasets and model parameters, HOG's effect on the PET and CT separately will be analyzed. Table 4.3 shows how the medical data with and without HOG is affected by using augmentation and windowing.

Computational topography

Table 4.3 shows that both the augmentation and windowing increase the performance using only the CT channel. However, implementing the HOGCT decreases the score.

Positron emission tomography

The PET channel is the second and last channel of the medical dataset. Table 4.3 shows the performance of the PET model alone and when the HOG representation is added as an extra channel. It also presents the effect augmentation has on the two different setups. The table shows that the accuracy for the PET dataset is slightly higher than for the CT, be that as it may, the same pattern on implementing HOG can be seen here. HOG decreases the score, but the augmentation also improves this setup.

Comparison

Following in Figure 4.1 and 4.2, two cat-plot showcasing the difference between augmentation and implementation of HOG is shown. Figure 4.1 shows the effect of implementing HOG as an extra channel, while Figure 4.2 shows the effect augmentation has on the performance.

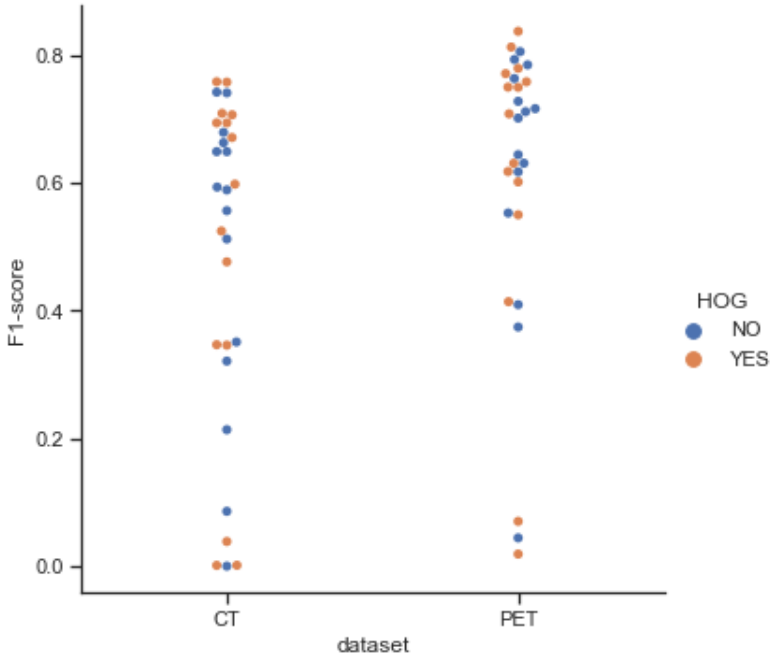


Figure 4.1: Cat plot visualizing the effect implementing HOG has on the two different channels.

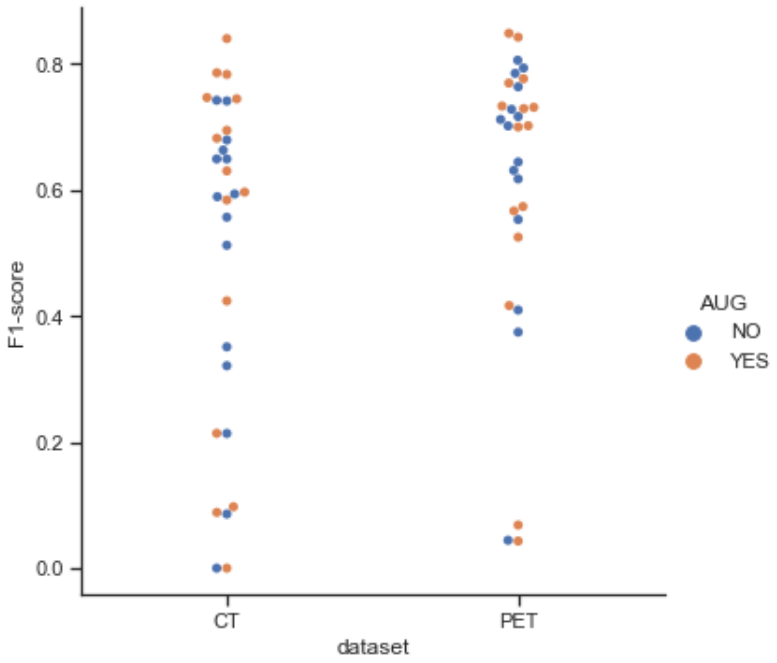


Figure 4.2: Catplot visualision the difference augmentation has on the two channels.

4.3 HOG-hyperparameters

When creating the HOG images, there are multiple parameters taken into account. The main parameters are the size of the descriptor block and the normalization filter. These parameters decide the size of the star histograms in the image and how much of the information of the neighboring pixels it should contain. Following is a table containing the names for the different datasets and which parameters have created them.

Table 4.4: A table showing the name of the most used HOG-models, and what datasets that goes into them.

		Descriptor Blocks			
Normal 2d-unet		2x2	4x4	8x8	16x16
	1x1	HOG1-2	HOG1-4	HOG1-8	HOG1-16
	3x3	HOG3-2	HOG3-4	HOG3-8	HOG3-16
Normalization	5x5	HOG5-2	HOG5-4	HOG5-8	HOG5-16

A visualization of the HOG images can be seen in figure 4.3. The figure shows how the different descriptor block sizes affect how the HOG images appear.

The performance for all the image representation setups shown in Table 4.4 is presented in Table 4.5

Table 4.5: A table presenting the performance for the different HOG-models shown in table 4.4.

		Descriptor Blocks			
Normal 2d-U-Net		2x2	4x4	8x8	16x16
	1x1	0.587916	0.589168	0.563940	0.509215
	3x3	0.605604	0.577251	0.565728	0.520242
Normalization	5x5	0.581331	0.588807	0.570478	0.514236

The table shows that reducing the descriptor block can improve the amount of information gained. HOG3-2 and HOG1-4, and HOG5-4 gave the best predictions. The rest of the thesis will use HOG3-2, also called HOG2.

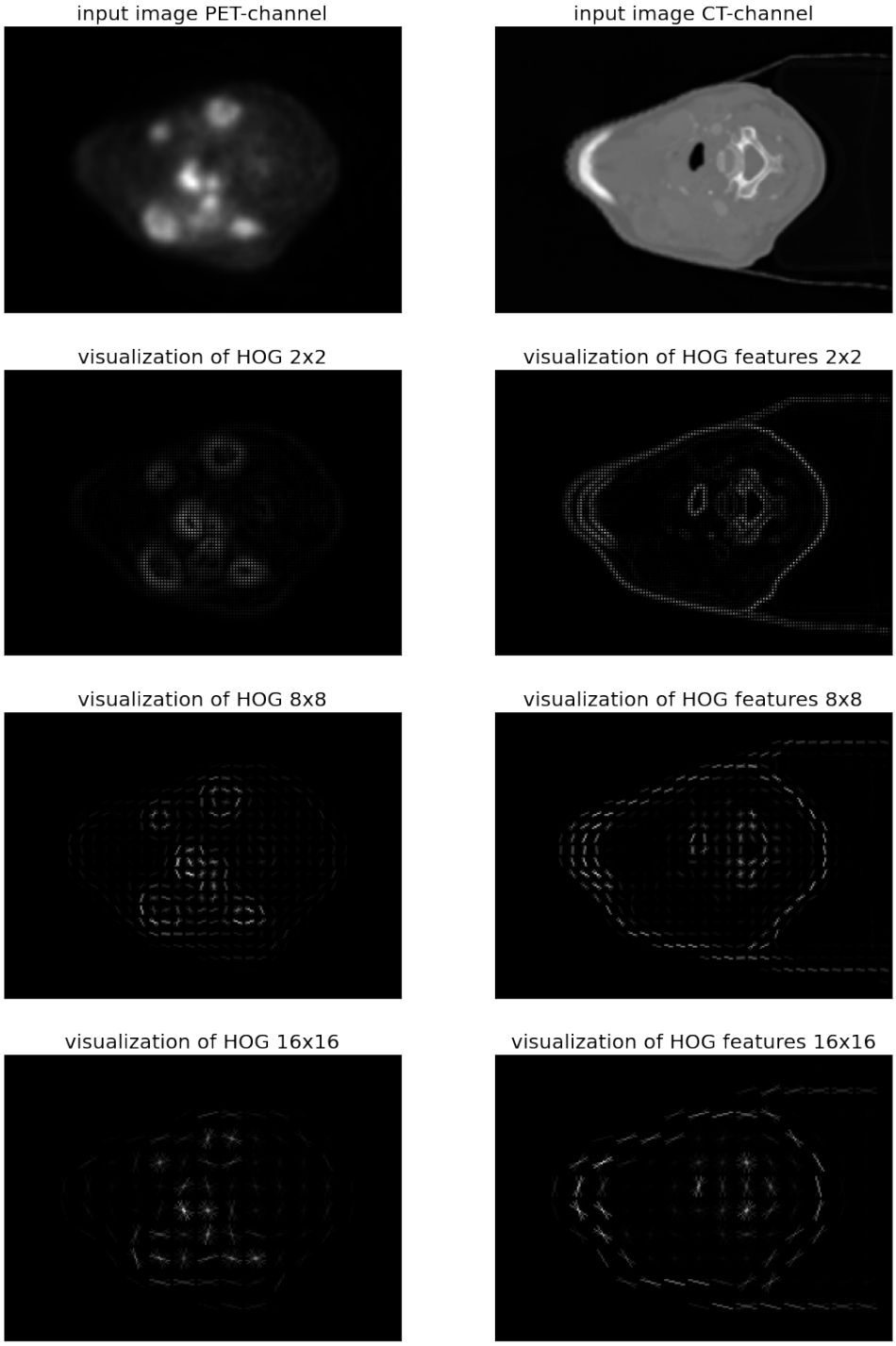


Figure 4.3: Effect of the different descriptor-block sizes

4.4 LBP-hyperparameters

There have been two different setups creating the LBP image representation, Table 4.6 shows the performance of the two different setups, with and without the medical images.

Table 4.6: A table showing the performance of the two LBP setups, with and without the medical images included.

Name	Dice Score
LBP	0.581825
LBP1	0.568232
MED LBP	0.583391
MED LBP1	0.570390

4.5 Model Parameters

4.5.1 Dilation

It has been tested whether the dilation rate could improve the model. Following are three tables with the results from the different dilation rates: (2, 2), (3, 3), and (5, 5). For the rest of the thesis, it will be assumed that the dilation rate is squared and referred to by one number. Example: a dilation rate = 2 equals a dilation rate equal to (2, 2).

Figure 4.4 shows a cat-plot visualizing the performance for five different HOG datasets using three different dilation rates.

Table 4.7: A table showing the results using a dilation rate = 2, using the datasets with the highest prediction from table 4.5.

Table A: Dilation rate: Two

Dilated 2d-U-Net		Descriptor-blocks		
Dilation rate = 2		2x2	4x4	8x8
	1x1		0.603580	
	3x3	0.58836	0.587468	0.58369
Normalization	5x5		0.583908	

Table B: Dilation rate: Three.

Dilated 2d-U-Net		Descriptor-blocks		
Dilation rate = 3		2x2	4x4	8x8
	1x1		0.575948	
	3x3	0.57864	0.563597	0.57445
Normalization	5x5		0.584474	

Table C: Dilation rate: Five.

Dilated 2d-U-Net		Descriptor-blocks		
Dilation rate = 5		2x2	4x4	8x8
	1x1		0.551873	
	3x3	0.51602	0.572577	0.526714
Normalization	5x5		0.563007	

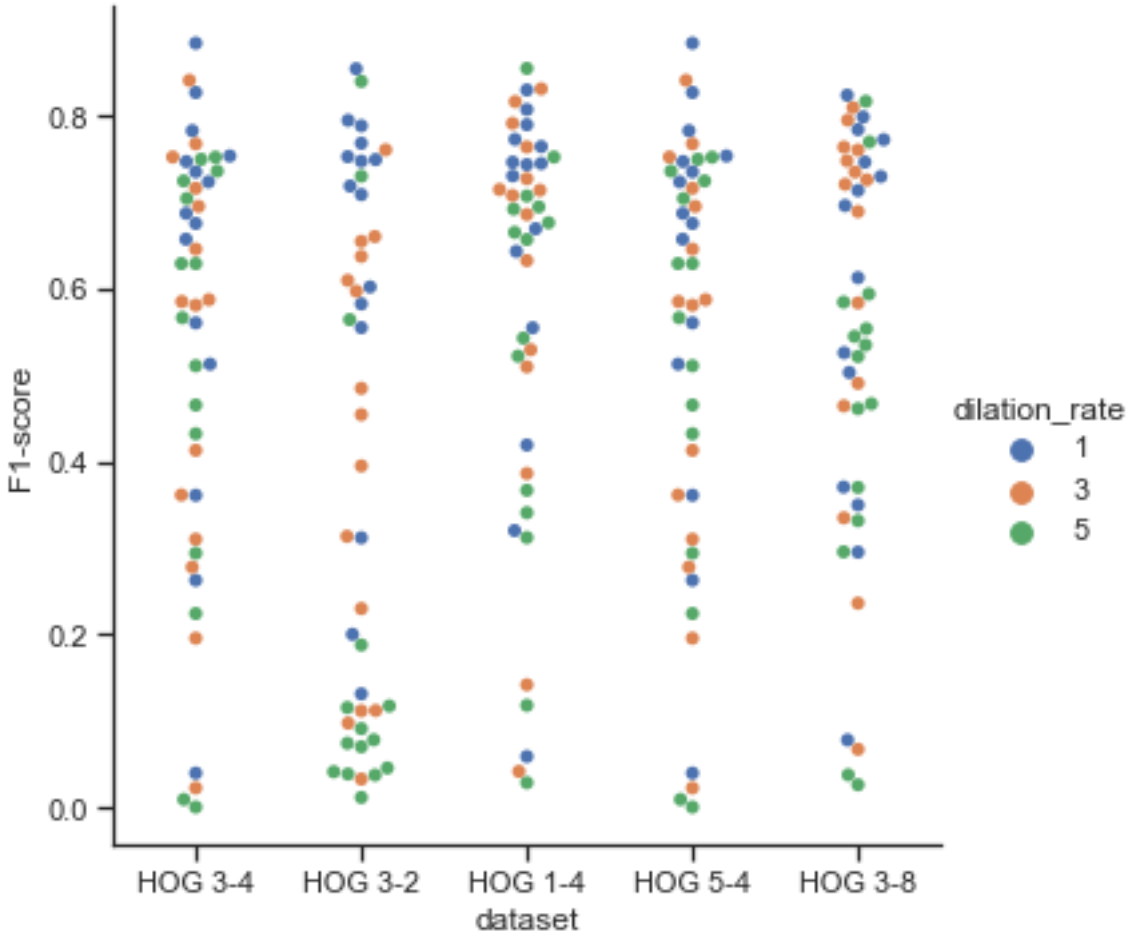


Figure 4.4: A cat-plot to visulize the different dilation rates on the HOG datasets with the highest predictions from table 4.5.

4.5.2 Dilation rates with medical images

Table 4.9: An overview of how dilation with both MED and HOG works.

MED HOG 3-2	**Augmentation**		
Dilation rate	NO AUG	AUG	A2 AUG
2	0.58636	0.64453	0.64965
3	0.56782	0.63033	0.63713
5	0.52029	0.61531	0.61638

4.5.3 Augmentation

One of the most critical parameters for the model is augmentation. In this thesis, there has mainly been tested out two types: the default augmentation and A2 Augmentation. These are explained in the method chapter. An overview of the parameter setup can be found in Table 3.7 and 3.8

Dataset	No - AUG	Def - AUG	A2 - AUG
MED	0.60516	**0.66097**	**0.67056**
HOG2	0.60560	0.62848	0.62876
LBP	0.55944	0.58406	0.57225
MED HOG2	0.60317	**0.67317**	**0.65341**
MED LBP	0.58183	0.62972	0.64302
HOG LBP	0.55988	0.57582	0.56861
MED HOG LBP	0.56390	0.62355	0.62672
mean	0.58271	0.62511	0.62333

Table 4.10: The dice score of the different augmentation used on the different datasets.

From Table 4.10, it is seen that augmented MED and MED HOG images give the best predictions. The thesis will further take the highlighted models and look deeper into each of the different segmentation to understand how, where, and if HOG contributes to the segmentation.

4.6 Comparison of the best models

This section will look into the four best models given from the augmentations in Table 4.10. In this section "HOG" refers to HOG(2-3) / HOG2, which was shown to be the best performing of the HOG datasets earlier in the chapter. An overview of the different datasets, augmentations, and names can be seen in the Table 4.11

Name of the datasets	Augmentation	
Datasets	Default (A)	A2
MED	MED A	MED A2
MED and HOG(2-3)	MED HOG A	MED HOG A2

Table 4.11: An overview of the names and what is included in the best models.

The best performing models were rerun. This time there was a slight change in performance, and the following table shows how they did this time. It is the rerun that is visualized in this chapter. The second run are showed in the Table **??**, followed by the difference between the runs in the Table 4.13.

Table 4.12: Results from the rerun of the four best models.

Dataset	Dice score	Std
MED A	0.65880	0.262005
MED A2	0.66589	0.257878
MED HOG A	0.66052	0.259745
MED HOG A2	0.67544	0.259273

Table 4.13: The table shows the change in performance between the two runs of the models.

-	MED A	MED A2	MED HOG A	MED HOG A2
first run	0.66097	0.67056	0.67317	0.65341
second run	0.65880	0.66589	0.66052	0.67544
difference	-0.00217	-0.00467	-0.1265	0.01803

4.6.1 Statistical Methods

The four statistical methods were tested for all the possible combinations on both the performance per slice and performance per patient on the different datasets, and none of them proved any significant change.

4.6.2 Difference per patient

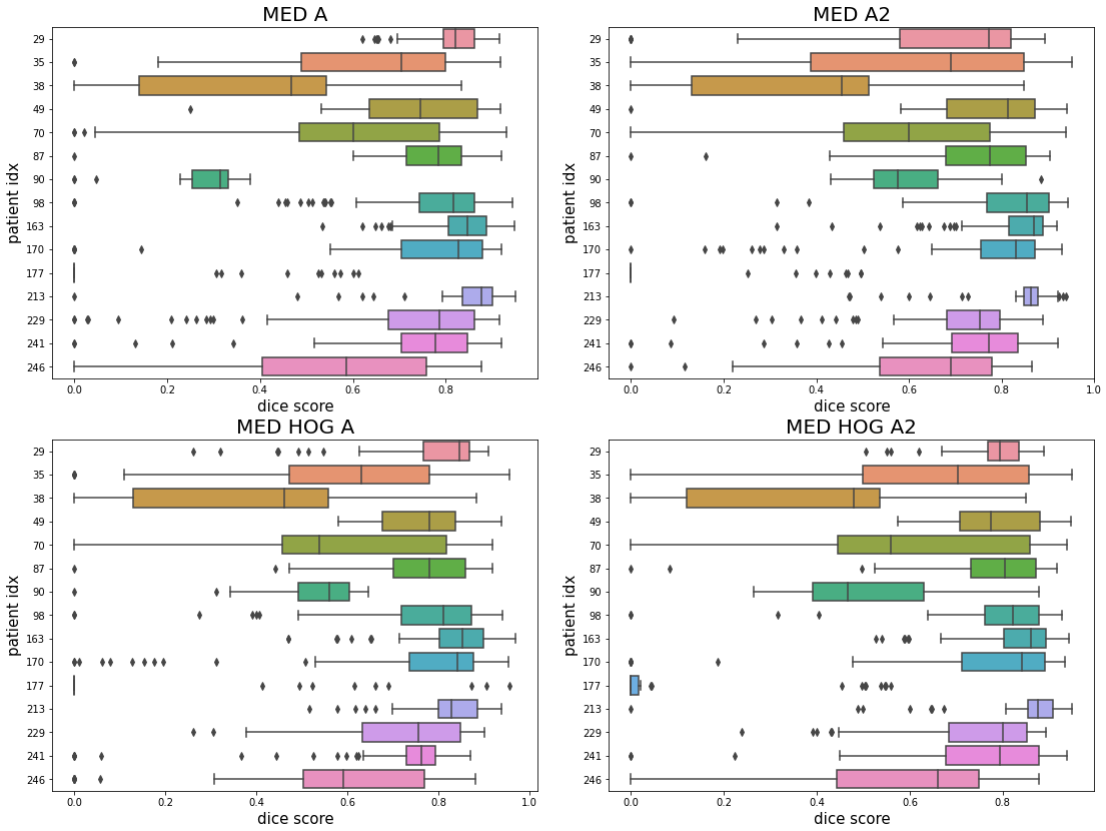


Figure 4.5: BOX-Plot to visualise the difference between the patients in the validation set for the different models. The center of the box is the median of the dice score for the patient, and the box represents the distribution from the 25% quantile to the 75% quantile. The end of the whiskers (lines) are representing the min and max of the distribution, while the points are the outliers.

Figure 4.5 visualizes the dice score for each patient using a box plot. From the figure it is seen that patient 177 is the worst performing patient and patient. No. 90, 38, and 246 predictions vary a lot between the models.

