Universitetet
i Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study program/specialization: Industrial Automation and Signal Processing	Spring semester, 2017 Open / Confidential: Open
Author: Rune Wetteland	..... Rune Wetteland ..... (signature author)
Instructor: Professor Kjersti Engang Supervisor(s): Professor Kjersti Engang (UiS), Jonatan Sjølund Dyrstad (SINTEF)	
Title of Master's Thesis: Classification of histological images of bladder cancer using deep learning Norwegian title: Klassifisering av histologiske bilder av blærekreft ved bruk av dyp læring	
ECTS: 30	
Subject headings: Bladder cancer, deep learning, autoencoder, neural networks, classification	Pages: 70 + attachments/other: 6 + attached zip file Stavanger, 15th of June, 2017 Date/year

Classification of histological images of bladder cancer using deep learning

by
Rune Wetteland

Master's Thesis

June 2017

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Under the Supervision of Professor Kjersti Engan

Abstract

In Norway bladder cancer is the fourth most common cancer type among men, with an almost 70 % increase in incidence the past four decades. For women, the increase has been about 40 %.

The histological images of bladder cancer are investigated by a pathologist to determine the grade and stage of cancer. In addition, the risk of recurrence and progression are also diagnosed. This is done manually by studying the histological images, but reproducibility of these results are low. To aid the pathologist, a proposed automatic system have been designed in this thesis consisting of six steps. Step one to four have been studied and experimented in detail, and step five and six are considered as future work.

The histological images are divided into smaller tiles, where each tile consists of one of several different categories; cancer tissue, damaged tissue, other tissue, blood or background. The aim is to make a system which automatically separates all tiles containing cancer tissue from the rest, as these have the potential to diagnose the cancer grade, stage, recurrence and progression.

To distinguish the different categories from each other, a classification system was constructed consisting of an autoencoder and a classifier trained in a semi-supervised fashion. The autoencoder was trained on 943,127 unlabeled tiles, extracted from seven histological images. Next, the encoder part of the autoencoder was connected to the classifier which was fine-tuned on 152,312 labeled images.

For evaluating the performance of the classifier, 10-fold cross-validation was calculated. Accuracy of the best classifier on a five class dataset was 97.7 % with a standard deviation of 3.2 %.

Preface

This thesis marks the end of the Master of Science degree at University of Stavanger, Department of Electrical Engineering and Computer Science. The thesis was conducted during the spring semester of 2017, and has not only been challenging, but also educational and exciting.

I am grateful for the opportunity I have gotten to be able to work with new technology, state-of-the-art hardware at my disposal at the University, and surrounded by people from several disciplines for continuous support.

I want to give a big thanks to my head supervisor professor **Kjersti Engang** for her excellent support and guidance during the thesis, and much-appreciated feedback throughout the entire master period.

Also, I would like to thank my co-supervisor **Jonatan S. Dyrstad** for his time and valuable inputs. Furthermore, I would like to thank **Emiel A.M. Janssen** and **Vebjørn Kvikstad** from Stavanger University Hospital for their help regarding the dataset and medical knowledge. At last, I would like to thank **Theodor Ivesdal** for his help and support with the UNIX network.

Stavanger, 15. June 2017
Rune Wetteland

Contents

List of abbreviations and nomenclature

CNN	Convolutional Neural Network
Conv	Convolution
DL	Deep Learning
Epoch	One forward pass and one backward pass of all the training examples
FC	Fully-connected
Hyperparameter	Parameters used to alter settings of network
Image tile	Small part of the original image
NN	Neural Network
PUNLMP	Papillary urothelial neoplasm of low malignant potential
px	Image Pixel
ReLU	Rectified linear unit
SGD	Stochastic gradient descent
Tensor	Multidimensional array
TensorFlow	An open-source software library for machine intelligence
TNM	Tumor, Node, Metastasis classification
WHO73	1973 World Health Organization classification of papillary urothelial neoplasms
WHO04	2004 World Health Organization classification of papillary urothelial neoplasms

1 Introduction

This chapter gives a motivation to the work of this thesis, as well as a brief history of artificially intelligence and use of this technology in medicine. Thesis objectives and structures are also presented.