4.6.3 Comparing the dice score per slice for the different models

From the 15 patients in the validation set, there is a total of 1033 slices /images containing cancer. The following figure shows the difference between implementing HOG images to the medical model using different augmentations.

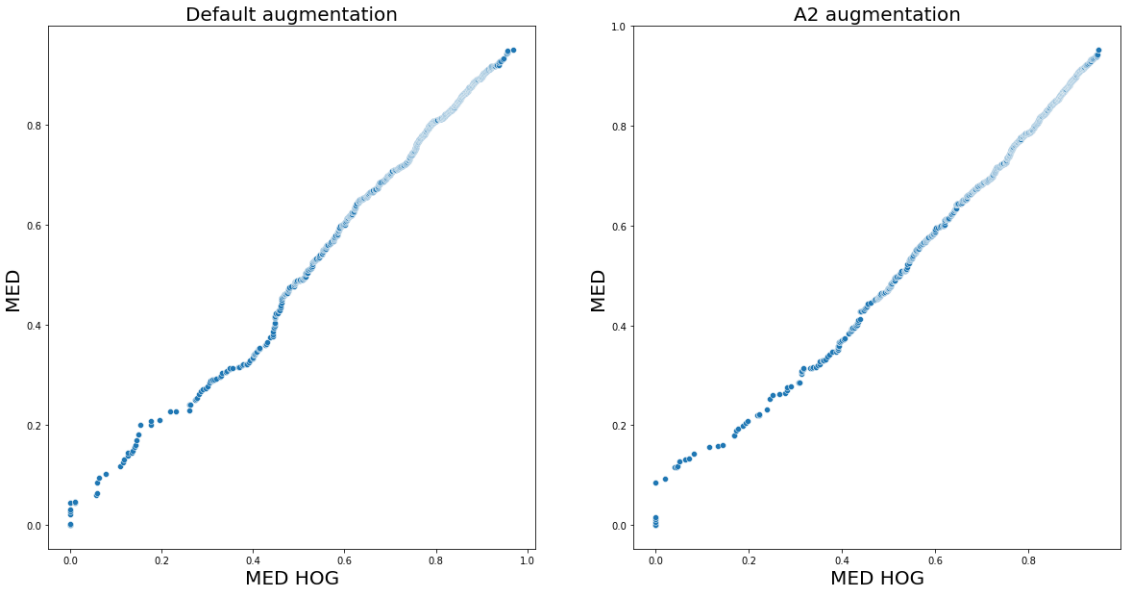


Figure 4.6: Plotting the dice score for the validation patients sorted by size within each model, where the left side shows the difference between MED and MED HOG where both using default augmentation. The right side shows the same scenario with A2 augmentation.

Figure 4.6 shows that both the Default (A) and A2 Augmentation follow a similar distribution. If the plot had the same distribution of dice scores the line would be straight. There is a tiny change in the left side of each plot, showing there are fewer lower dice scores for the MED models. That means it is slightly below the rest of the plot since the mean is the same. Both of the figure are almost in a straight line, meaning they have nearly the same distribution of dice score. For both scenarios, the data follows the same trend making it challenging to find a pattern in the data and to conclude with anything.

4.6.4 The tumor size effect on performance

The results go deeper into the data. After plotting the slices per patient, seen in Appendix C, it was clear that the performance between the models differs the most in the beginning and ends of the tumors. There was an assumption that this was because of the size of the tumor. Figure 4.7 shows all the slices, but the slice index for the patient is scaled down to be between zero and one. Both of the figures show a high difference in performance at the end of the tumors.

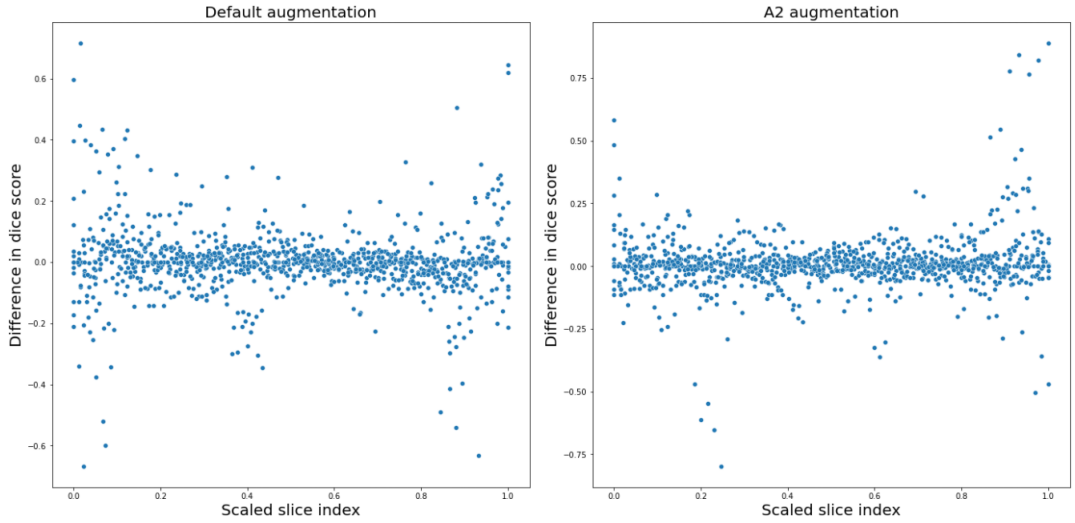


Figure 4.7: Plot showing the difference in data against each other, where the x - axis is the new scaled down slice axis.

Since the tumors are not necessarily small at the end of the slice, a figure plotting the difference in the model against the tumor size was created. Figure 4.8 shows the difference in models plotted against the size of the tumor, summing all the pixels containing tumors. The smallest tumor performance is drawn to the right in the figures, and the most extensive tumors slice performance to the right. It shows that the most significant changes are in the regions where the tumor is smaller.

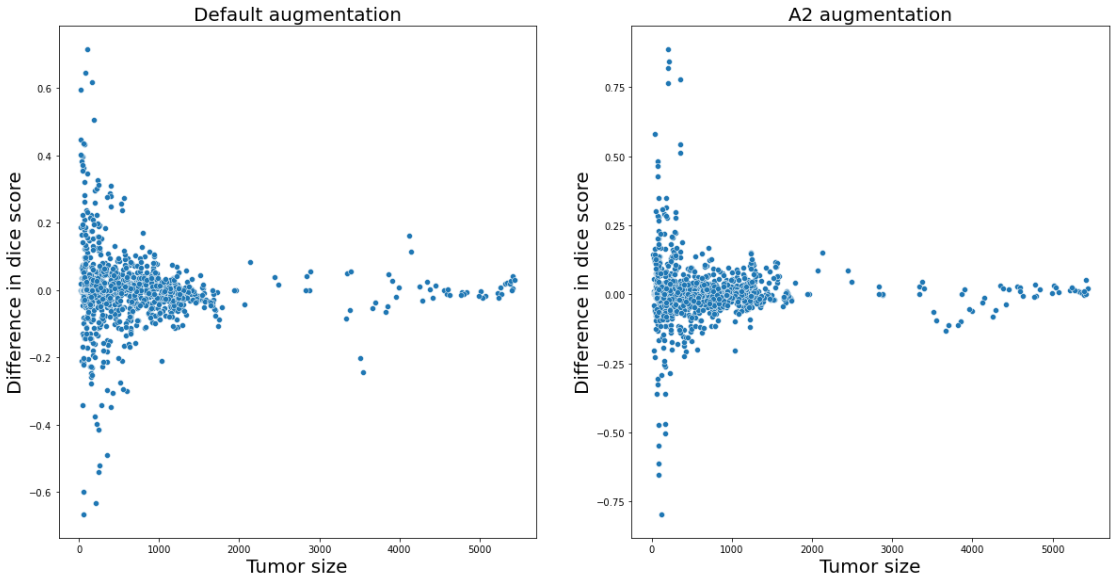


Figure 4.8: Plotting the difference in performance, between the datasets, against the size of the tumor. The difference is computed taking the performance of MED HOG - MED.

Chapter 5

Discussion

The thesis' goal is to determine if it is possible to improve the existing cancer segmentation using new image representations. In this thesis HOG and LBP were tested. The results indicate that HOG is the highest performing image representation out of the two and that LBP did not add as much improvement. The HOG alone does not outperform the medical images. However, when combining the medical and HOG datasets, the same score as the top-performing model was reached. The thesis then looked deeper into each segmentation to find out how these differ and to find structural changes within the data.

This section will go through the process leading to the results, the different parameters and models, how the parameters were chosen, and the results. Furthermore, it will be discussed which changes that could have been implemented, also addressing how this research can help to solve other similar problems and finally consider possible future research.

5.1 Dataset

5.1.1 Creation of medical images

The data leave some uncertainty. The PET and CT images were taken around a decade ago. Since then, the CT and PET imaging machines have become better and faster [74] [75], due to improved technology. For instance, now a CT exam can collect the images in less than two seconds, while it 20 years ago used 30 minutes.

There is a saying, "One year in real life, is like ten years in CT time [74]." The images used are still relatively new and up to date. However, also state-of-the-art machines are improving, and these images can also lead to better performance.

5.1.2 Contouring of tumor

Another factor causing uncertainty is the huge variety of radiologists, oncologists, and doctors participating. Using different personnel to perform contouring can lead to varying mask-drawings of the same tumor. Weiss and Hess showed that the variation of mask-drawings provided by different oncologists was one the most significant factors for uncertainty regarding the contouring of the tumors. For some tumors, the location was the most crucial factor leading to geometric inaccuracy [76].

5.1.3 Splitting of dataset

The data is divided into three groups; test, train, validation, as mentioned in Section 3.1. In this thesis mainly the validation sets of the different models have been compared. The validation data only represent 15 patients, where two of the patients have consistently been hard to segment correctly. These are patient No. 90 and No. 177. While the average across the slices for these two patients can vary a lot and reach scores down to 0,05, the remaining 13 patients generally lie on the same level, as observed in the figures in Section 4.4. The overall performance is firmly dependent on how well the prediction is of these patients. The per-patient score may be observed from the slice plots in Appendix A and the box plots in Section 4.4, which among others shows that patient No. 177 is the worst performing patient in the four best models. MED HOG A2 is one of the few models that manage to achieve an overall dice score that is slightly larger than the rest. This is also one of the reasons why the performance for MED HOG A2 is slightly higher than the three others.

5.2 Dataset Parameters

Firstly, the differences between the various datasets will be discussed. It is clear from the tables in Section 4.0 that there are differences in the performance of the

various datasets. None of the image representations manage to reach the same level of performance as the medical dataset. However, both of the image representations manage to capture a significant amount of the information alone. Nevertheless, when trying to combine the datasets, the models performance was not as high. Implementing HOG image representation was the one that decreased the score the least. Including LBP in addition to MED reduced the performance by almost five percent when alone and two percent if implemented with the medical images. The HOG decreases the score by around two percent when alone and when implemented with the medical images. After trying out the different datasets, the parameters that went into the creation of the image representation were optimized.

5.2.1 HOG-Parameters

The tables in Section 4.1 show how the different HOG parameters affect the models performance. Several parameters have been tested when creating the HOG image representation. The Cells Per Block (CPB) and Pixels Per Cell (PPC) size are the parameters that have been experimented with the most, creating the hog images. These parameters were chosen because the effect they have on the images change can easily be seen. Since the image representation is used, it is assumed that these effects have the biggest impact. The result of changing the descriptor block can be seen in figure 4.3.

Descriptor Block (PPC)

From Table 4.5 it may be observed that decreasing the size of the descriptor block generally increases the performance of the prediction. However, in most previous studies regarding HOG, such a small descriptor block has been used. The main reason for this is that in their thesis they have used the output vector and not the image representation. A smaller descriptor block size will lead to additional blocks/histograms to cover the image. This will further lead to more computations and an increased length of vectors. This affects the creation of the HOG data, but mainly when it is being fed into the classifier [7] [13]. However, in this thesis, it only affects the creation of the images. Since here, the image representations are used and not the vector. The size of the image size is set to the same resolution as the medical image, making the data size constant. This also makes it more tolerable to use a smaller descriptor block.

However, just the computational cost of creating the HOG images is still substan-

tial. Therefore in this thesis, one-pixel descriptor blocks have not been tried out. However, this could have been interesting future work. The result here would look more like an edge detector since there will only be one magnitude to place in each histogram for all the pixels in the image without the normalization blocks. With only one magnitude, the maximum number of bins used for each histogram will be two.

Another important factor that has not been tested is the usage of different descriptor block shapes. In the theory chapter, both the C-HOG and R-HOG have been discussed. Both got their breakthrough from Dalal and Triggs paper in 2005 [7]. They achieved good results with both C-HOG and R-HOG but a slightly better score using the R-HOG. This has also been the case in numerous later studies. Therefore, the R-HOG has become the most used normalization block. It also benefits from easier implementation. A third option that has been discovered in the last decade is the Segmental-HOG. The Segmental-HOG, known as S-HOG, uses a flexible block division. The method was developed in 2015 by Tsuyoshi Kato and was used on microscopic images of kidneys to segment out "glomerium "[77]. This method achieved significantly better performance, halving the number of false negatives while keeping the same number of true positives as the R-HOG. This paper is also using a sliding window with an SVM for classification.

Normalization Blocks (BPC)

The different normalization block results are shown in Table 4.5, with the usage of dilation in the tables in Section 4.3. From the tables, it is not possible to see a direct pattern change over the four different descriptor blocks. From earlier studies, the overlapping local contrast normalization has improved the segmentation [7]. As described in the last section, these results are done using the vector and not feed into a CNN. The reason the normalization filter does not perform so well might be because of the U-Nets structure. The convolutional layers may manage to capture the relationships between each pixel so well that using overlapping blocks is unnecessary.

Another reason why the performance does not increase might be because the image does not manage to show such a detailed star histogram. With a small descriptor block, the star histogram becomes so small that the change in the image is hard to visualize because the normalization increases the vector by taking the surrounding blocks vectors into account. Each of these vectors is to be shown by the 9 bin histogram in the plot. It is possible that the image representation does not capture the added information that well. A vector can capture the information

of multiple bins oriented in the same direction by extending the vector. Still, an image representation might have trouble with handling the same scenario because it has to be presented as an image.

5.2.2 LBP-Parameter

The LBP image representation has not been tested as much as HOG. In this thesis, there have mainly been used two different parameter setups with LBP, which are shown in Table 3.6. One setup used a one-pixel radius with eighth number of points while the other with a two-pixel radius and 16 number of points. The tables in Section 4.2 showed that the smaller radius performed better than the larger radius. This corresponds well with other research undertaken on similar data segmentation [78]. There could also have been tested parameter setups with a higher number of points on a one-pixel radius. Proven to give a high accuracy in rectal cancer earlier [78].

5.2.3 Model hyper-parameters

The two main different parameters tested in the model are dilation and augmentation. Explained in the theory and method.

Dilation

The dilation used is explained in the theory chapter (section 2.6). It involves increasing the convolutional kernel without increasing the number of weights, which is achieved by adding a spatial distance between them, called dilation rate. Different spacing in the dilated kernels was tested. However, the dilation rate within the model was always the same. The dilation was tried out on the HOG data alone and on the HOG data combined with the medical data. The tables in section 4.3 show that the model does not improve when implementing dilated kernels. In fact, the performance decreased as the dilation rate went up. This might be because the close spatial relationship in the medical images is crucial for a good delineation. Other research has tried replacing the pooling layer with a pyramid dilated kernel with success on medical image segmentation [79]. However, they still used the normal convolutional network.

Augmentation

The augmentation data used in this thesis is gathered from another master thesis from this year and earlier year master students work on the same dataset. The augmentation showed to be one of the most important factors for improving the delineation. The different parameters in the two setups used can be seen in Table 3.7 and 3.8. There is a large number of options to how to augment the data, so finding the perfect setup would take too long to implement in this master thesis. As a result, only these two augmentation setups were used. From Table 4.10, it is obvious that both the "def"/"A" augmentation and the "A2" augmentation increase the performance of all the datasets models. This correlates well with both earlier work done with the same dataset [16] [17] and work done on other medical data [80].

Since the augmentations are found testing multiple parameter setups on only the medical images, it means that these parameters are not necessarily the best for the HOG and LBP image representations. Future work could be to find an augmentation to improve the HOG prediction.

Windowing

One of the other parameters that could be classified under preprocessing is windowing. The windowing chooses which specter of the tissues we want to enhance and visualize. Choosing the right specter is important for a good delineation.

,

5.3 Analysis of the models

This section will review and compare the performance of the different models.

5.3.1 PET and CT

The PET- and CT-channels are the base channels for all image representations. Table 4.3 shows that both channels capture a lot of the information and that the

PET channel alone performs slightly better than the CT channel alone. From Table 4.10, it may be observed that the combination of them ("MED"), improves the prediction even further. The cat-plot in section 4.0 showcases the difference between implementing augmentation and adding the corresponding HOG images. It is clear that combining the CT and PET images gave the best results, and implementing the HOG image representations of them decreased the performance. However, they also show that implementing preprocessing methods such as augmentation and windowing increases the performance significantly.

5.3.2 The best models

Section 4.5 shows the best models performance. The four best models are medical images with "def" augmentation, "A2" augmentation, with and without the HOG data. These four tests were rerun, and the performance of the new run varied. The variation between the two runs can be seen in Table 4.13. The result from the table indicates that the model's performances are quite close to one another, and it seems that only the initialization of weights may explain any differences in performance. The best HOG model performs slightly better than the best medical dataset. However, there is a higher computational cost for creating the image representations.

From Appendix C, the dice score per slice is plotted for each of the patients in the validation set. Both of the run performance plots can be found there. These plots show that the prediction varies and that maybe all models should have been run multiple times to get a better estimate for each of the models. However, there has not been time to rerun the tests, except for the four best models.

The models' performance also varies because of the sample size. The validation data consists of only 15 patients, and the models have been run on both the test data and a second dataset on rectal cancer. These performances are shown in Section 4.7. The performance on the test data is generally increased. This might be due to the high percentage of patients in the validation set where predictions are poor.

However, it is hard to tell if there is any significant change to the data from the dice scores alone, something that was also concluded based on the statistical analysis. In Section 4.5 the performance was further analyzed to see if there were any structural changes between the models.

Performance per patient

Figure 4.5 shows the performance of all 15 patients in the validation set for all four models. From figure it can be observed that all patients follow the same pattern, where each patient's performance stays almost the same in each of the models.

5.3.3 Dice score across slices

The per slice section mainly looked into two structural differences: the distribution of the dice scores and the performance compared against tumor size.

Dice distribution

From Figure 4.6 the sorted dice distribution is plotted against each other. The plots are similar to a Q-Q plot, and they show a slight difference in the number of poor predictions favoring the model based on medical images only. However, the plots are making up a straight line and hardly any differences are observed between the models.

Tumors size per slice

The dice distribution per patient was also analyzed. The plots are seen in Appendix C. It is clear from the plot that the models vary mostly at the beginning and end of the tumors when the tumors are smaller in size. This was expected since it is harder to detect smaller objects. Figure 4.7 shows the beginning and end of the tumors. However, there is not always the first slice of the tumor that is the smallest, and Figure 4.8, shows how the different models' dice scores differ from each other depending on the size of the tumor. Moreover, this could also be caused by the performance measurement, which will be discussed in Section 5.4.

5.4 Analysis of the static methods used

The original plan for this thesis was to treat the data as if it was normally distributed, and use an ANOVA test. However, the results were shown to not follow

a normal distribution, using a Q-Q plot. There were then used multiple non-parametric methods to see if there was any significant change between the dataset. Where none of them prove a significant change between the models predictions.

Statistical methods

One of the main difficulties in proving any significant differences in the models, is the large number of slices that achieve a zero performance and that no pixels predicted correctly.

5.5 Analysis of the performance metric

Dice Score

The dice score is calculated using the formula shown in definition (3.9.1). The definition takes the total number of values that is predicted right (TP) times two divided by the sum of the predicted cancer cells (TP + FN) and actual cancel cells (FN + TN). The performance does not value the size of the tumor. Each pixel for a smaller cancer prediction is valued higher. This might be one of the reasons why the smaller cancer segmentation is varying more when it comes to the segmentation, seen in the figures in Section 4.4.

5.6 Use of research

The research in this thesis focus on the segmentation of cancer. Be that as it may, the study could be generalized for multiple segmentation tasks. Using the image representation for segmentation and not their descriptor vector for classification is proved to give positive feedback. This research concludes that it could be possible to include the HOG images as external channels for other types of segmentation tasks.

Chapter 6

Conclusion

Conclusion

In this thesis, it has been tested whether new image representations can boost the all-ready existing cancer segmentation using a U-Net structure. The image representations used are histograms of oriented gradients (HOG) and local binary patterns (LBP). Numerous different parameters and parameter combinations within these image representations, in particular HOG, have been tested. The size and the normalization of the histogram have been the most tested parameters. The image representations are created in Python using modules from sklearn.

The thesis found that the image representation alone decreased the score slightly. However, the image representations managed to capture a lot of the information. The performance of all the image representations was higher on the PET images compared to the CT-scans images. Further, implementing HOG with a smaller descriptor block gave a dice score of 0.675 on the validation set (second run, MED HOG A2). This result was higher than the best performance using only the medical data, which reached a dice score of 0.67056 (first run, MED A2).