1.1 Motivation and previous work

In 2015, 1626 people were diagnosed with bladder cancer in Norway. Of these, 1208 were men and 418 women. Bladder cancer is the fourth most common cancer type among men after prostate, lung and colorectal carcinomas [1]. Bladder cancer rarely develops for people below the age of 50, and usually the first diagnostic happens at the age 60-80 with median age of 70-74 years [2].

In Norway there has been an almost 70 % increase in bladder cancer incidence among men the past four decades, and approximately 40 % increase for women [2]. Globally bladder cancer resulted in 114,000 deaths in 1990. In 2010 this number was 170,000, which is an increase of 49 % [3].

For patients diagnosed with bladder cancer, 50-70 % will experience one or more recurrences, and 10-30 % will have disease progression to a higher stage [2]. Patient treatment, follow-up and calculating the risk of recurrence and disease progression depend largely on the histological grade and stage of the cancer. Correct prognosis of recurrence and progression is important to avoid under- or over-treatment of the patient, as well as unnecessary suffering and cost [4].

To correctly advise the cancer stage and grade, the histological images is a valuable resource. These images are analyzed manually by a pathologist, but due to the vast amount of information in the images it is both time consuming and difficult to process everything manually to retrieve the relevant information that is needed. Another problem is that the prognosis is both subjective and not very reproducible between pathologist. As stated by O.M. Mangrud "*In conclusion, the challenges of reproducibility and prediction of disease progression have not been resolved*" [2, p.61].

Development of computer systems that are more objective and reproducible are wanted to assist diagnosing of histological images [2]. As also stated by O.M. Mangrud "*Efforts to improve reproducibility have been made, but no new methods or additional biomarkers have gained wide accept for use in a clinical setting. It is therefore still important to search for methods which can enhance reproducibility and prognostic strength of the histological examination*" [2, p.13].

The traditional method of examination has been done by using a microscope, but have later been exchanged to digital microscope images. The whole-slide containing the cancer tumor is scanned using a digital slide scanner and the pathologist can

view the images on a computer. These images are capable of zooming in to 400x magnification like a traditional microscope. Tools have been developed to help the pathologist in their work, including image processing to automate some of the work. Examples are tools for counting cells, measure distance and mark specific areas and put comments on the image. However, no automatic detection and classification of cancer grade or stage have been developed.

Previous work
The problem of automatic classification of cancer grade or stage based on the histological images has been tried solved earlier. In 2016 a master thesis had the same dataset and faced the problem of trying to predict recurrence and disease progression. This thesis used an image processing technique called local binary pattern to predict recurrence and progression based on the texture in the images, but the overall results were low. This work is described in both [5] and an approved but not yet published article here [6].

The article "Automatic staging of bladder cancer on CT urography" [7] from 2016 describes a system which extracts morphological features and uses a linear discriminant analysis (LDA) classifier to predict the cancer stage. This study used CT urography images, and not histological images. The method achieved an 85 % accuracy based on images from 42 patients.

1.2 Image processing

Traditional digital image processing and computer vision use computer algorithms or software to change an image or to extract information. The goal of the processing can be to alter the appearance of the image by e.g. denoising or enhancement techniques. Other tasks can be to segment region of interest from images, and extract features from such regions by feature extraction techniques. Such features can be used together with classification and machine learning techniques to label images or regions of images.

ImageNet is an image dataset consisting of millions of photography images of different objects divided into 1000 different classes. The dataset is created to provide data for researchers to help develop more sophisticated models and algorithms, primarily in computer vision. Since 2010 ImageNet has arranged an annual contest called ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where teams compete to classify objects and scenes [8].

In the first years of the competition, all entries were based on traditional computer vision utilizing image processing, feature extraction, and classification schemes. In 2012 one team decided to try a new method called deep learning neural networks. This is a technique which utilizes machine learning for feature extraction and classification. As shown in Figure 1.1, the team was superior compared to its competitors.