Multiple different setups have been used when creating the image representations. In order to compare the models, the dice score on the validation set was used. The best model was further combined using the other images and image representation as new channels in the U-Net to see if this could improve the performance. The model was tested with multiple augmentations and the use of dilational convolutional layers in the U-Net. The augmentation proved to be an essential parameter

for the model's performance, but the dilation did not add any improvement. The augmentation was created by the former master and doctoral students working on the medical data. It is possible that optimizing the augmentation setup to fit the image representations could increase the performance even further.

Then, the best performing models were rerun and the prediction for each of the slices and patients were compared to see whether there were any structural changes in the data. From the patient information, it was found that performance is following almost the same pattern. Be that as it may, it was seen that the tumor size was an essential factor for the performance. The differences in the models dice score were more remarkable when the tumor was smaller. The dice scores calculation could be one of the reasons for this.

Overall, the LBP is the worst performing of the two image representations. There was also found a slight improvement combining the medical data with the HOG data. The slight improvement might be worth the extra computational cost since it can be a matter of life and death. However, HOG has still not reached its full potential. The HOGs image representation proved to provide a lot of information, making it attractive for implementation in further works related to image segmentation.

Chapter 7

Future work

This thesis raises several issues to be addressed in future projects. Among others, this may be to consider the parameter in the creation of HOG. In this thesis, the focus has been on PPC and CPB. But the other parameters, the number of orientation, the normalization, and the shape of the descriptor block should also be tested in more detail. It is possible to change the shape of the descriptor block. Test for instance C-HOG or S-HOG described earlier in this thesis, or try to create a new one. There is also potential to try to make an optimized augmentation for HOG and LBP.

For LBP, the Local Structure Local Binary Pattern (LS LBP) could be tested to see if it is possible to capture more of the data.

It could also be adding a new image representation. The most used comparison to the HOG and LBP is the Haar cascade classifier, also known as the Haar classifier. Haar cascade uses the Haar features to detect patterns in the data. These are achieved by taking the sum of regions in an image and comparing them to the adjacent areas. The regions are linearly separable and known as a "weak classifier." By using multiple classifications and boosting, hopefully, an even better segmentation could be reached.

Further work regarding machine learning is implementing boosting to the current models. Boosting is an ensemble method, with the purpose of training the computer to use the best model for each slice in the segmentation, combining the different models.

The use of the HOG vector and a classifier such as SVM for segmentation could

also be a subject for further testing. This is the most common way to use the HOG, LBP, and Haar descriptors. However, for Haar, LBP, and the HOG-vector, the most common approach is to use a sliding window. The sliding window could make it challenging to get an accurate segmentation, compared to CNN networks.

There is a lot of possible future work. Both in terms of the preprocessing part; creation of new image representation, optimizing parameters/model and augmentation, or in the direction of machine learning; boosting the models, implementing Haar classifier, and HOG vector representation.

Bibliography

[1] WHO. (2021). Who - cancer key facts, [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/cancer.

[2] T. N. C. association. (2021). What is cancer, [Online]. Available: https://kreftforeningen.no/om-kreft/hva-er-kreft/ (visited on 25/01/2021).

[3] N.-N. C. institute. (2021). Understanding cancer, [Online]. Available: https://www.cancer.gov/about-cancer/understanding/what-is-cancer (visited on 25/01/2021).

[4] D. M. Brizel and D. J. Adelstein, 'Locally advanced squamous carcinoma of the head and neck', *Perez and Brady's Principles and Practice of Radiation Oncology*, vol. 6, pp. 718–29, 2013.

[5] Cancer.net. (2021). Head and neck cancer: Statistics, [Online]. Available: https://www.cancer.net/cancer-types/head-and-neck-cancer/statistics (visited on 25/01/2021).

[6] ——, (2019). Head and neck cancer: Risk factors and prevention, [Online]. Available: https://www.cancer.net/cancer-types/head-and-neck-cancer/risk-factors-and-prevention (visited on 25/01/2021).

[7] D. Triggs, 'Histograms of oriented gradients for human detection', 2005.

[8] D. G. Lowe, 'Distinctive image features from scale-invariant keypoints', 2004.

[9] Y. Socarrás Salas, D. Vázquez Bermudez, A. M. López Peña, D. Gerónimo Gomez and T. Gevers, 'Improving hog with image segmentation: Application to human detection', in *Advanced Concepts for Intelligent Vision Systems*, J. Blanc-Talon, W. Philips, D. Popescu, P. Scheunders and P. Zemčík, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 178–189, ISBN: 978-3-642-33140-4.

[10] X. Wang, T. Han and S. Yan, 'An hog-lbp human detector with partial occlusion handling', Nov. 2009, pp. 32 –39. DOI: 10.1109/ICCV.2009.5459207.

[11] J. Zhang, K. Huang, Y. Yu and T. Tan, 'Boosted local structured hog-lbp for object localization', in *CVPR 2011*, IEEE, 2011, pp. 1393–1400.

[12] K. Reddy, B. Solmaz, P. Yan, N. Avgeropoulos, D. Rippe and M. Shah, 'Confidence guided enhancing brain tumor segmentation in multi-parametric mri', *Proceedings - International Symposium on Biomedical Imaging*, pp. 366–369, May 2012. DOI: 10.1109/ISBI.2012.6235560.

[13] T. Kato, R. Relator, H. Ngouv, Y. Hirohashi, O. Takaki, T. Kakimoto and K. Okada, 'Segmental hog: New descriptor for glomerulus detection in kidney microscopy image', *Bmc Bioinformatics*, vol. 16, no. 1, pp. 1–16, 2015.

[14] A. D. Midtfjord, 'Prediksjon av behandlingsutfall for hode- og halskreft ved bruk av radiomics av pet/ct-bilder', 2018.

[15] G. S. R. E. Langberg, 'Searching for biomarkers of disease-free survival in head and neck cancers using pet/ct radiomics', 2019.

[16] Y. Moe, 'Deep learning for automatic delineation of tumours from pet/ct-images', 2019.

[17] B. N. Huynh, 'Visualization of deep learning in auto-delineation of cancer tumors', 2020.

[18] A. cancer society. (2015). What is cancer, [Online]. Available: https://www.cancer.org/cancer/cancer-basics/what-is-cancer.html.

[19] M. clinic. (2015). Cancer, [Online]. Available: https://www.mayoclinic.org/diseases-conditions/cancer/symptoms-causes/syc-20370588 (visited on 01/02/2021).

[20] O. Klepp. (2015). Metastase, [Online]. Available: https://sml.snl.no/metastase (visited on 02/02/2021).

[21] Cancer.net. (2019). Head and neck cancer: Introduction, [Online]. Available: https://www.cancer.net/cancer-types/head-and-neck-cancer/introduction (visited on 16/02/2021).

[22] mayoclinic. (2019). Squamous cell carcinoma of the skin, [Online]. Available: https://www.mayoclinic.org/diseases-conditions/squamous-cell-carcinoma/symptoms-causes/syc-20352480 (visited on 16/02/2021).

[23] Medanta. (2017). What is carcinoma buccal mucosa, [Online]. Available: https://www.medanta.org/cancer-hospital/head-and-neck/disease/carcinoma-buccal-mucosa/ (visited on 05/05/2021).

[24] J. T. Mullen, L. Feng, Y. Xing, P. F. Mansfield, J. E. Gershenwald, J. E. Lee, M. I. Ross and J. N. Cormier, 'Invasive squamous cell carcinoma of the skin: Defining a high-risk group', *Annals of surgical oncology*, vol. 13, no. 7, pp. 902–909, 2006.

[25] mayfair diagnostics. (2018). Cancer: Why do men have higher cancer risk, [Online]. Available: https://www.radiology.ca/article/cancer-why-do-men-have-higher-risk.

[26] C. R. UK. (2018). Age the biggest cancer risk factor, [Online]. Available: https://scienceblog.cancerresearchuk.org/2018/06/20/age-the-biggest-cancer-risk-factor/.

[27] ——, 'Cancer awareness measure (cam), key findings report; 2014 trends analysis (2008-2014)', Nov.2006. DOI: https://www.cancerresearchuk.org/sites/default/files/cam_key_findings_report_-_2014_trends_analysis_v5.pdf?_gl=1*1s7d9wb*_ga*NjQwOTA5MTQ3LjE2MjAyMTYwMjU.*_ga_58736Z2GNN*MTYyMTMzNDIyMy4zLjAuMTYyMTMzNDIyMy42MA..&_ga=2.118556495.714016522.1621334223-640909147.1620216025.

[28] M. C. White, D. M. Holman, J. E. Boehm, L. A. Peipins, M. Grossman and S. J. Henley, 'Age and cancer risk: A potentially modifiable relationship', *American journal of preventive medicine*, vol. 46, no. 3, S7–S15, 2014.

[29] N. C. institute. (march 11, 2020). Target therapy to treat cancer, [Online]. Available: https://www.cancer.gov/about-cancer/treatment/types/targeted-therapies (visited on 17/02/2021).

[30] nhs national health service. (29.jan 2020). Chemotherapy, [Online]. Available: https://www.nhs.uk/conditions/chemotherapy/.

[31] N. cancer institute. (2020). Side effects of cancer treatment, [Online]. Available: https://www.cancer.gov/about-cancer/treatment/side-effects (visited on 05/05/2021).

[32] P. Medical. (2019). What is pet-ct?, [Online]. Available: https://www.plusmedical.ro/en/articole/ce-este-un-pet-ct/?gclid=EAIaIQobChMIhrX7i5nw7AIVl6myCh1BwE. (accessed: 07.11.2020).

[33] D. W. Townsend, 'Combined positron emission tomography–computed tomography: The historical perspective', in *Seminars in Ultrasound, CT and MRI*, Elsevier, vol. 29, 2008, pp. 232–235.

[34] I. college London. (2021). How does pet scan work, [Online]. Available: https://www.youtube.com/watch?v=yrTy03O0gWw&ab_channel=ImperialCollegeLondon. (accessed: 25.05.2021).

[35] P. Brian Nett. (2020). Illustrated comparison of ct scan modes [axial, helical, wide-cone axial, low pitch, high pitch] for radiologic technologists, [Online]. Available: https://howradiologyworks.com/14040-2/. (accessed: 25.05.2021).

[36] Z. Xue, S. Antani, L. R. Long, D. Demner-Fushman and G. R. Thoma, 'Window classification of brain ct images in biomedical articles', in *AMIA Annual Symposium Proceedings*, American Medical Informatics Association, vol. 2012, 2012, p. 1023.

[37] SciKit-Image. (2020). Scikit-image feature module: Hog, [Online]. Available: https://scikit-image.org/docs/dev/api/skimage.feature.html?highlight=hog#skimage.feature.hog. (accessed: 04.11.2020).

[38] M. Harmouch. (2020). Local bianry pattern algorithm: The math behind it, [Online]. Available: https://medium.com/swlh/local-binary-pattern-algorithm-the-math-behind-it-%EF%B8%8F-edf7b0e1c8b3.

[39] J. J. McCarthy, 'Artificial intelligence', 2010. [Online]. Available: https://www.researchgate.net/publication/41667499_Artificial_Intelligence (visited on 16/02/2021).

[40] Pathmind. (2020). Artifical intelligence vs machine learning vs deep learning, [Online]. Available: https://wiki.pathmind.com/ai-vs-machine-learning-vs-deep-learning (visited on 16/02/2021).

[41] N. Samani, M. Gohari-Moghadam and A. Safavi, 'A simple neural network model for the determination of aquifer parameters', *Journal of Hydrology*, vol. 340, no. 1, pp. 1–11, 2007, ISSN: 0022-1694. DOI: https://doi.org/10.1016/j.jhydrol.2007.03.017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022169407001862.

[42] V. M. Sebastian Raschka, *Python machine learning*. Packt publishing limited, 2017-09-01, ISBN: 9781787125933.

[43] J. Han and C. Moraga, 'The influence of the sigmoid function parameters on the speed of backpropagation learning', in *From Natural to Artificial Neural Computation*, J. Mira and F. Sandoval, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 195–201, ISBN: 978-3-540-49288-7.

[44] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, J. Kolen and S. Kremer, 'A field guide to dynamical recurrent neural networks', *chapter Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies*, pp. 237–243, 2001.

[45] P. Ramachandran, B. Zoph and Q. V. Le, *Searching for activation functions*, 2017. arXiv: 1710.05941 [cs.NE].

[46] X. Glorot, A. Bordes and Y. Bengio, 'Deep sparse rectifier neural networks', in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2011, pp. 315–323.

[47] V. Kakaraparthi. (2019). Activation functions ini neural netwroks- what are they?, how they work? and where to use them?, [Online]. Available: https://prateekvishnu.medium.com/activation-functions-in-neural-networks-bf5c542d5fec (visited on 02/03/2021).

[48] W. Wang and Y. Lu, 'Analysis of the mean absolute error (mae) and the root mean square error (rmse) in assessing rounding model', in *IOP conference series: materials science and engineering*, IOP Publishing, vol. 324, 2018, p. 012 049.

[49] F. Chollet *et al.*, 'Deep learning with python', vol. 361, 2018.

[50] S. Albawi, T. A. Mohammed and S. Al-Zawi, 'Understanding of a convolutional neural network', in *2017 International Conference on Engineering and Technology (ICET)*, Ieee, 2017, pp. 1–6.

[51] S. Saha. (2018). A comprehensive guide to convolutional neural networks — the eli5 way, [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. (accessed: 27.04.2021).

[52] A. LeNail, 'Nn-svg: Publication-ready neural network architecture schematics', *The Journal of Open Source softwear*, p. 1, 2019. DOI: https://joss.theoj.org/papers/10.21105/joss.00747.pdf.

[53] ——, (2019). Publication-ready nn-architecture schematics. this web page visualization tool was part of his thesis, [Online]. Available: http://alexlenail.me/NN-SVG/AlexNet.html. (accessed: 27.04.2021).

[54] W. Burger and M. Burge, *Digital Image Processing*. Springer London, 2016, ISBN: 978-1-4471-6683-2.

[55] M. Mishra. (2020). Convolutional neural networks, explained, [Online]. Available: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939. (accessed: 11.03.2021).

[56] g. f. g. Savak Hosla. (2019). Cnn introduction to pooling layer, [Online]. Available: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/. (accessed: 03.05.2021).

[57] O. Ronneberger, P.Fischer and T. Brox, 'U-net: Convolutional networks for biomedical image segmentation', in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, ser. LNCS, (available on arXiv:1505.04597 [cs.CV]), vol. 9351, Springer, 2015, pp. 234–241. [Online]. Available: http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a.

[58] S. Wang, S.-Y. Hu, E. Cheah, X. Wang, J. Wang, L. Chen, M. Baikpour, A. Ozturk, Q. Li, S.-H. Chou, C. D. Lehman, V. Kumar and A. Samir, *U-net using stacked dilated convolutions for medical image segmentation*, 2020. arXiv: 2004.03466 [eess.IV].

[59] F. Yu and V. Koltun, *Multi-scale context aggregation by dilated convolutions*, 2016. arXiv: 1511.07122 [cs.CV].

[60] tensorflow. (2021). Tf.keras.layers.conv2dtranspose, [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2DTranspose. (accessed: 03.05.2021).

[61] O. Ronneberger, P. Fischer and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015. arXiv: 1505.04597 [cs.CV].

[62] python. (2021). Python - homepage, [Online]. Available: https://www.python.org/. (accessed: 04.05.2021).

[63] B. N. Hyunh, 'Development of a keras-based cnn framework for automatic delineation of cancer tumors', (accessed: 26.04.2021).

[64] ——, (2019). Deoxys documentation, [Online]. Available: https://deoxys.readthedocs.io/en/latest/index.html. (accessed: 26.04.2021).

[65] Cigene. (2016). Orion, [Online]. Available: https://cigene.no/lab-and-infrastructure/cigene-computational-unit/. (accessed: 26.04.2021).

[66] S.-I. imageprocessing in python. (2021). Scikit-image - imageprocessing in python, [Online]. Available: https://scikit-image.org/. (accessed: 04.05.2021).

[67] SciKit-Image. (2019). Histogram of oriented gradients, [Online]. Available: https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_hog.html#sphx-glr-auto-examples-features-detection-plot-hog-py. (accessed: 04.05.2021).

[68] ——, (2019). Local binary pattern for texture classification, [Online]. Available: https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_local_binary_pattern.html. (accessed: 04.05.2021).

[69] matplotlib. (2021). Matplotlib: Visualization with python, [Online]. Available: https://matplotlib.org/. (accessed: 04.05.2021).

[70] Pandas. (2021). Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the python programming language., [Online]. Available: https://pandas.pydata.org/. (accessed: 04.05.2021).

[71] NumPy. (2021). The fundamental package for scientific computing with python, [Online]. Available: https://numpy.org/. (accessed: 04.05.2021).

[72] Keras. (2021). Deep learning for humans, [Online]. Available: keras.io. (accessed: 04.05.2021).

[73] NMBU. (2020). Nmbu orion hpc cluster, [Online]. Available: https://support.nmbu.no/it-dokumentasjon/nmbu-orion-regneklynge/. (accessed: 26.04.2021).

[74] I. I. society for Computed Tomography. (2016). Half a century in ct: How computed tomography has evolved, [Online]. Available: https://www.isct.org/computed-tomography-blog/2017/2/10/half-a-century-in-ct-how-computed-tomography-has-evolved. (accessed: 20.05.2021).

[75] T. Jones and D. Townsend, 'History and future technical innovation in positron emission tomography.', *Journal of medical imaging (Bellingham, Wash.)*, vol. 4.1, 2017. DOI: 10.1117/1.JMI.4.1.011013.

[76] C. F. H. Elisabeth Weiss, 'The impact of gross tumor volume (gtv) and clinical target volume (ctv) definition on the total accuracy in radiotherapy theoretical aspects and practical experiences', *Strahlenther Onkol. 2003 Jan;179*, vol. 1, pp. 21–30, 2003. DOI: 10.1007/s00066-003-0976-5.

[77] H. N. Y. H. O. T. T. K. K. O. Tsuyoshi Kato Raissa Relator, 'Segmental hog: New descriptor for glomerulus detection in kidney microscopy image', *BMC Bioinformatics*, vol. 16, no. 316, 2015. DOI: https://doi.org/10.1186/s12859-015-0739-1.

[78] Y. Shi, W. Yang, Y. Gao and D. Shen, 'Does manual delineation only provide the side information in ct prostate segmentation?', in *Medical Image Computing and Computer Assisted Intervention MICCAI 2017*, M. Descoteaux, L. Maier-Hein, A. Franz, P. Jannin, D. L. Collins and S. Duchesne, Eds., Cham: Springer International Publishing, 2017, pp. 692–700, ISBN: 978-3-319-66179-7.

[79] Q. Zhang, Z. Cui, X. Niu, S. Geng and Y. Qiao, 'Image segmentation with pyramid dilated convolution based on resnet and u-net', in *Neural Information Processing*, D. Liu, S. Xie, Y. Li, D. Zhao and E.-S. M. El-Alfy, Eds., Cham: Springer International Publishing, 2017, pp. 364–372, ISBN: 978-3-319-70096-0.

[80] S. O. M. J. C. Zach Eaton-Rosen Felix Bragman, 'Improving data augmentation for medical image segmentation', p. 3, 2018. DOI: https://openreview.net/pdf?id=rkBBChjiG.

Chapter 8

Appendix

8.1 Appendix A: DAT390 Report

DAT390

Histogram of oriented gradients on medical images

Markus Ola Holte Granheim

DAT390-article 2020

Contents

1 Summary

This paper aims to show how Histograms of oriented Gradients can help improve the computational cost of cancer segmentation and visualize medical data. The aim is to understand how it can help reduce the load on radiologists and leave more of the segmentation to the computer. HOG uses the orientation and magnitude of small pixel-patches in an image to represent it as a histogram of oriented gradients and then contrast normalize each patch. This method is used on Positron-Emission-Tomography(PET) - and Computer-Tomography(CT) -scans gathered from Oslo University Hospital. The results show that the HOG images catch up a lot of the essence in PET and CT scans. When the model is finished, the prediction can tell us whether it provides a better result or not. The paper shows us what the different parameters can help us achieve and what methods can help receive as clear images as possible.

2 Introduction

The need for radiologists(interpreters) has become larger and larger over the last decades. More medical images are produced from a growing number of medical equipment. Analyzing all these images is both times consuming and hard. Implementing data science into this process can help shorten down this process a lot. This issue has been investigated for many years by computer scientists. The medical data may be huge, which makes it computationally expensive.

To reduce the computational cost, an approach known as HOG(Histogram of oriented gradients) is used in this paper. HOG is a feature descriptor that reduces the data size by only catching the important parts and removing the rest. It reduces a 2-dimensional image down to a single array. This array can, e.g., be trained by classifiers to recognize cancer. HOG can handle almost any type of image data, making it robust and easy to implement. That is also one reason it has been used for more than a decade after Dalal and Triggs got their breakthrough in 2005 to detect standing people.[1]

This paper will look into the HOG functions, find the important information in an image, discard the useless information, and reduce the data size. It will also discuss some pre-processing methods to boost the performance of the predictions. The paper will mainly focus on how the data can be optimized and not on the prediction results. The latter will be further discussed in my master thesis.