The following years almost all entries were based on this technology.

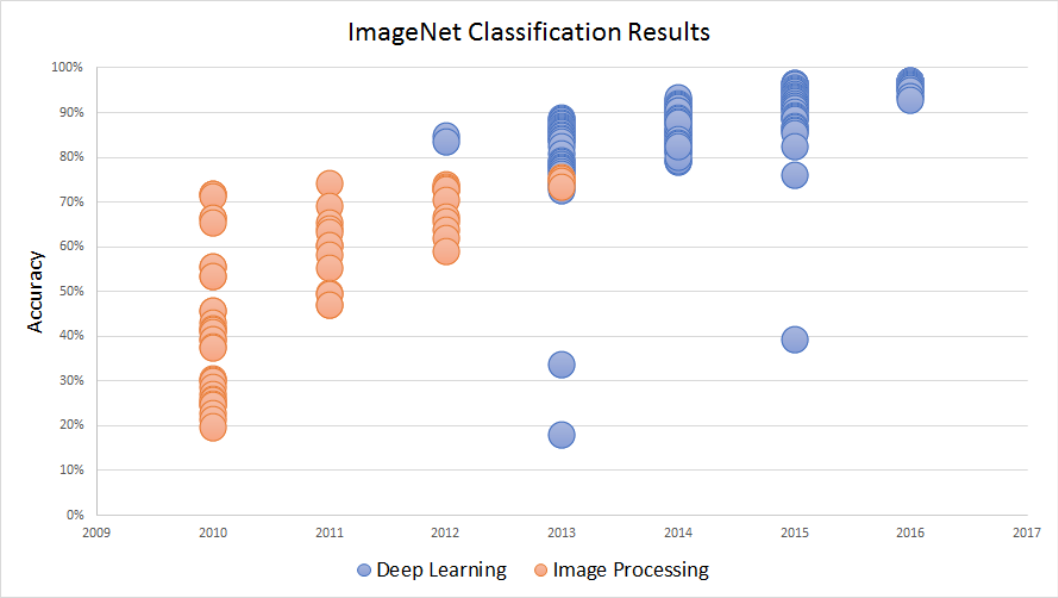


Figure 1.1: Traditional image processing vs. deep learning classification accuracy at the 2012 ImageNet competition. Chart is made with results available at www.image-net.org/

1.3 A brief history of artificial intelligence

Artificial intelligence is a field of computer science which tries to mimic human intelligence and behavior. Another field called machine learning, which is a subfield of artificial intelligence, was defined by Arthur Samuel in 1959 as the "*field of study that gives computers the ability to learn without being explicitly programmed*" [9, p.1]. An under-field of machine learning again called deep learning uses multiple layers neural networks for feature extraction and transformation [10]. Figure 1.2 visualizes how the different fields relate to each other [11].

According to Figure 1.1, deep learning may seem like a recent technology. But artificial intelligence, which deep learning is a subfield of, has a long history. In this chapter, some of the key moments of this history will be presented.

In 1943 the paper "*A logical calculus of the ideas immanent in nervous activity*" [12] was published by Walter Pitts and Warren McCulloch who had developed a technique called "thresholded logic unit" which was designed to mimic the neurons in the brain [12, 13].

In 1949 a small informal dining club called the Ratio Club was formed in Britain by the psychiatrist W. Ross Ashby. In his journal he wrote, "*We have formed a cybernetics group for discussion – no professors and only young people allowed in.*" [14, p.1]. It consisted of about twenty outstanding scientists and carefully selected