The data-set used here consists of Head and Neck scans gathered from Oslo university hospital. The data contains both Computer- Tomography(CT)-scans and Positron-Emission-Tomography(PET) scans for 198 patients. We will look into how to pre-process each value and what parameters can help optimize the HOG output.

The rest of the paper will explain the dataset in section 3, explain the theory and the method/implementation in section 4, the result from HOG transformation in section 5, discuss possible changes in section 6 and provide a conclusion in section 7 before further work and references are shown.

3 Dataset

The data used for this research is gathered from Oslo University Hospital. It contains both CT-scans and PET-scans. The data is gathered from 197 different cancer-patients. The scan is taken layer by layer from the patient's neck to the top of their head. To reduce the data, all scans that do not contain cancer have been removed. This makes the images per patient different since the tumor size can be different from patient to patient. In total, there are 13750 images. The scans are seen from above the head and down. Making the first slice for each patient the top of the scalp, and the last slice for each patient the bottom of the neck. Each image slice has a correlating cancer-mapping, an image where a radiologist predicts cancer. These slices are seen as the truth, and we wish the computer to estimate after the training process.

4 Theory and Method

4.1 Overview of Method

This section gives an overview of the method used to manufacture the HOG-images and each step's theory. It will go through each of the steps on how the HOG-features are created. The math behind the different operations is based on simple calculus and trigonometry.

The method uses a locally contrast normalized histogram of oriented gradients given from the edge gradients in a dense grid of the image. From previous works, it is observed that the distribution of local intensity gradients can represent an image. This is done by dividing the image into blocks and using each pixel's gradient values to make a histogram representation of the block. In the end, it uses descriptor blocks to normalize the image.

This transforms the image into a one-dimensional array. This array contains the same number of features for each image slice and can be seen as a feature vector. The images can then easily be represented as a matrix where each image is a row, and the column space is made out of the output-array.

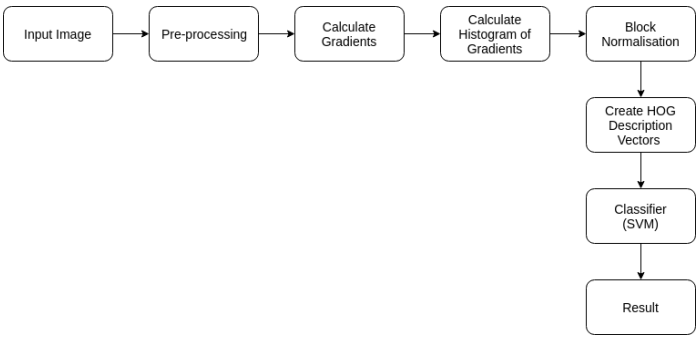


Figure 1: Overall method for extraction of HOG-features[2]

4.2 Creating histogram of oriented gradients

The histogram's creation is based on the gradients and magnitude for each pixel in one of the many blocks in an image. In an image, the highest amount of information is found around the edges. The edges tell us the shape of an object and give

us an indication of depth and texture in an image from the shadowing.

4.2.1 Edge detection

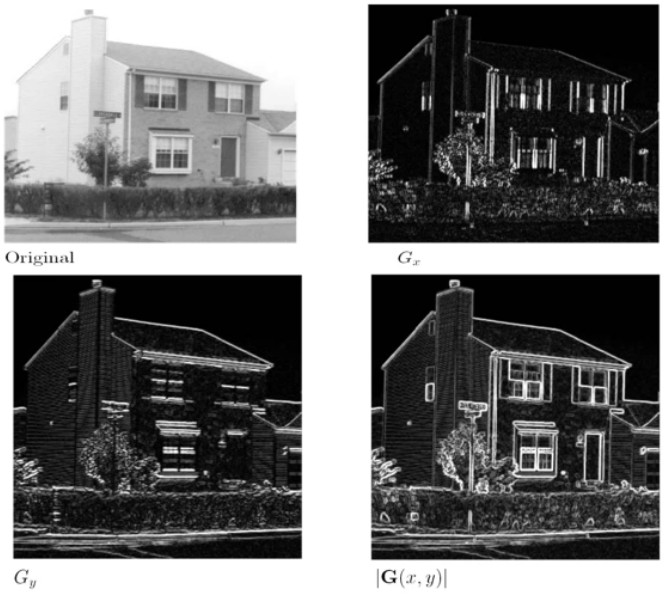


Figure 2: An image to represent the effect of edge detectors [3]

From Figure,2 it is possible to see the difference between the original image and the results after edge detection. It is still clear that there is a house that has been photographed. So it is possible to see that a lot of the information in the image is gone, but the main parts are still captured. Using this method reduces the amount of unnecessary information and still captures the essential parts. There are multiple ways to find the edges. The most simple method and what is used in this article is an approximation of the Sobel edge-detector.[4]

The Sobel filter uses a 3x3 filter that iterates over the image. This paper has used 2 matrices with dimensions: 1x3 and 3x1 to do the same. The two matrices calculate the gradients in the x and y dimensions. The filters look at the surrounding pixel-intensities. If the image is a color image, it will look at all the color channels at once and pick out the highest value. Below follows a representation of the filter used:

$$\begin{bmatrix}1 & 0 & -1\end{bmatrix} \begin{bmatrix}1\\0\\-1\end{bmatrix}$$

It is also possible to use more than just the neighbor pixels to calculate the gradients, which is done in this research for simplicity reasons. T2he two gradients have been combined to make a complete image by taking the maximum value from the two separate matrices in Figure. Instead of combining the X-gradient and Y-gradient matrices to plot the edge detection, they are used to find the gradient's angle and magnitude for each pixel.

In the paper from Dalal and Triggs[1] about detection humans, they considered a more complex filter, using both a 3x3 Sobel filter and diagonal filters and the combination of the diagonal filters. However, they all performed more poorly than the 1x3 and 3x1 filter shown above. They also tried to use a Gaussian smoothing before applying the filters but found no effects.

Dalal and Triggs tested other, more complex tasks, such as the 3x3 Sobel mask or diagonal masks, but these masks generally performed more poorly in detecting humans in images. They also experimented with Gaussian smoothing before applying the derivative mask but similarly found that omission of any smoothing performed better in practice.[1]

4.2.2 Calculation of magnitude and gradient

After the edge detection, there are two matrices, one gradient matrix and one direction matrix for all of the original image pixels. These two new matrices can be combined to calculate the gradient's orientation and magnitude for each pixel. The gradient and magnitude are given by[5]:

$$\Theta = \arctan(\tfrac{G_y}{G_x})$$

$$G = \sqrt{G_x^2 + G_y^2}$$

Where the same pixel location gives G_x and G_y from the two gradient matrices. This will result in two new matrices, one for each pixel's magnitude and one for the gradient of each pixel.

4.2.3 Creating of histogram

To compress the data, even more, it can be represented as a histogram. However, reducing the entire image down to one histogram will lead to a big data loss. Instead, the images are divided into blocks. The block-size is the same for the entire image, but different block-sizes will lead to different data compression. The bigger the blocks are, the more compressed the data will be. The histograms are calculated from the magnitude and gradient matrices. A nice illustration is given from OpenCVlearn[4]. An 8x8 block of the image is divided into a nine-bin histogram. Each of these nine bins represents 20 degrees, so in total, there can be 180 degrees. Here the magnitude will be negative if the orientation is on the other side of the 180-degree specter. It is also possible to increase the bin size to a 360-degree specter and have only positive magnitudes. The histograms are calculated using the magnitude and orientation matrices.

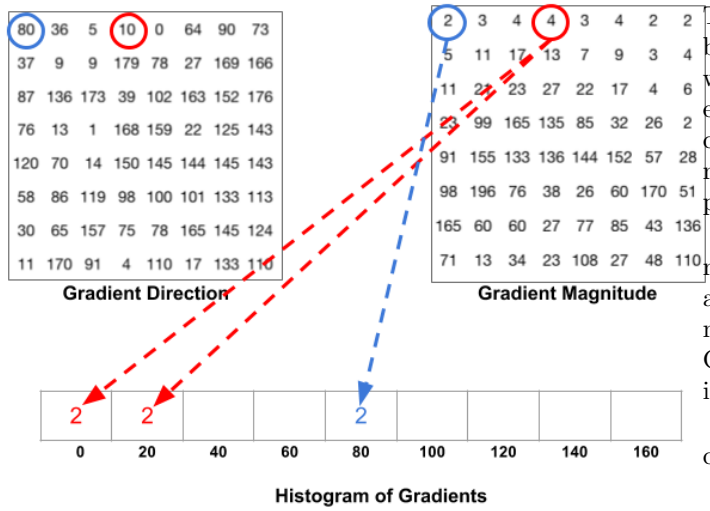


Figure 3: Calculation of histogram from magnitude and orientation matrices [4]

The magnitude of the pixels is placed in the bin that fits the pixel's orientation the most. If the orientation falls between two bins, it is divided into the two respective groups. How much of the magnitude that goes into each group is dependent on how close the orientation is to the two different bins. If it is exactly in the middle, it is split 50/50 into the two groups, as shown in Figure3.

Other options are adding the entire magnitude into the group that fits the most, not split it into different groups, or adding one to each bin that fits the gradient and not the magnitude. The third option is to divide the histogram into 180/360 bins so it doesn't have to split the magnitude. These options are all more clearly explained in Aishwarya Singh's paper Analytics Vidhya. [6]

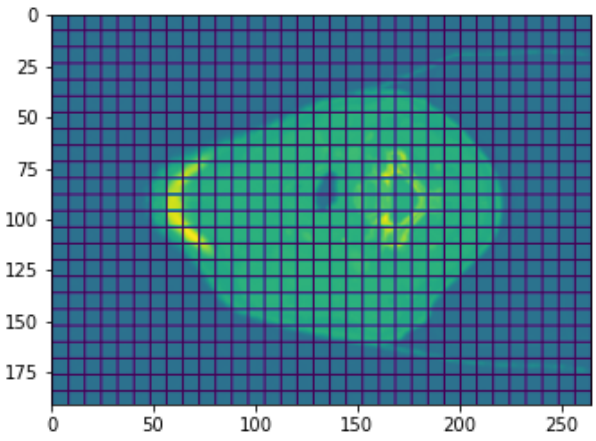


Figure 4: Image to visualize the block structure

4.3 Descriptor blocks and normalization

The gradient edges can be clear or stretch over multiple pixels. Both of the scenarios are really important to detect. That is why a local contrast normalization has shown to be the most successful way to normalize the data. In this way, local information doesn't get lost by normalizing the entire picture. With smaller normalization-groups, the local changes are taken more into account. Moreover, with bigger normalization-blocks, the clear and big edges will be better perceived.

There are several different ways to local contrast-normalize an image. Several of these normalization methods are investigated in this paper. A review of the two most common detector blocks used in HOG calculations, R-HOG and C-HOG, is presented. Both of them used by Dalal and Triggs in their "detection of standing people"-paper [1]

The detector blocks decide how much and what should be considered when normalizing patch by patch in the image.

4.3.1 R-HOG

The R-HOG is the most used descriptor block when using HOG. The R in R-HOG stands for rectangular. This descriptor block is also the simplest of the two taken into consideration. The detector blocks iterate over the image while normalizing one patch at a time. This makes it a local normalization. To catch up with the changes between blocks, it contrast normalizes with the neighbor blocks. This means that if a 3x3 normalization is used, it will take the eight blocks closest to the block that is being normalized. The nine bins in each of the histograms for the nine blocks will be used to normalize. The 81 normalized values (9bins * 9blocks) will be added to the block. This process is continued for each block in the image.

This makes the descriptor-blocks overlap so that all blocks will be used multiple times to normalize the surrounding blocks. This can be avoided by iterating with a bigger step-size, but it has positively affected the image[1]. Instead of having nine bins in each block, there are now 81 normalized values. This makes the data-set bigger. However, also more robust to noise.

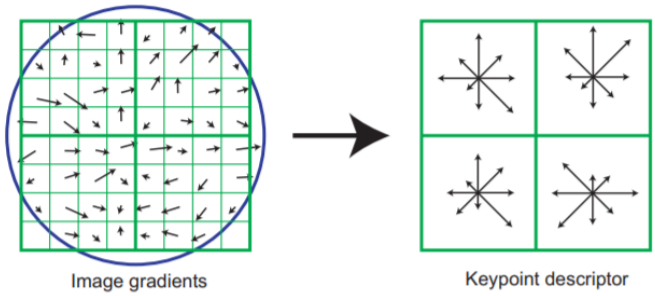


Figure 5: Image to visualise C-HOG vs R-HOG[7]

4.3.2 C-HOG

The C-HOG uses a circular detection-block instead of the rectangular, which is used in R-HOG. The C stands for circular. Instead of using blocks, it looks directly at the pixels' orientations. The idea came from Lowe's article released in 2004 [7]. The concept revolves around using the closest pixels

to the normalization center. In the example above, the center pixels are divided into 4 bins, used to normalize.

Lowe's article method was fundamental for Dalal and Triggs C-HOG, but they also added neighbor blocks to contrast normalization. They reached better results using the outer layer but not better results than the R-HOG.[1]

4.4 Normalization

There is four main block normalization scheme used in HOG. All of them are rather similar to L1 and L2. They are all used in Dalal and Triggs paper as well as they are the four built-in methods in SciKit-Image's HOG-function[1] [8].

The normalization concept is to re-scale the data into a scale between 0-1, so you get all the bins in the histogram into a common scale. This is done by finding the sum of the vector's values and dividing each value by it. How you calculate the total value is different for each function. With L2, you use the squared value:

$$L2 : \|\mathbf{W}\|^2 = \sum_{j=1}^{m} \mathbf{W}_j^2 \tag{1}$$

Here W is the sum of the vector squared. The vector is the bins in the histogram, M is the number of values, and j is the iteration factor, iterating through each of the histogram bins. Then each value is divided by the sum, which will be clearly shown later in the text.

A related approach we can use to reduce the model complexity is the **L1 regularization**

$$L1 : \|\mathbf{W}\| = \sum_{j=1}^{m} |\mathbf{W}_j| \tag{2}$$

In the L1-Regularization, we are taking the absolute values of the weights into account instead of the squared value used in the L2- regularization.

In the two examples, W_j is an unnormalized value, given by the histograms in each of the descriptor-blocks. W is the sum of the vector.

The normalized vector is calculated by dividing the unnormalized vector by the vector's sum and adding a penalty term. The penalty adds a value to the term, seen as the bias. The bias makes the prediction a bit "worse" but makes it easier to generalize and reduce the chance of overfitting.

L2-normal uses the square of every value in the vector, adding a penalty term and normalizing each number by this factor. Below is the function for the normalization where the penalty term ϵ is included. ϵ ranges from 0 and upwards. With zero penalties, the prediction can tend to overfit the data easier. With a high penalty term, important features can go missing.

$$W = \frac{W}{\sqrt{\|W\|_2^2 + \epsilon^2}} \tag{3}$$

L2-His is the second method and uses the same normalization as L2-normal but limits the maximum values to be less than or equal to 0.2. To avoid getting values to overrule the others.

L1 uses the absolute values instead of the square root as in L2. The L1's function has the same variables as L2 and is given by:

$$W = \frac{W}{(\|W\|_1 + \epsilon)} \tag{4}$$

L1-sqrt is the same as L1 followed by a square root:

$$W = \sqrt{\frac{W}{\|W\|_1 + \epsilon}} \tag{5}$$

This gives a normalized descriptor vector that can be seen as a probability distribution using the Bhattacharya distance between them. The Bhattacharya distance measures the similarity between two distributions.

4.5 Creating vector

The vector for each block is then added together to create one long vector. This vector can be used as a feature vector for each image, creating a matrix where the number of rows is equal to the number of pictures and columns equal to the number of features given from the HOG-vector. This matrix can be fed into a classifier and be used for prediction.

4.6 Theory behind image

PET and CT images have been used for a long time and complement each other very well. The PET is good at catching up with the parts where there is a lot of cell division, often found in the soft tissues. The CT looks more into the head structure and contains obvious images of the different tissues. Combining these two methods has provided good results for a long time.

4.6.1 PET-scan

Positron emission tomography (PET) uses radio-labeled substances, also known as "tracers." These are introduced to the body, and the PET camera catches up with their distribution.[9] These "tracers" will gather up where there is a lot of cell activity going on, catching the brain's chemical activity. The cancer cells have a higher metabolic rate level, which results in brighter spots than regular cells in the brain. These brighter spots are good for detecting the cancer cells, see if the cancer is spreading, checking if the treatment is working, and finding cancer recurrence.

4.6.2 CT-SCAN

Computer tomography is an x-ray operation. The CT fires X-rays into the head, and depending on how many of the X-rays are absorbed; it can tell what type of tissue it is. The more of the X-ray absorbed, the brighter the spot shown in the image. Like bones, hard tissues collect a lot of X-rays, while soft tissues like brain and cancer tumors absorb less. This scan can give a really detailed image of the head, with up to multiple thousands of gray tones. That can be hard to see on

a computer without looking at the exact gray-tones/specter under investigation. This can be achieved using windowing.

4.6.3 Windowing

A common way to pre-process the CT-images is called "windowing." Windowing is used to enhance special intensity levels in the images. This is mainly used to enhance the specter where the difference is small, like soft tissues where the intensity variation is little. This is done by looking at the specific gray-level specter of the image. A CT-image can contain up to several thousand grey levels. They are calculated from the number of X-rays absorbed or how much they attenuated at each location in the brain. The measurement is done in Houndsfield Units (HU). The more the X-rays attenuated, the brighter the spot will be. The brightest spots have the highest HU values and are usually bones and other hard tissues. These can range between a couple hundred to several thousand. Water is set to be at 0 HU, and the air is set to -1000HU.

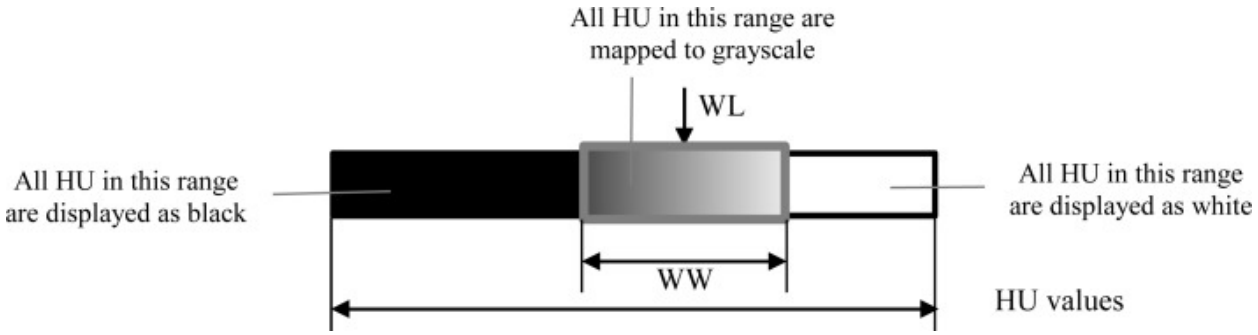


Figure 6: An illustration of the windowing method [10]

To enhance the contrast in the image and get a good representation of specific tissues, windowing can be used. Windowing will choose a specter from the CT image and represent it as a gray-level specter with intensity values from 0-255. All the brighter values than the specter will be shown as white, and all colors darker will be shown as black. Two parameters choose the specter; the center of the specter and the specter's width. The center is called WL, and the width is called WW in the figure6.

5 Results

This section presents the different results and compares them to the original images. All images are taken from the same slice of the same patient. All the images are gathered from the University of Oslo (UiO). The images are not in raw format. They have been worked on in several earlier written masters and doctoral degrees at the Norwegian University of Life Science. Most of the raw file's extraction work is done by Moe [11] and further worked on by Huynh [12].

The HOG images have been reduced, and only the clearest visible parts are left. The output of HOG is a vector, and the images are just a representation of the vectors. The image is represented by plotting the histogram for each block. The histogram is shown as a star histogram where the orientation has been taken into account. A larger bin in the histogram will be shown as a brighter and longer line in the image. This makes it possible to visualize both the direction and magnitude for each bin in each histogram.