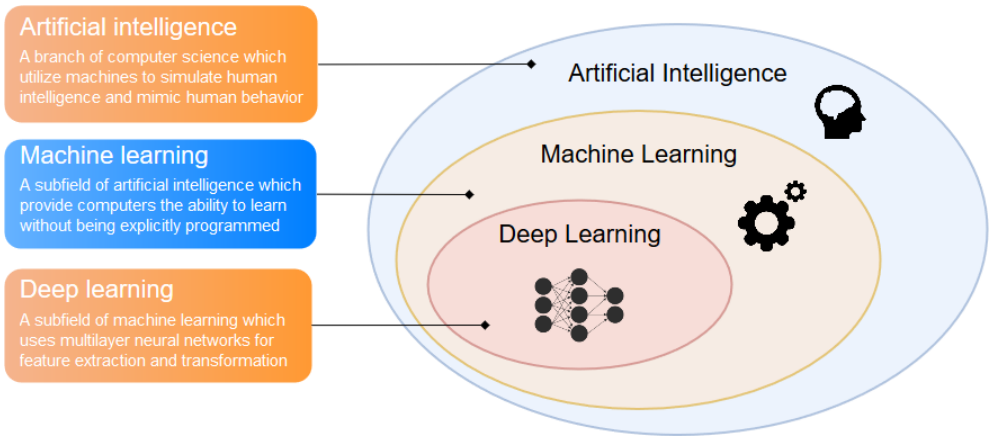


Figure 1.2: Artificial intelligence, machine learning, deep learning and how they relate to each other.

members of young psychiatrists, psychologists, physiologists, mathematicians and engineers [14]. Several of its members did research related to machine intelligence and brain modeling, and it is also stated that "*it is clear that the centre of gravity of the club was in the brain sciences.*" [14, p.6].

One of its members, mathematician Alan Turing, tackled the challenges of intelligent machines and in 1950 published his seminal paper "*Computing Machinery and Intelligence*" [15]. In the paper, Turing introduced the *Turing test* which is a set of criteria to see if a machine can be said to be intelligent. Alan Turing is universally regarded as one of the fathers of both computer science and artificial intelligence [14, 15].

In 1958 the psychologist Frank Rosenblatt published the paper "*The perceptron: A probabilistic model for information storage and organization in the brain*" [16] where he introduced the perceptron model. The perceptron was a simplified mathematical model of how the neurons in the brains operate, and was the first real precursor to modern neural networks. The perceptron consisted of one or more inputs, a processor, and a single output. Each input had randomly initialized weights associated with them, and by updating these weights during training the model could learn to converge to the correct solution of linear problems [13, 11]. An illustration of the original perceptron is shown in Figure 1.3.

Development of the perceptron continued through the 1960s, but in 1969 the book entitled "*Perceptrons: An introduction to computational geometry*" [17] was published by Marvin Minsky and Seymour Papert which put an end to this. Minsky and Seymour proved that the perceptron was theoretically incapable of learning non-linear functions like the XOR-function, no matter how long the model trained. This proof put a stop to research regarding neural nets, and the field entered a period known as the AI winter [11, 18].

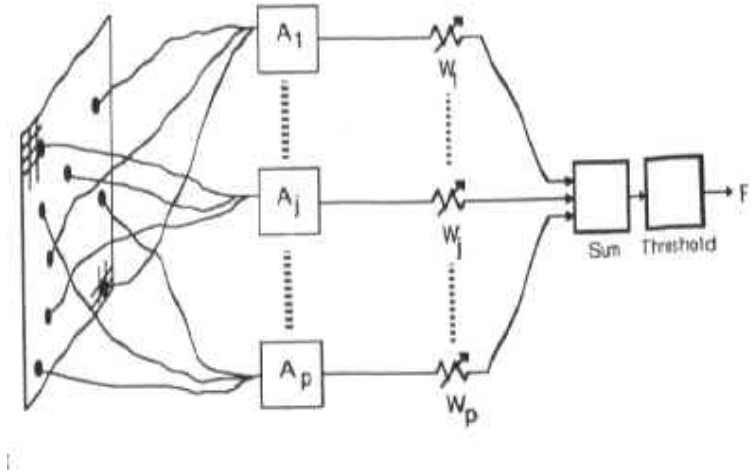


Figure 1.3: Frank Rosenblatt's original perceptron[1] [17]. The boxes to the right says *sum* and *threshold*.