5.1 CT-images

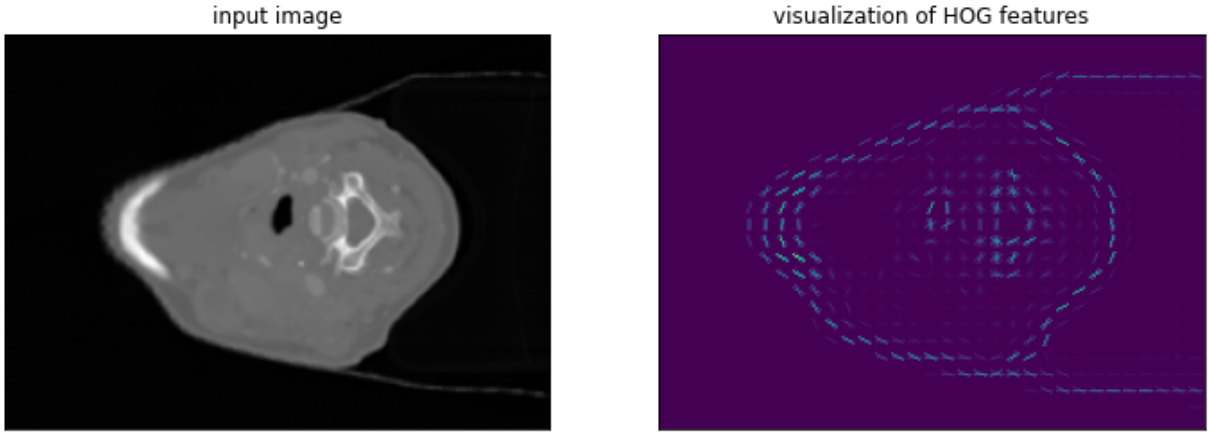


Figure 7: Original CT + HOG

In Figure 7, the result of using HOG on CT images is shown. The CT image on the left was used as the input, and the right is the result after HOG. The different parts of the brain and the bone structure can still be seen clearly. The brighter parts in the CT image are hard. Often these represent bones and other "hard" tissues. In this CT image, the left bright side is the jaw, and the middle bright part is the spine.

5.2 PET-images

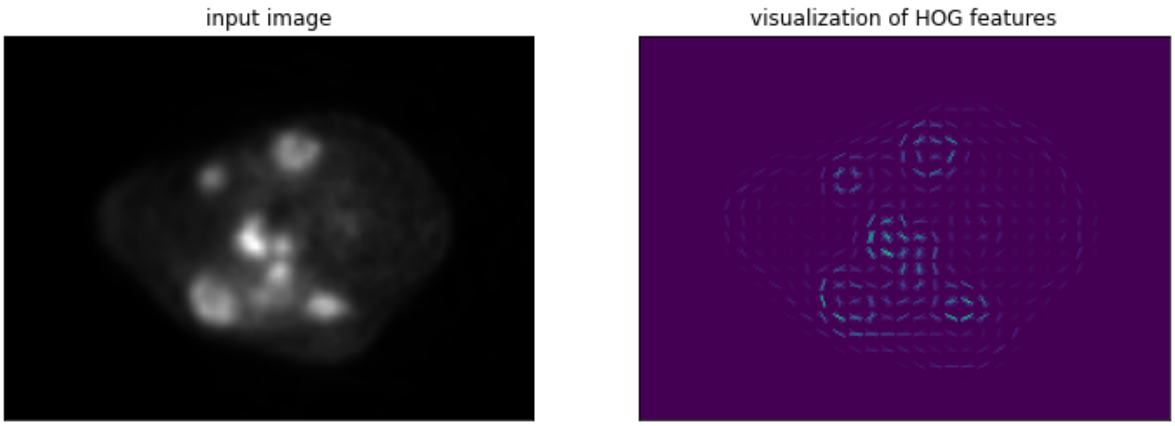


Figure 8: Original PET-scan and HOG

Figure 8 shows the PET scan on the right and the HOG on the left. The PET scan detects the brain's activity, and there can only be active cancers where there is cell division. This leaves a lot of noise out of the image, detecting only small parts of the head. This makes it easy for HOG to catch the edges. We can clearly see that the brightest parts are in the same area for the two images, as shown in Figur9, where the tumor is drawn in. It can be observed that the PET and HOG detect the changes and that the HOG image and tumor contour fit nicely.

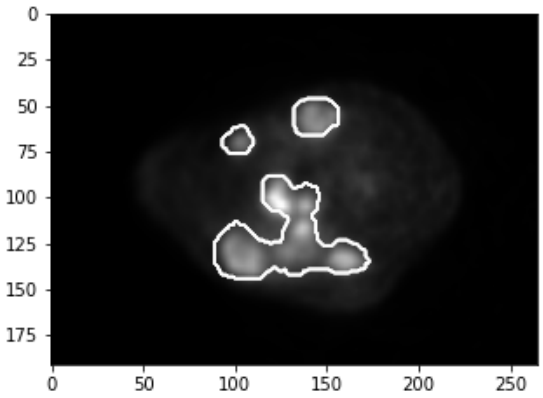


Figure 9: Combination of PET and Tumor

5.3 Combination

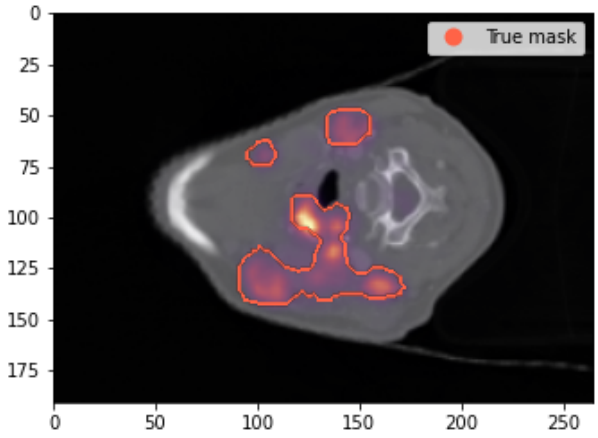


Figure 10: Combination from the different scans and the tumor prediction, using Yngve M. Moe's "MedVis" repository to show the images.[13] The red line is the tumor prediction, and the red heat map is gathered from the PET-scan. The gray parts are gathered from the CT-scans.

Figure 10 shows all the scans combined. Here the correlation between the PET-scan in red dots and the cancer markings(red line) can be clearly seen.

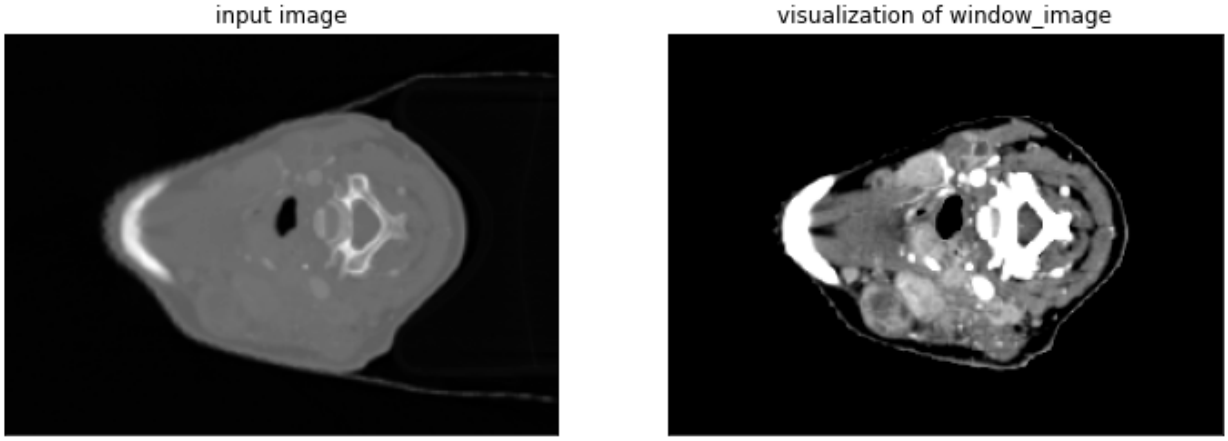


Figure 11: The image before and after windowing. Here the Window width is 200 and the center value is 1094. The method is gathered from Moe's Master Thesis[11]

Figure 11 shows the difference between the original image and the windowed image. There are plenty of different window sizes and center values to test out. This gives a representation of how the image can look like after the transformation. It is clearly observed that the contrasts get higher.

6 Discussion

The cancer prediction result is still not ready and will be presented in my Master Thesis alongside the analysis of the prediction methods. This section presents possible changes that could lead to better results, mainly focusing on the HOG-images and their output.

There are multiple factors in the creation of HOG-images. The blocks' size, normalization, what descriptor-block to use, and what normalization to apply are some of them. They all lead to different results. This section will also discuss other features that could be implemented to boost the possible prediction results.

6.1 Pre-processing

6.1.1 Normalization

The first section of the calculation of the HOG-Images is using an edge detector. However, some work could be done before this point. Normalizing the image before calculating the HOG vectors has been shown to make it more computationally demanding. Nevertheless, if it improves the prediction, it should definitely be a part of the calculation. Previously Dalal and Triggs tried out multiple methods as Gamma/power-law Normalization, square root, and log normalization, but only received a better result using Square Root normalization on color images and not on a grayscale.[1] However, this performance boost was so small that they decided to leave it out from the rest of the prediction.

The medical data does not contain color channels but a CT and PET- channel. This may lead to other results. The log normalization tends to be too strong and remove important features in the data. At least on delicate data where the gray-level in the image has a lot to say about where the tumors are. While the gamma/power and square-root normalization usually perform equally. Contrast normalization has been used for decades, and different normalization methods could lead to improvements.

6.1.2 Windowing

A second pre-process method is using different window sizes and center values. This can be used to either enhance the contrast or take with a bigger window-patch. With higher contrast, the edges will be clearer, which will lead to higher magnitudes. This affects the normalization is unclear, but it will hopefully make the tumors sharper and easier to predict. Some disadvantages are that it could make unnecessary information clearer or even remove important features of the images that can fall outside the window range.

6.2 Parameter tuning

6.2.1 Histogram parameters

After the pre-processing, the main parts that can be changed in creating the HOG-images are optimizing the HOG parameters. All of them will be tested against the results to see which of the parameters leads to the best possible results. The first parameters which can be changed are the orientations.

The orientations are mainly divided into two groups, signed or unsigned. Signed orientation uses a 360-degree specter and captures gradients in every orientation. Unsigned angels use a 180-degree specter and use negative magnitudes to replace the specter's angles. Unsigned angle specter has been used for a long time because it usually gives the same results or even better, and the computational cost is a bit less. The better results could come from the fact that the unsigned value uses the total magnitude. If some orientation points in the opposite direction of the rest, they will be over-ruled by the majority. This can help improve the estimation since often, these orientations are noise in an image. With signed values, these orientations are taken into their own bins. These bins are small but still there and can make a difference.

The block size is the number of pixels in each block. Here, testing will lead to the best parameter. However, with a too low block-size, the majority will not be as dominant to noise and could easily lead to overfitting. This will make it much more computational heavy. With two big blocks, the local information features might be gone because the domain features overrule the small details, and important information can get lost.

The bin-size is the number of bins in each histogram. With a low number of bins, some of the information can go lost, but with a two-high number of bins, the computational cost may become large. Doubling the bin-size will double the computational cost. It can also lead to overfitting and loss of the overall features. The descriptor blocks will try to avoid overfitting by combining with neighbor blocks. With good parameters from the bin size, the block normalization will be easier.

6.3 Normalization

After all the histograms are made, only the block-normalization is left before the prediction stage. The block normalization is divided into two sections, the detector blocks and the choice of normalization method.

The detector block decides the area that is being normalized over. The two methods used and described in the theory section are R-HOG and C-HOG. However, other blocks can be used and have been tested by others. The third option which has been considered is using vertical and horizontal detector blocks and the combination of them. This method is tried out earlier in the standing people paper by Dalal and Triggs without any luck [1]. This descriptor block's basic concept is that it will be a method in between the R-HOG and C-HOG. It uses square blocks, but just the closest ones. If 3x1 and 1x3 are used for vertical and horizontal descriptor blocks, the diagonal representation will use the four closest blocks when normalizing. This method will reduce computational demand. This will lead to a reduction in overlapping blocks. From Dalal and Triggs paper [1], They got a better result by adding overlapping blocks, but the reason for the improvement was unclear.

The two most common block-normalization methods are L1 and L2, where both have been used, and the difference in the HOG-images is minor but can lead to different predictions. It is not entirely clear which one that has the best performance. L1 tends to leave features out because of its absolute value penalty. Occasionally the L1 can leave out

important information or value the wrong features more. In that case, L2 would perform well because it does not discard features that easily. L1 tends to be more generalizing with sparse solutions than L2.

A combination of these two features is called an elastic net. Here you use a percentage of L1 and a percentage of L2. This could potentially lead to better results. The relation between L1 and L2 normalization can be chosen, and this is a parameter that can be tuned.

6.4 Dimension reduction

The dimension reduction is determined by the parameters that give the best prediction. The computational cost can vary a lot depending on the different variables. Below, there is a function for the number of variables given from the HOG function, where R-HOG is used as a descriptor block, and the step length is set to one.

$$NrF = \left\lfloor \frac{X}{X_b} - (X_n - 1) \right\rfloor * \left\lfloor \frac{Y}{Y_b} - (Y_n - 1) \right\rfloor * H * X_n * Y_n \tag{6}$$

N - Number of Features (The size of the Vector output from HOG)

X, Y - Is the image size

X_n, Y_n - descriptor block-size

X_b, Y_b - block-size

H - number of bins

The function tells us that with a big descriptor block or a small block-size, the Number of Features will become large.

Table 1: Show the number of features depending on what descriptor block-size and block-size used, the estimation is done with R-HOG as descriptor block type and step length equal to one

BS/DB	**1x1**	**2x2**	**3x3**
4x4	27198	107640	233280
8x8	6831	25344	52731
12x12	2970	10584	21060

The original image contains around fifty thousand pixel-values. The output vector from HOG is still huge. Extracting information from over 10 000 features can be hard and has a huge potential for overfitting. Even if most of the image does not contain any information and can be removed, there will still be many features to handle.

7 Conclusion

In conclusion, this paper has shown how HOG can be used to visualize medical data. It has discussed different pre-processing methods, how each parameter affects the result and how to optimize the HOG output. There are a lot of different variables to take into consideration. What parameters give the best result will be found from prediction when machine learning could be applied. Further work will be to get an image-segmentation from the HOG-image ready and train up the parameters. Afterward, include the CT-scans, PET-scans, and possibly more image-representation, e.g., Local Binary Patterns(LBP) in the prediction, to find new features the data can use to optimize the prediction.

References

[1] Dalal Triggs. Histograms of oriented gradients for human detection. 2005.

[2] Open Genus. Using histogram of oriented gradients (hog) for object detection.

[3] Sivaram Rasathurai. An analysis of edge detection depth.

[4] Satya Mallick. Histogram of oriented gradients.

[5] Dr. Ry. Histogram of oriented gradients (hog) — by dr. ry @stemplicity.

[6] Aishwarya Singh. Feature engineering for images: A valuable introduction to the hog feature descriptor.

[7] David G. Lowe. Distinctive image features from scale-invariant keypoints. 2004.

[8] SciKit-Image. Scikit-image feature module: Hog.

[9] Plus Medical. What is pet-ct?

[10] Sameer Antani PhD L. Rodney Long MA Dina Demner-Fushman MD PhD Zhiyun Xue, PhD and PhD George R. Thoma. Window classification of brain ct images in biomedical articles. 2012 Nov 3.

[11] Yngve Moe. Deep learning for automatic delineation of tumours from pet/ct-images. 2019.

[12] Bao Ngoc Huynh. Visualization of deep learning in auto-delineation of cancer tumors. 2020.

[13] Yngve Mardal Moe. Github: Medvis, tools to visulaize medical images.

8.2 Appendix B: Codes

8.2.1 CreateDataSet-Class

Following is the structure of the class used to generate the new datasets. The parent Class "createdataset" stores the functions and variables used to create both a train, test, val structured dataset and a dataset divided into folds. The Head Neck Cancer dataset is an example of a train, test, val structured dataset, while the Maastro dataset uses a k-fold structure.

The Parent class contain two sub classes shown further down in the code; the HNC containing the information peculiar for creating train, test, val structured datasets, while the Maastro class contains the functions peculiar for the k-fold structure.

```
import numpy as np
import h5py
import matplotlib.pyplot as plt
from skimage import data, color, feature


class create_data_set(object):
    def __init__(self):
        self.set_parameters

    def set_parameters(self, orientations=9, PPC=(2, 2), CPB = (3,
       3), block_norm='L2',
                       visualize=True, transform_sqrt=False,
                          feature_vector=False, multichannel=None,
                       radius=1, n_points = 8, window_center = 1094,
                           window_width = 200):
        # HOG-params
        self.orientations    = orientations
        self.PPC             = PPC
        self.CPB             = CPB
        self.block_norm      = block_norm
        self.visualize       = visualize
        self.transform_sqrt  = transform_sqrt
        self.feature_vector  = feature_vector
        self.multichannel    = multichannel
        # LBP-params
        self.radius = radius
        self.n_points = n_points
        # Datasets
```

```
27          self.WIN_CT  = False
28          self.MED_CT  = False
29          self.MED_PET = False
30          self.HOG_CT  = False
31          self.HOG_PET = False
32          self.LBP_CT  = False
33          self.LBP_PET = False
34          # Windowing
35          self.window_center = window_center
36          self.window_width = window_width
37
38
39      def HOG_calculation(self, image):
40          HOG = feature.hog(image, self.orientations, self.PPC, self.
                CPB, self.block_norm,
41                            self.visualize, self.transform_sqrt, self.
                                feature_vector, self.multichannel)
42          return HOG[1]
43
44      def LBP_calculation(self, image):
45          LBP = feature.local_binary_pattern(image, self.radius, self.
                n_points)
46          return LBP
47
48      def what_datasets(self, WIN_CT = False, MED_CT=False, MED_PET=
            False, HOG_CT=False, HOG_PET=False, LBP_CT=False, LBP_PET=
            False):
49          self.WIN_CT  = WIN_CT
50          self.MED_CT  = MED_CT
51          self.MED_PET = MED_PET
52          self.HOG_CT  = HOG_CT
53          self.HOG_PET = HOG_PET
54          self.LBP_CT  = LBP_CT
55          self.LBP_PET = LBP_PET
56
57      def windowing(self, image_window): # ww 100-300 try out 200-400
58          image_window = image_window - self.window_center
59          image_window[image_window < -self.window_width / 2] = -self.
                window_width / 2
60          image_window[image_window > self.window_width / 2] = self.
                window_width / 2
61          return image_window
62
63  class HNC(create_data_set):
64      def __init__(self):
65          super().__init__()
66
67
68      def gather_data(self,data):
```

```
        X_train = []
        X_test  = []
        X_val   = []

        if self.WIN_CT:
            X_train.append(self.Windowed_CT_train_img)
            X_test.append(self.Windowed_CT_test_img)
            X_val.append(self.Windowed_CT_val_img)

        if self.MED_CT:
            X_train.append(self.CT_train)
            X_test.append(self.CT_test)
            X_val.append(self.CT_val)

        if self.MED_PET:
            X_train.append(self.PET_train)
            X_test.append(self.PET_test)
            X_val.append(self.PET_val)

        if self.LBP_CT:
            X_train.append(self.LBP_CT_train_img)
            X_test.append(self.LBP_CT_test_img)
            X_val.append(self.LBP_CT_val_img)

        if self.LBP_PET:
            X_train.append(self.LBP_PET_train_img)
            X_test.append(self.LBP_PET_test_img)
            X_val.append(self.LBP_PET_val_img)

        if self.HOG_CT:
            X_train.append(self.HOG_CT_train_img)
            X_test.append(self.HOG_CT_test_img)
            X_val.append(self.HOG_CT_val_img)

        if self.HOG_PET:
            X_train.append(self.HOG_PET_train_img)
            X_test.append(self.HOG_PET_test_img)
            X_val.append(self.HOG_PET_val_img)

        X_train = np.moveaxis(X_train, 0, -1)
        X_test = np.moveaxis(X_test, 0, -1)
        X_val = np.moveaxis(X_val, 0, -1)
        return X_train, X_test, X_val

    def gather_target_data(self,data):
        train_y = np.asarray(data['train']['target'])
        test_y = np.asarray(data['test']['target'])
        val_y = np.asarray(data['val']['target'])
        return train_y , test_y, val_y
```

```
118
119     def create_file(self, save_path, file_name, data):
120         # Next get the shape of your data
121         img_dim1, img_dim2, num_channel = self.X_train.shape[1:]
122
123         # Finally create your file
124         path = save_path+ file_name
125         with h5py.File(path, 'a') as f:
126             train_group = f.create_group('train')
127             train_group.create_dataset('x', data=self.X_train, dtype
                    ='f4',
128                                        chunks=(1, img_dim1, img_dim2
                                               , num_channel),
129                                        compression='lzf')
130             train_group.create_dataset('y', data=self.train_y, dtype
                    ='f4',
131                                        chunks=(1, img_dim1, img_dim2
                                               , 1),
132                                        compression='lzf')
133
134             val_group = f.create_group('val')
135             val_group.create_dataset('x', data=self.X_val, dtype='f4
                    ',
136                                        chunks=(1, img_dim1, img_dim2
                                               , num_channel),
137                                        compression='lzf')
138             val_group.create_dataset('y', data=self.val_y, dtype='f4
                    ',
139                                        chunks=(1, img_dim1, img_dim2
                                               , 1),
140                                        compression='lzf')
141
142             test_group = f.create_group('test')
143             test_group.create_dataset('x', data=self.X_test, dtype='
                    f4',
144                                        chunks=(1, img_dim1, img_dim2
                                               , num_channel),
145                                        compression='lzf')
146             test_group.create_dataset('y', data=self.test_y, dtype='
                    f4',
147                                        chunks=(1, img_dim1, img_dim2
                                               , 1),
148                                        compression='lzf')
149
150
151     def create(self, save_path, file_name, data):
152         if self.WIN_CT:
153             self.Windowed_CT_train_img = np.stack([np.asarray(
                    windowing(im)) for im in data['train']['input'
```

```
                ][:,:,:,0]])
154             self.Windowed_CT_test_img = np.stack(np.asarray(
                    windowing(im)) for im in data['test']['input'
                    ][:,:,:,0])
155             self.Windowed_CT_val_img = np.stack(np.asarray(windowing
                    (im)) for im in data['val']['input'][:,:,:,0])
156
157         if self.MED_CT:
158             self.CT_train = np.asarray(data['train']['input'
                    ][:,:,:,0])
159             self.CT_test = np.asarray(data['test']['input'
                    ][:,:,:,0])
160             self.CT_val = np.asarray(data['val']['input'][:,:,:,0])
161
162         if self.MED_PET:
163             self.PET_train = np.asarray(data['train']['input'
                    ][:,:,:,1])
164             self.PET_test = np.asarray(data['test']['input'
                    ][:,:,:,1])
165             self.PET_val = np.asarray(data['val']['input'][:,:,:,1])
166
167         if self.LBP_CT:
168             self.LBP_CT_train_img = np.stack((np.asarray(self.
                    LBP_calculation(im, n_points, radius)) for im in
                    data['train']['input'][:,:,:,0]))
169             self.LBP_CT_test_img = np.stack((np.asarray(self.
                    LBP_calculation(im, n_points, radius)) for im in
                    data['test']['input'][:,:,:,0]))
170             self.LBP_CT_val_img = np.stack((np.asarray(self.
                    LBP_calculation(im, n_points, radius)) for im in
                    data['val']['input'][:,:,:,0]))
171
172         if self.LBP_PET:
173             self.LBP_PET_train_img = np.stack((np.asarray(self.
                    LBP_calculation(im, n_points, radius)) for im in
                    data['train']['input'][:,:,:,1]))
174             self.LBP_PET_test_img = np.stack((np.asarray(self.
                    LBP_calculation(im, n_points, radius)) for im in
                    data['test']['input'][:,:,:,1]))
175             self.LBP_PET_val_img = np.stack((np.asarray(self.
                    LBP_calculation(im, n_points, radius)) for im in
                    data['val']['input'][:,:,:,1]))
176
177         if self.HOG_CT:
178             self.HOG_CT_train_img = np.stack((np.asarray(self.
                    HOG_calculation(im)) for im in data['train']['input'
                    ][:,:,:,0]))
179             self.HOG_CT_test_img = np.stack((np.asarray(self.
                    HOG_calculation(im)) for im in data['test']['input'
```

```
                ][:,:,:,0]))
            self.HOG_CT_val_img = np.stack((np.asarray(self.
                HOG_calculation(im)) for im in data['val']['input'
                ][:,:,:,0]))

        if self.HOG_PET:
            self.HOG_PET_train_img = np.stack((np.asarray(self.
                HOG_calculation(im)) for im in data['train']['input'
                ][:,:,:,1]))
            self.HOG_PET_test_img = np.stack((np.asarray(self.
                HOG_calculation(im)) for im in data['test']['input'
                ][:,:,:,1]))
            self.HOG_PET_val_img = np.stack((np.asarray(self.
                HOG_calculation(im)) for im in data['val']['input'
                ][:,:,:,1]))

        self.X_train, self.X_test, self.X_val = self.gather_data(
            data)
        self.train_y, self.test_y, self.val_y = self.
            gather_target_data(data)

        self.create_file(save_path, file_name, data)


class maastro(create_data_set):
    def __init__(self,):
        super().__init__()


    def create_lists(self, data): # gj r generaliserende hvis tid
        # print(['Fold_{}'.format(i) for i in range(8)])

        input_0 = data['fold_0']['input']
        target_0 = data['fold_0']['target']
        ptidx_0 = data['fold_0']['patient_idx']

        input_1 =data['fold_1']['input']
        target_1 = data['fold_1']['target']
        ptidx_1 = data['fold_1']['patient_idx']

        input_2 = data['fold_2']['input']
        target_2 = data['fold_2']['target']
        ptidx_2 = data['fold_2']['patient_idx']

        input_3 =  data['fold_3']['input']
        target_3 =  data['fold_3']['target']
        ptidx_3 =  data['fold_3']['patient_idx']

        input_4 =  data['fold_4']['input']
```

```
        target_4 = data['fold_4']['target']
        ptidx_4 =  data['fold_4']['patient_idx']

        input_5 =  data['fold_3']['input']
        target_5 =  data['fold_3']['target']
        ptidx_5 =  data['fold_3']['patient_idx']

        input_6 =  data['fold_3']['input']
        target_6 =  data['fold_3']['target']
        ptidx_6 =  data['fold_3']['patient_idx']

        input_7 =   data['fold_3']['input']
        target_7 =  data['fold_3']['target']
        ptidx_7 =   data['fold_3']['patient_idx']

        self.input = [input_0, input_1, input_2, input_3, input_4,
            input_5, input_6, input_7]
        self.target = [target_0, target_1, target_2, target_3,
            target_4, target_5, target_6, target_7]
        self.ptidx = [ptidx_0, ptidx_1, ptidx_2, ptidx_3, ptidx_4,
            ptidx_5, ptidx_6, ptidx_7]


    def concat(self, X, element):
        if X == 0:
            X = element
        else:
            X = np.concatenate((X, element), axis=3)

        return X


    def get_fold(self, index):
        X = 0

        if self.WIN_CT:
            self.WIN_CT_list = np.asarray(self.WIN_CT_list)
            element = np.expand_dims(self.WIN_CT_list[index], axis
                =3)
            X = self.concat(X, element)

        if self.MED_CT:
            self.MED_CT_list = np.asarray(self.MED_CT_list)
            element = np.expand_dims(self.MED_CT_list[index], axis
                =3)
            X = self.concat(X, element)

        if self.MED_PET:
            self.MED_PET_list = np.asarray(self.MED_PET_list)
```

```
262             element = np.expand_dims(self.MED_PET_list[index], axis
                    =3)
263             X = self.concat(X, element)
264
265
266         if self.LBP_CT:
267             self.LBP_CT_list = np.asarray(self.LBP_CT_list)
268             element = np.expand_dims(self.LBP_CT_list[index], axis
                    =3)
269             X = self.concat(X, element)
270
271         if self.LBP_PET:
272             self.LBP_PET_list  = np.asarray(self.LBP_PET_list)
273             element = np.expand_dims(self.LBP_PET_list[index], axis
                    =3)
274             X = self.concat(X, element)
275
276         if self.HOG_CT:
277             self.HOG_CT_list  = np.asarray(self.HOG_CT_list)
278             element   = np.asarray(self.HOG_CT_list)
279             X = self.concat(X, element)
280
281
282         if self.HOG_PET:
283             self.HOG_PET_list  = np.asarray(self.HOG_PET_list)
284             element  = np.asarray(self.HOG_PET_list)
285             X = self.concat(X, element)
286
287         target = np.asarray(self.target)
288         y = target[index]
289
290         return X, y
291
292
293     def create_file(self, save_path,file_name):
294         # Loop through your data and create your dataset
295         path = save_path + file_name
296         print(self.get_fold(0)[0].shape)
297         dim1, dim2, num_channel = self.get_fold(0)[0].shape[1:]
298
299         for i in range(8):
300             with h5py.File(path, 'a') as f:
301                 group = f.create_group(f'fold_{i}')
302                 data_x, data_y = self.get_fold(i)
303                 group.create_dataset('input', data=data_x, dtype='
                        f4',
304                                      chunks=(1, dim1, dim2,
                                         num_channel),
305                                      compression='lzf')
```

```
306                     group.create_dataset('target', data=data_y, dtype='
                            f4',
307                                          chunks=(1, dim1, dim2,
                                                 num_channel),
308                                          compression='lzf')
309
310
311
312
313     def create(self, save_path, file_name, data):
314         self.WIN_CT_list  = []
315         self.MED_CT_list  = []
316         self.MED_PET_list = []
317         self.HOG_CT_list  = []
318         self.HOG_PET_list = []
319         self.LBP_CT_list  = []
320         self.LBP_PET_list = []
321         self.list = []
322
323         self.create_lists(data)
324
325         for k in self.input:
326             if self.WIN_CT:
327                 windowed_CT = np.stack([np.asarray(windowing(im))
                            for im in k[:,:,:,0]])
328                 self.WIN_CT_list.append(windowed_CT)
329
330             if self.MED_CT:
331                 MED_CT = np.asarray(k[:,:,:,0])
332                 self.MED_CT_list.append(MED_CT)
333
334             if self.MED_PET:
335                 MED_PET = np.asarray(k[:,:,:,1])
336                 self.MED_PET_list.append(MED_PET)
337
338             if self.HOG_CT:
339                 HOG_CT = np.stack((np.asarray(self.HOG_calculation(
                            im)) for im in k[:,:,:,0]))
340                 self.HOG_CT_list.append(HOG_CT)
341
342             if self.HOG_PET:
343                 HOG_PET = np.stack((np.asarray(self.HOG_calculation(
                            im)) for im in k[:,:,:,1]))
344                 self.HOG_PET_list.append(HOG_PET)
345
346             if self.LBP_CT:
347                 LBP_CT = np.stack((np.asarray(self.LBP_calculation(
                            im, n_points, radius)) for im in k[:,:,:,0]))
348                 self.LBP_CT_list.append(LBP_CT)
```

```
349
350             if self.LBP_PET:
351                 LBP_PET = np.stack((np.asarray(self.LBP_calculation(
                        im, n_points, radius)) for im in k[:,:,:,1]))
352                 self.LBP_PET_list.append(LBP_PET)
353
354         self.create_file(save_path, file_name)
```

Listing 8.1: Create Dataset Class

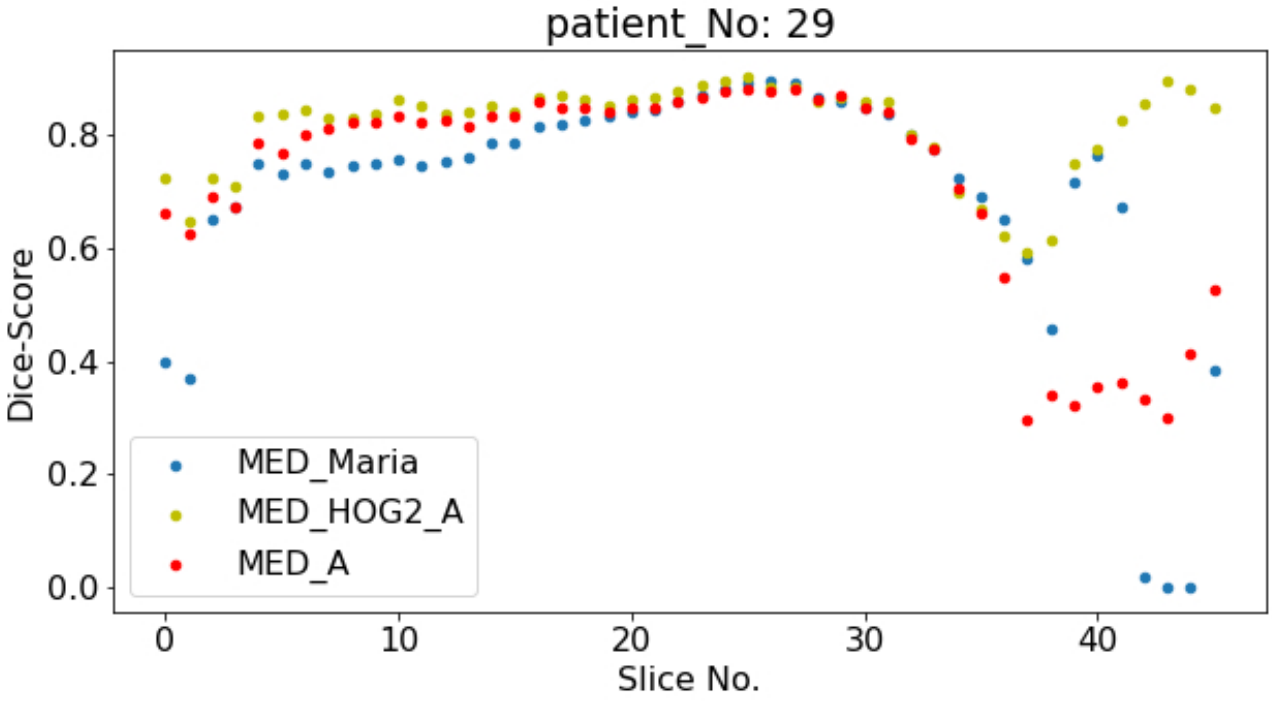
8.2.2 Example usage

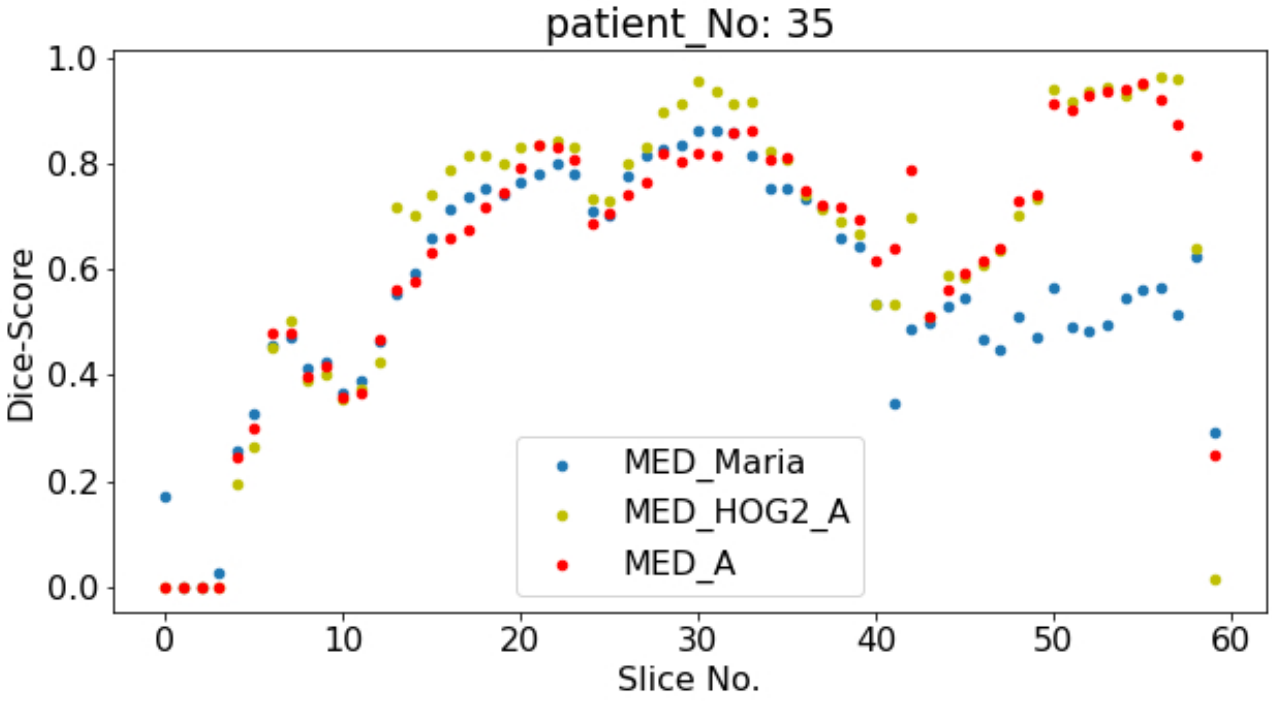
```
1  test = HNC()
2  test.set_parameters(orientations = 18,
3                      PPC = (2,2),
4                      CPB = (3,3),
5                      block_norm = 'L2-Hys')
6
7  test.what_datasets(MED_CT  = True,
8                     MED_PET = True,
9                     HOG_CT  = True,
10                    HOG_PET = True)
11
12 test.create(save_path = 'E:/OneDriveFreeSpace/Data/' ,
13             file_name = 'MED_HOG2_Ori_18.h5',
14             data=data)
```

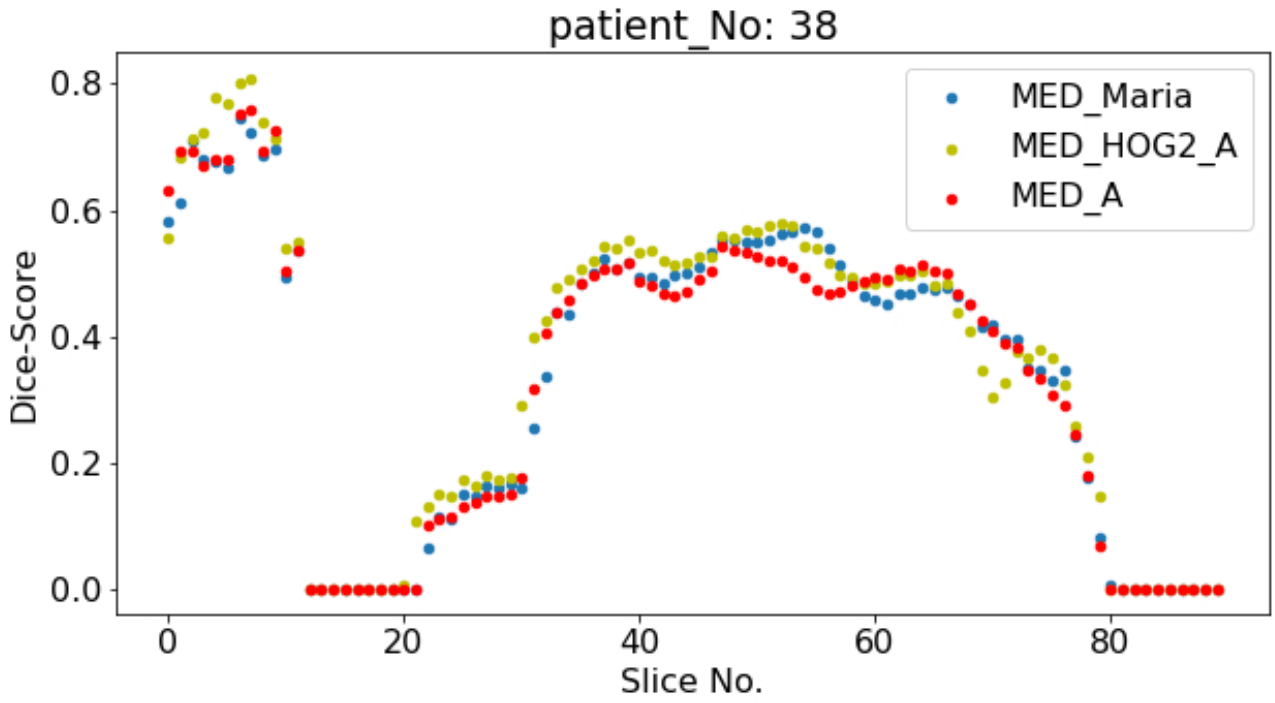
Listing 8.2: Example Code of how to use the class