In 1986, G. Hinton co-authored a paper along with D.Rumelhart and R. Williams entitled "*Learning representations by back-propagating errors*" [19]. They showed that neural networks with many hidden layers could effectively be trained using backpropagation, which would vastly improve the performance [18, 19].

In 1989 K. Hornik et al. published the paper "*Multilayer feedforward networks are universal approximators*" [20] which mathematically proved that using multiple layers would allow neural networks to learn any function, including non-linear functions like XOR. The results of this paper are known as the universal approximation theorem. In another paper by Hornik in 1991 he stated the following "*Hence, we conclude that it is not the specific choice of the activation function, but rather the multilayer feedforward architecture itself which gives neural networks the potential of being universal learning machines.*" [21, p.2] [11, 18].

In the late 1980s and early 1990s when Yann LeCun was working at Bell Labs, he utilized backpropagation to train a convolutional neural network called *LeNet*. The system was used to classify machine-printed and handwritten characters. Bell Labs deployed several of these systems in banks to automatically read checks, making it the first commercial application of a convolutional neural network. In an interview, LeCun said that "*At some point in the late 1990s, one of these systems was reading 10 to 20 % of all the checks in the US*" [22, p.1] [23].

In 2006, G. Hinton, Simon Osindero, and Yee-Whye Teh made a breakthrough with their paper "*A fast learning algorithm for deep belief nets*" [24]. They introduced the idea of unsupervised pre-training each layer as a Restricted Boltzmann Machine, before stacking all layers together as a deep belief net. This strategy allowed for deeper networks than before and achieved even better results [11, 24].

[1]Reprint permission granted by the MIT Press Subsidiary Rights Manager.

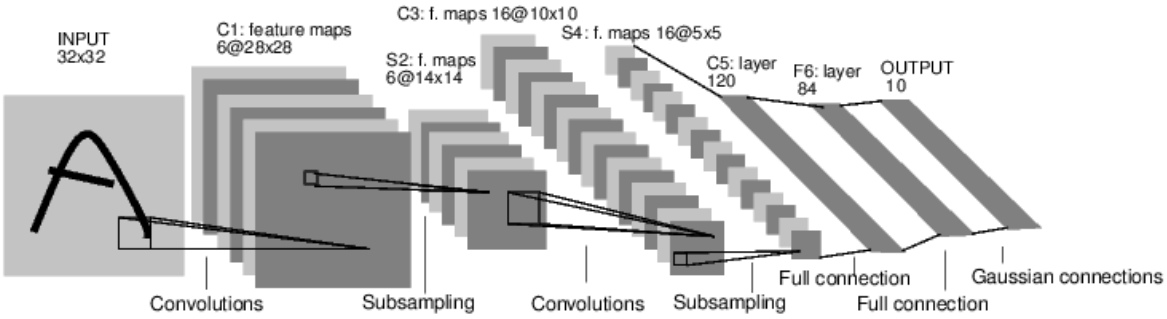


Figure 1.4: Architecture of LeNet, a convolutional neural network used for digits recognition[2]. Copyright 1998 IEEE [23].

By now, the core concepts behind deep learning was well established. As training dataset got larger and computers got faster, models got deeper and results better. In 2009 the first paper which utilized GPU's to train networks was published. They wrote that "*Our implementation of DBN learning is up to 70 times faster than a dual-core CPU implementation for large models*" [25, p.1].

In 2012 the paper "*ImageNet Classification with Deep Convolutional Neural Networks*" [26] was published by Alex Krizhevsky, I. Sutskever and G. Hinton. The paper described a deep convolutional neural network called *AlexNet*, which is the one used in the ILSVRC-2012 ImageNet competition. The architecture of the network is shown in Figure 1.5 below. In Figure 1.1 it is the only entry in 2012 using deep learning, and won far ahead of its opponents. This victory marked the abandonment of feature engineering, in favor of feature learning in computer vision tasks [11].