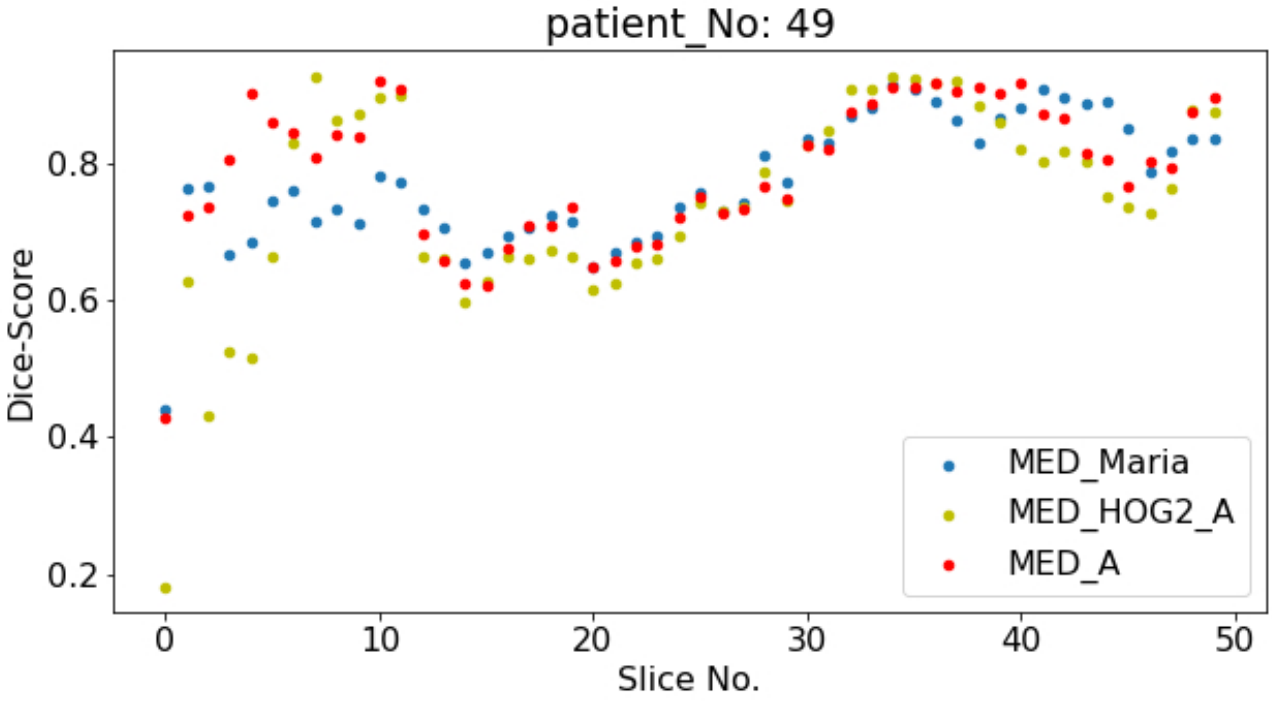
8.3 Appendix C: Extra plots

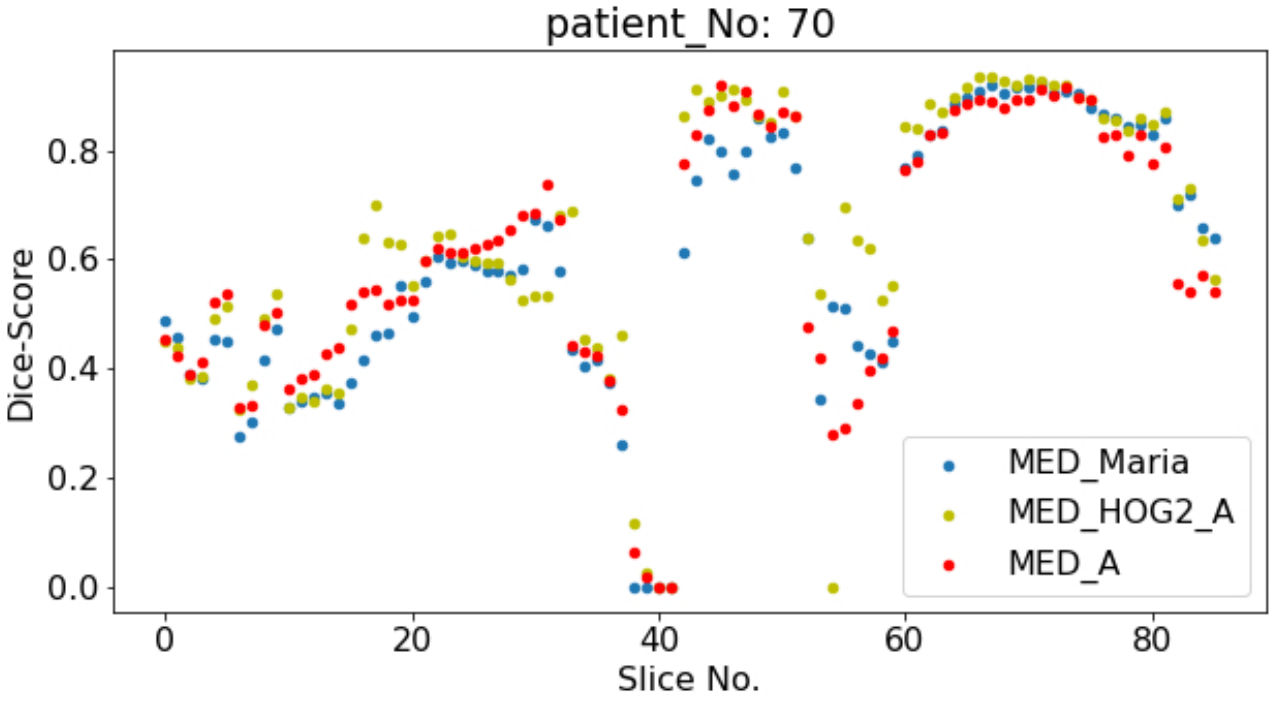
Slices per patient, second run

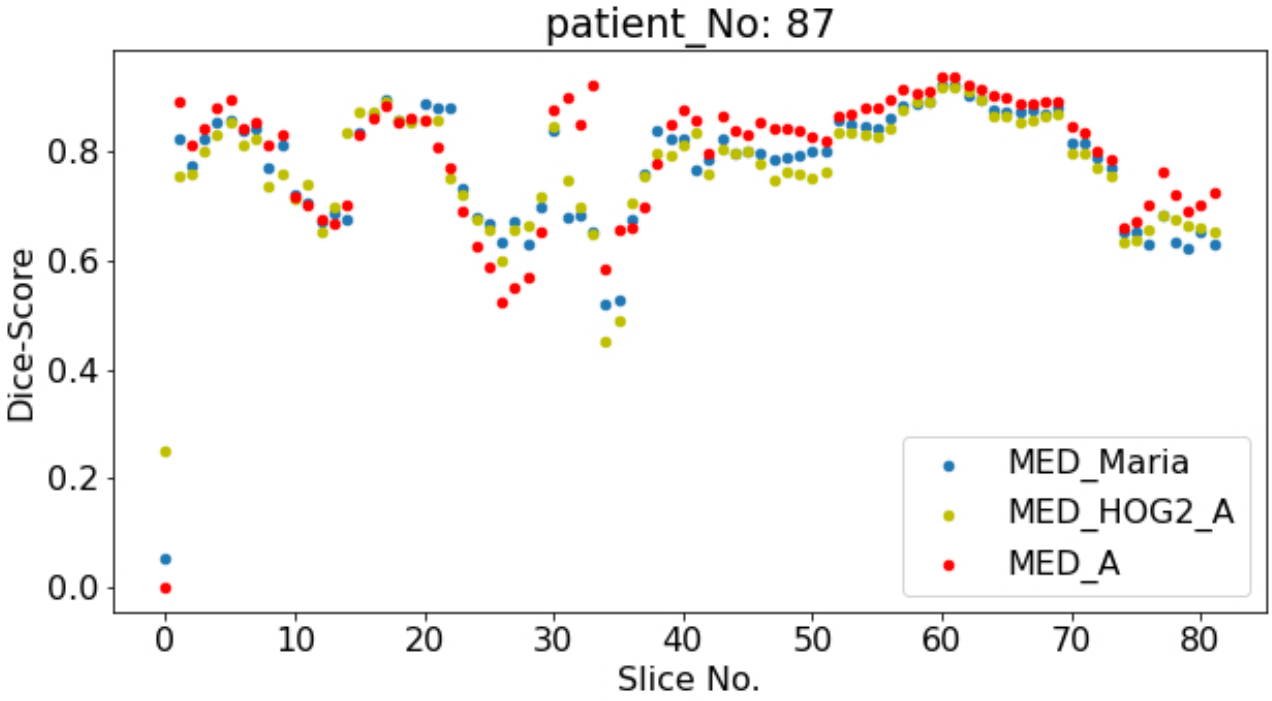

patient_No: 29
Dice-Score
0.0
0.2
0.4
0.6
0.8
0
10
20
30
40
Slice No.
MED_Maria
MED_HOG2_A
MED_A

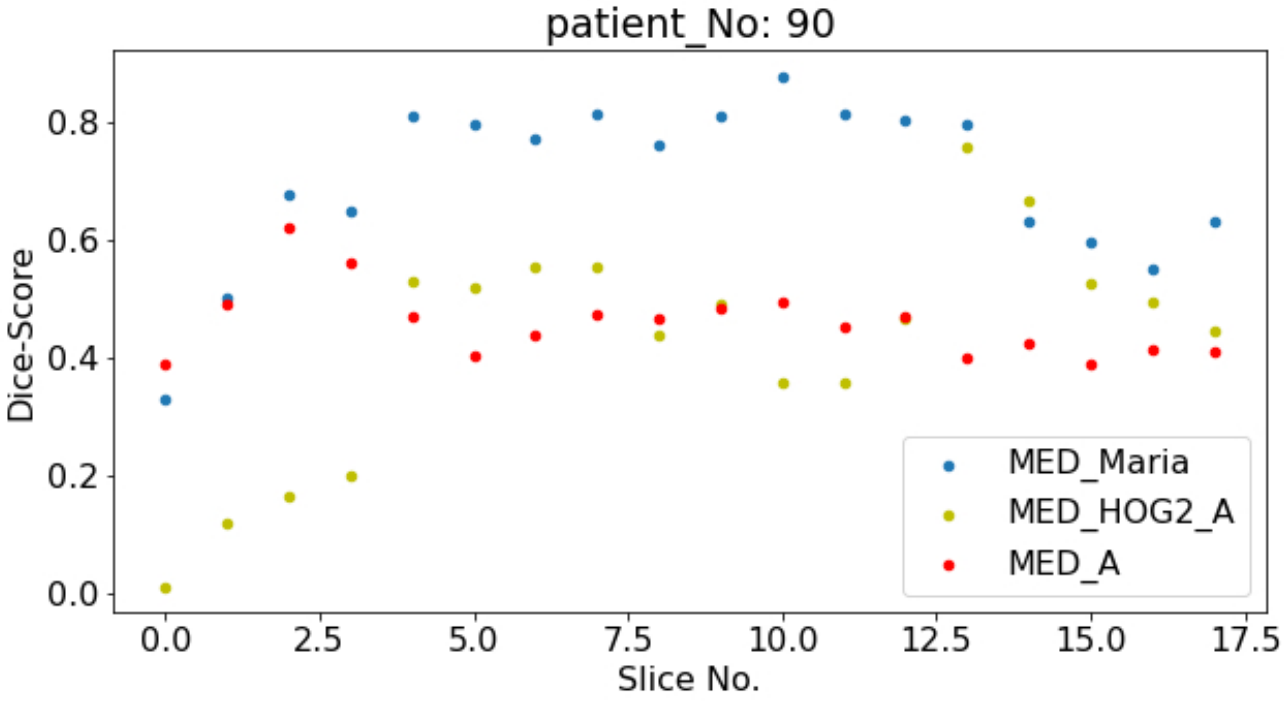

patient_No: 35
Dice-Score
Slice No.
MED_Maria
MED_HOG2_A
MED_A
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50
60

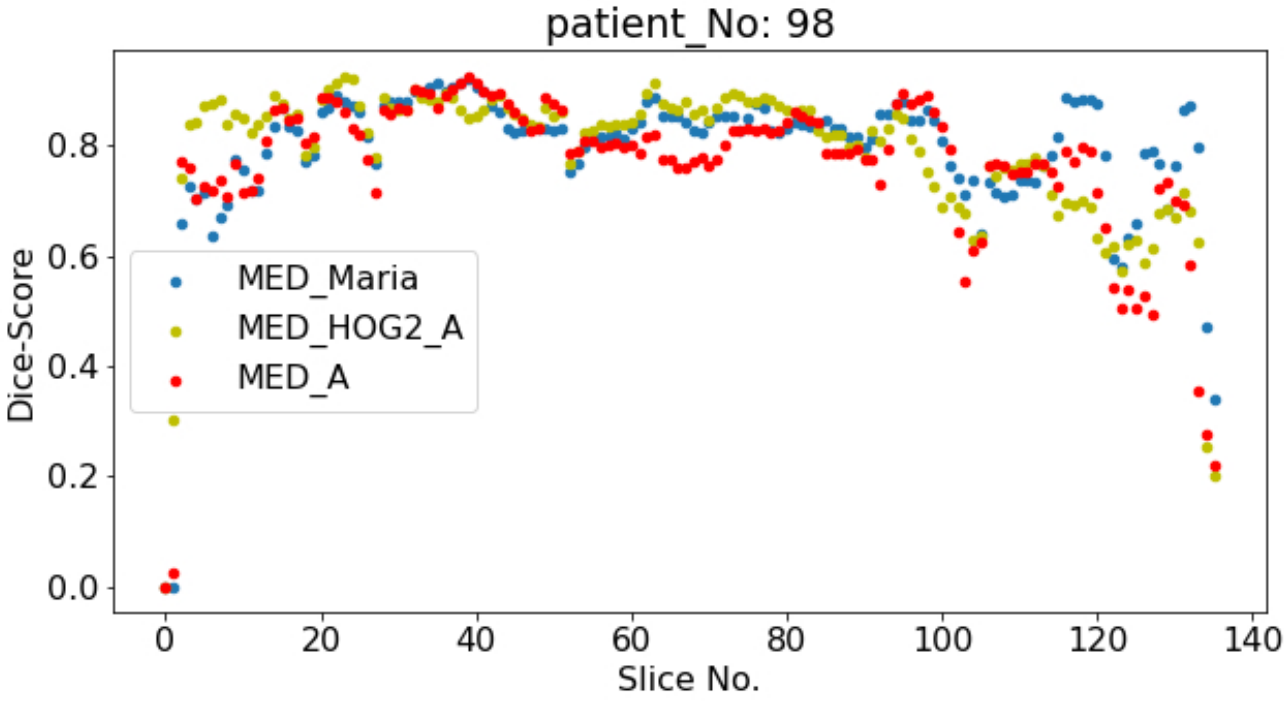

patient_No: 38
MED_Maria
MED_HOG2_A
MED_A
Dice-Score
0.0
0.2
0.4
0.6
0.8
0
20
40
60
80
Slice No.

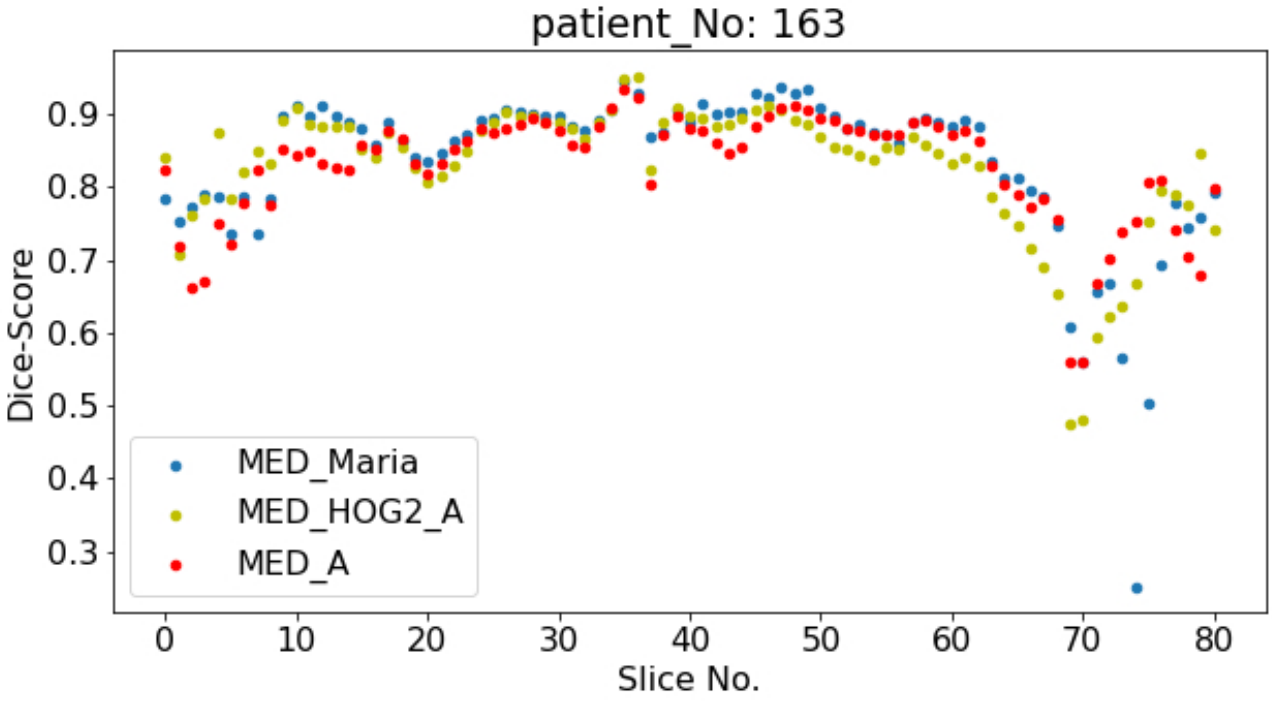

patient_No: 49
Dice-Score
0.2
0.4
0.6
0.8
0
10
20
30
40
50
Slice No.
MED_Maria
MED_HOG2_A
MED_A

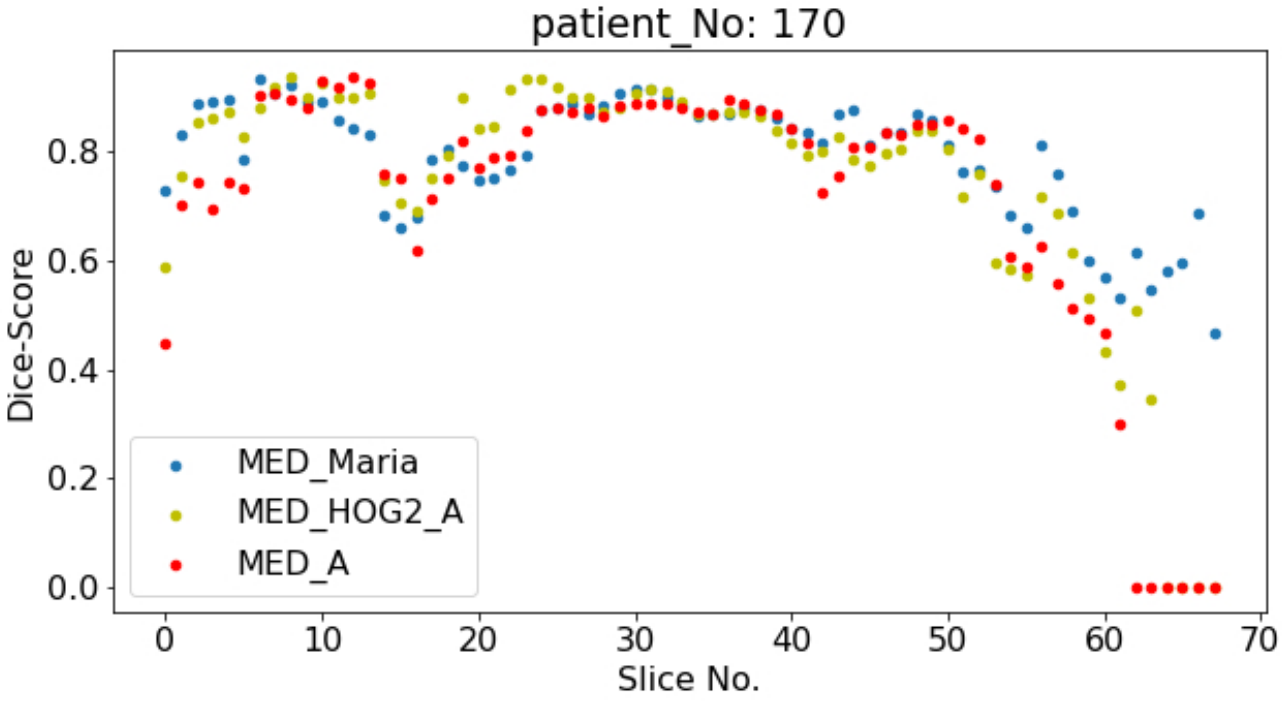

patient_No: 70
Dice-Score
0.0
0.2
0.4
0.6
0.8
0
20
40
60
80
Slice No.
MED_Maria
MED_HOG2_A
MED_A

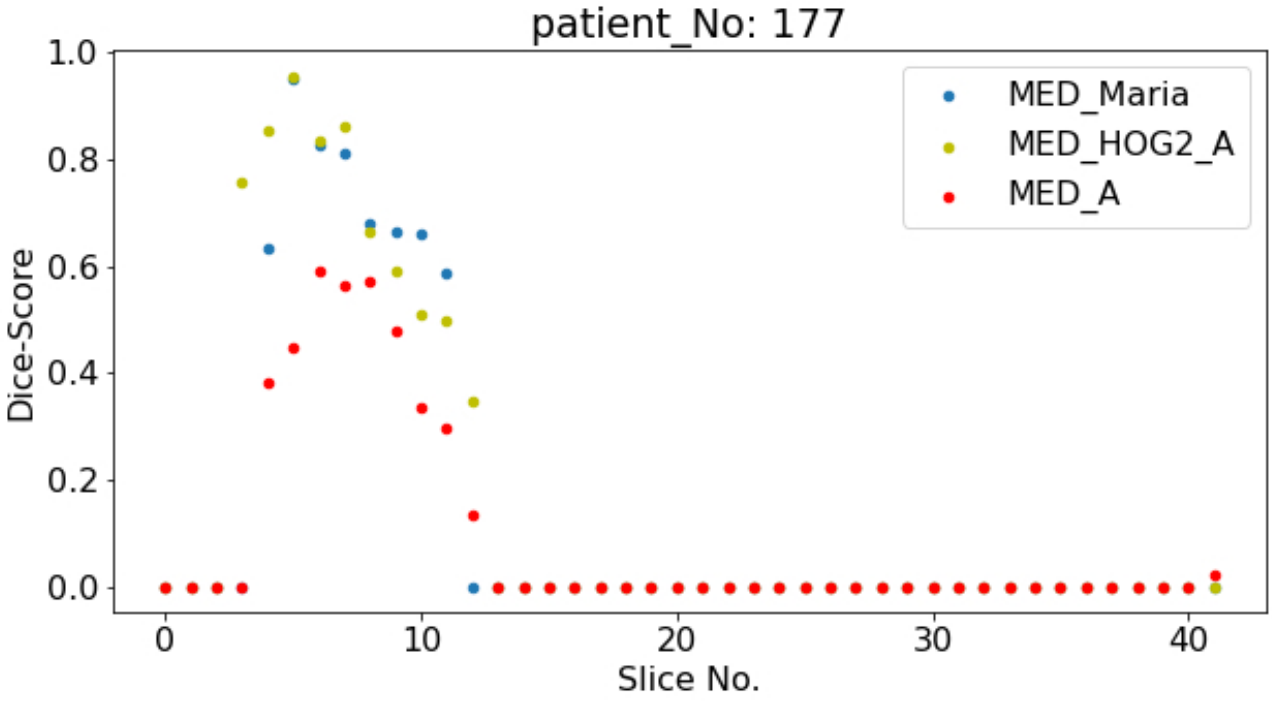

patient_No: 87
Dice-Score
0.0
0.2
0.4
0.6
0.8
0
10
20
30
40
50
60
70
80
Slice No.
MED_Maria
MED_HOG2_A
MED_A

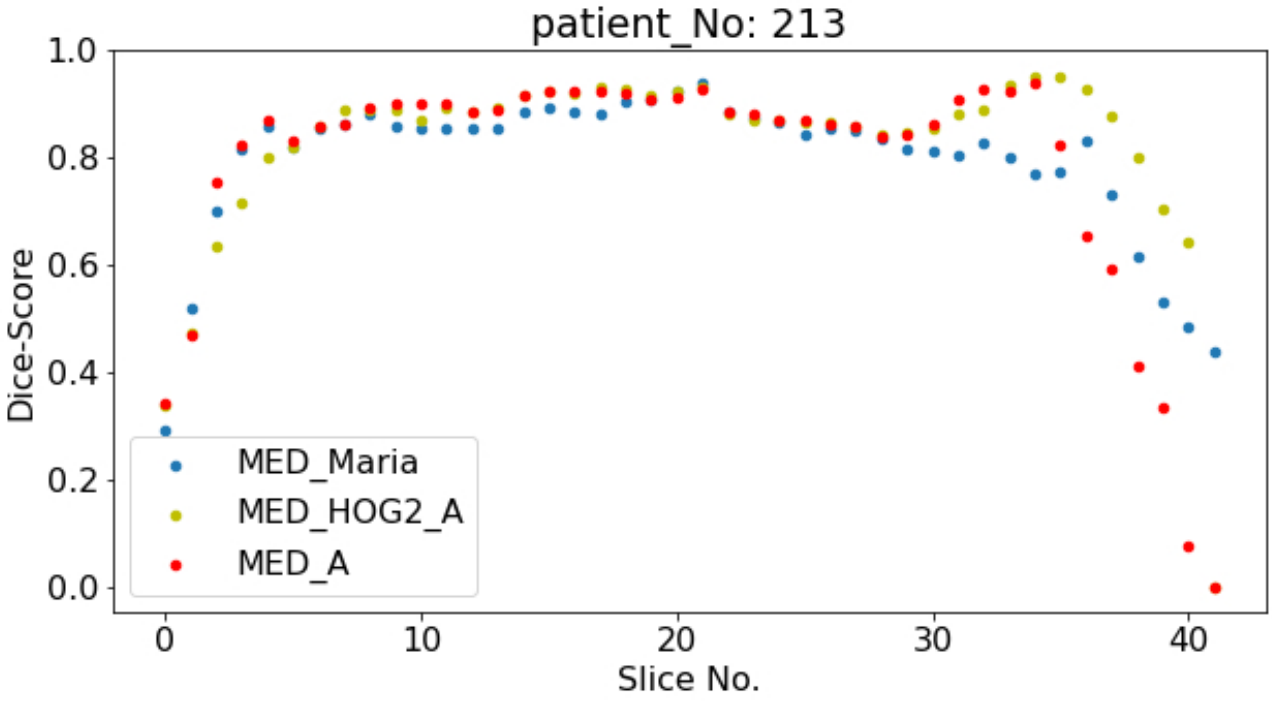

patient_No: 90
Dice-Score
Slice No.
0.0
0.2
0.4
0.6
0.8
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
MED_Maria
MED_HOG2_A
MED_A

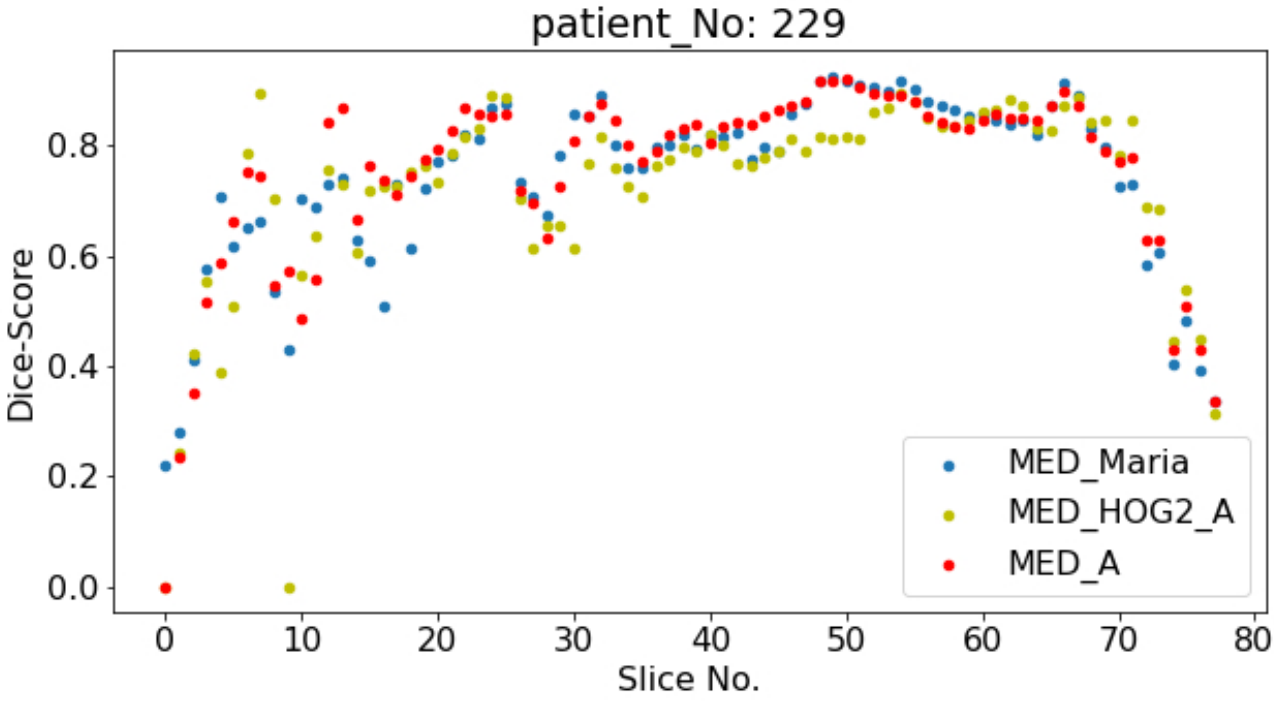

patient_No: 98
Dice-Score
Slice No.
MED_Maria
MED_HOG2_A
MED_A
0.0
0.2
0.4
0.6
0.8
0
20
40
60
80
100
120
140

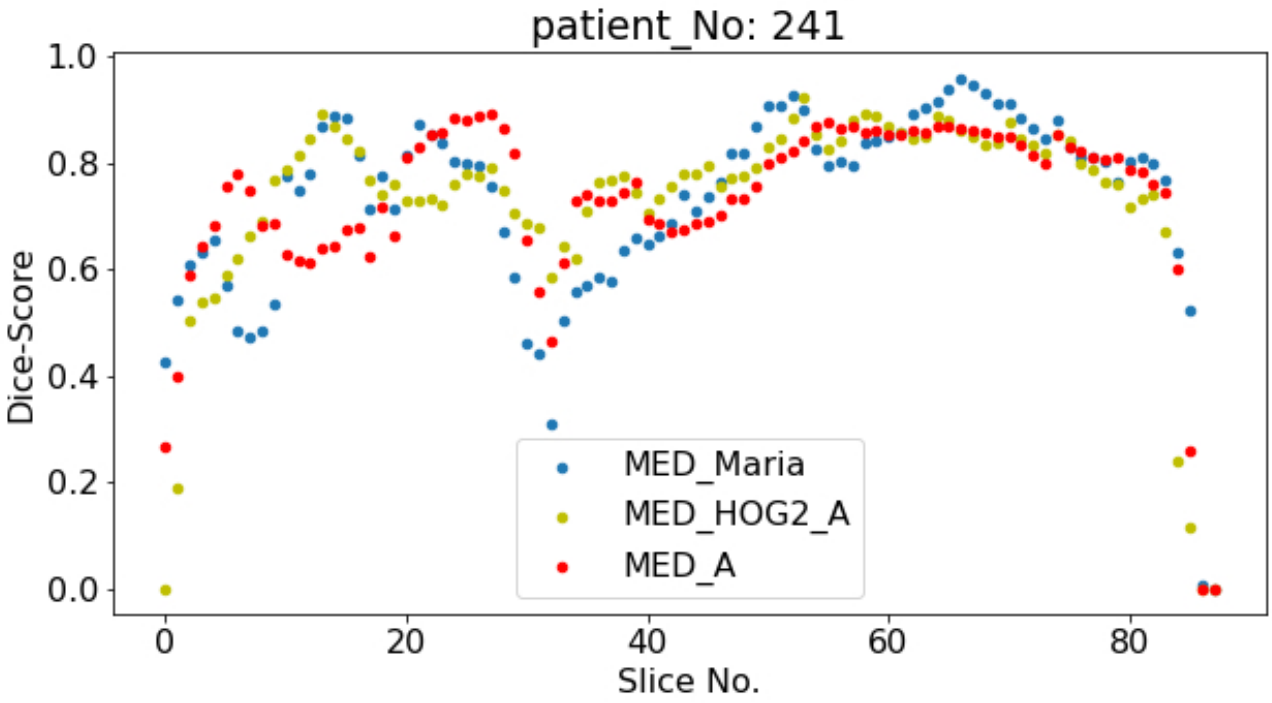

patient_No: 163
Dice-Score
Slice No.
MED_Maria
MED_HOG2_A
MED_A
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
10
20
30
40
50
60
70
80

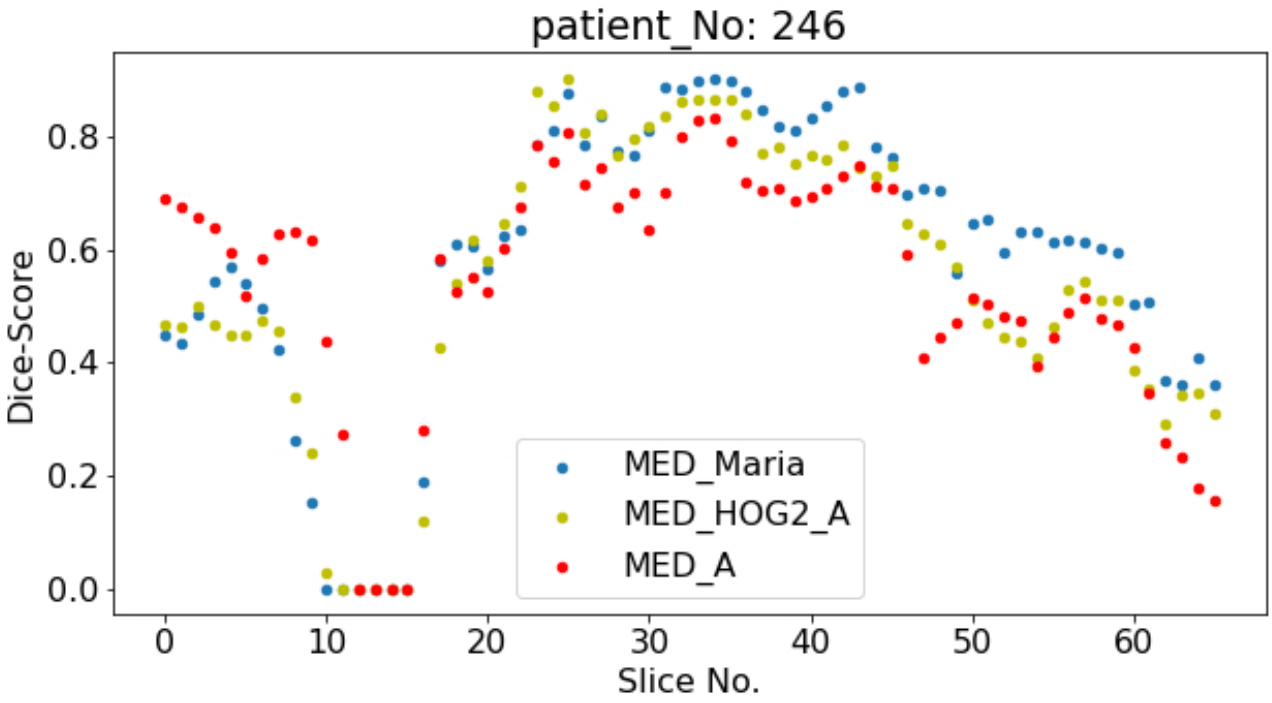

patient_No: 170
Dice-Score
Slice No.
MED_Maria
MED_HOG2_A
MED_A
0.0
0.2
0.4
0.6
0.8
0
10
20
30
40
50
60
70


patient_No: 177
MED_Maria
MED_HOG2_A
MED_A
Dice-Score
Slice No.
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40


patient_No: 213
Dice-Score
Slice No.
MED_Maria
MED_HOG2_A
MED_A
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40


patient_No: 229
Dice-Score
0.0
0.2
0.4
0.6
0.8
0
10
20
30
40
50
60
70
80
Slice No.
MED_Maria
MED_HOG2_A
MED_A


patient_No: 241
1.0
0.8
0.6
0.4
0.2
0.0
Dice-Score
0
20
40
60
80
Slice No.
MED_Maria
MED_HOG2_A
MED_A


patient_No: 246
Dice-Score
0.0
0.2
0.4
0.6
0.8
0
10
20
30
40
50
60
Slice No.
MED_Maria
MED_HOG2_A
MED_A

8.4 Appendix D: Example structure of Model-parameters

Model-Structure

An example of Model files used, stored in a JSON file. This is the model used to achieve the highest Val Dice Score.


```
{
    "dataset_params": {
        "class_name": "H5Reader",
        "config": {
            "filename": "/net/fs-1/home01/mgranhei/datasets/headneck
               /MED_HOG2.h5",
            "batch_size": 16,
            "x_name": "x",
            "y_name": "y",
            "batch_cache": 10,
            "shuffle": true,
            "fold_prefix": "",
            "train_folds": [
                "train"
            ],
            "val_folds": [
                "val"
            ],
            "test_folds": [
                "test"
            ],
            "preprocessors": [
                {
                    "class_name": "HounsfieldWindowingPreprocessor",
                    "config": {
                        "window_center": 70,
                        "window_width": 200,
                        "channel": 0
                    }
                },
                {
                    "class_name": "ImageNormalizerPreprocessor",
                    "config": {
                        "vmin": [
                            -200,
```

```
                            0
                        ],
                        "vmax": [
                            200,
                            25
                        ]
                    }
                }
            ],
            "augmentations": {
                "class_name": "ImageAugmentation2D",
                "config": {
                    "rotation_range": 90,
                    "zoom_range": [
                        0.8,
                        1.2
                    ],
                    "shift_range": [
                        10,
                        10
                    ],
                    "flip_axis": 0,
                    "brightness_range": [
                        0.8,
                        1.2
                    ],
                    "contrast_range": [
                        0.7,
                        1.3
                    ],
                    "noise_variance": 0.05,
                    "noise_channel": 1,
                    "blur_range": [
                        0.5,
                        1.5
                    ],
                    "blur_channel": 1
                }
            }
        }
    },
    "train_params": {
        "epochs": 5,
        "callbacks": [
            {
                "class_name": "EarlyStopping",
                "config": {
                    "monitor": "val_loss",
                    "patience": 30
```

```
                }
            }
        ]
    },
    "input_params": {
        "shape": [
            191,
            265,
            4
        ]
    },
    "model_params": {
        "loss": {
            "class_name": "BinaryFbetaLoss"
        },
        "optimizer": {
            "class_name": "adam",
            "config": {
                "learning_rate": 0.0001
            }
        },
        "metrics": [
            {
                "class_name": "BinaryFbeta"
            },
            {
                "class_name": "Dice"
            }
        ]
    },
    "architecture": {
        "type": "Unet",
        "layers": [
            {
                "class_name": "Conv2D",
                "config": {
                    "filters": 64,
                    "kernel_size": 3,
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                }
            },
            {
                "name": "conv2",
                "class_name": "Conv2D",
```

```
                "config": {
                    "filters": 64,
                    "kernel_size": 3,
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                }
            },
            {
                "class_name": "MaxPooling2D"
            },
            {
                "class_name": "Conv2D",
                "config": {
                    "filters": 128,
                    "kernel_size": 3,
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                }
            },
            {
                "name": "conv4",
                "class_name": "Conv2D",
                "config": {
                    "filters": 128,
                    "kernel_size": 3,
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                }
            },
            {
                "class_name": "MaxPooling2D"
            },
            {
                "class_name": "Conv2D",
                "config": {
                    "filters": 256,
                    "kernel_size": 3,
```

```
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                }
            },
            {
                "name": "conv6",
                "class_name": "Conv2D",
                "config": {
                    "filters": 256,
                    "kernel_size": 3,
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                }
            },
            {
                "class_name": "MaxPooling2D"
            },
            {
                "class_name": "Conv2D",
                "config": {
                    "filters": 512,
                    "kernel_size": 3,
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                }
            },
            {
                "name": "conv8",
                "class_name": "Conv2D",
                "config": {
                    "filters": 512,
                    "kernel_size": 3,
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
```

```
                    "class_name": "BatchNormalization"
                }
            },
            {
                "class_name": "MaxPooling2D"
            },
            {
                "class_name": "Conv2D",
                "config": {
                    "filters": 1024,
                    "kernel_size": 3,
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                }
            },
            {
                "class_name": "Conv2D",
                "config": {
                    "filters": 1024,
                    "kernel_size": 3,
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                }
            },
            {
                "name": "conv_T_1",
                "class_name": "Conv2DTranspose",
                "config": {
                    "filters": 512,
                    "kernel_size": 3,
                    "strides": 1,
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                }
            },
            {
                "class_name": "Conv2D",
                "config": {
                    "filters": 512,
                    "kernel_size": 3,
                    "activation": "relu",
```

```
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                },
                "inputs": [
                    "conv8",
                    "conv_T_1"
                ]
            },
            {
                "class_name": "Conv2D",
                "config": {
                    "filters": 512,
                    "kernel_size": 3,
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                }
            },
            {
                "name": "conv_T_2",
                "class_name": "Conv2DTranspose",
                "config": {
                    "filters": 256,
                    "kernel_size": 3,
                    "strides": 1,
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                }
            },
            {
                "class_name": "Conv2D",
                "config": {
                    "filters": 256,
                    "kernel_size": 3,
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                },
                "inputs": [
                    "conv6",
```

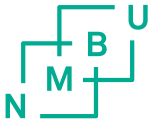
```
                    "conv_T_2"
                ]
            },
            {
                "class_name": "Conv2D",
                "config": {
                    "filters": 256,
                    "kernel_size": 3,
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                }
            },
            {
                "name": "conv_T_3",
                "class_name": "Conv2DTranspose",
                "config": {
                    "filters": 128,
                    "kernel_size": 3,
                    "strides": 1,
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                }
            },
            {
                "class_name": "Conv2D",
                "config": {
                    "filters": 128,
                    "kernel_size": 3,
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                },
                "inputs": [
                    "conv4",
                    "conv_T_3"
                ]
            },
            {
                "class_name": "Conv2D",
                "config": {
                    "filters": 128,
                    "kernel_size": 3,
```

```
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                }
            },
            {
                "name": "conv_T_4",
                "class_name": "Conv2DTranspose",
                "config": {
                    "filters": 64,
                    "kernel_size": 3,
                    "strides": 1,
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                }
            },
            {
                "class_name": "Conv2D",
                "config": {
                    "filters": 64,
                    "kernel_size": 3,
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                },
                "inputs": [
                    "conv2",
                    "conv_T_4"
                ]
            },
            {
                "class_name": "Conv2D",
                "config": {
                    "filters": 64,
                    "kernel_size": 3,
                    "activation": "relu",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                },
                "normalizer": {
                    "class_name": "BatchNormalization"
                }
            },
```

```
            {
                "class_name": "Conv2D",
                "config": {
                    "filters": 1,
                    "kernel_size": 3,
                    "activation": "sigmoid",
                    "kernel_initializer": "he_normal",
                    "padding": "same"
                }
            }
        ]
    }
}
```

Listing 8.3: Model Structure

Thank you.



Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway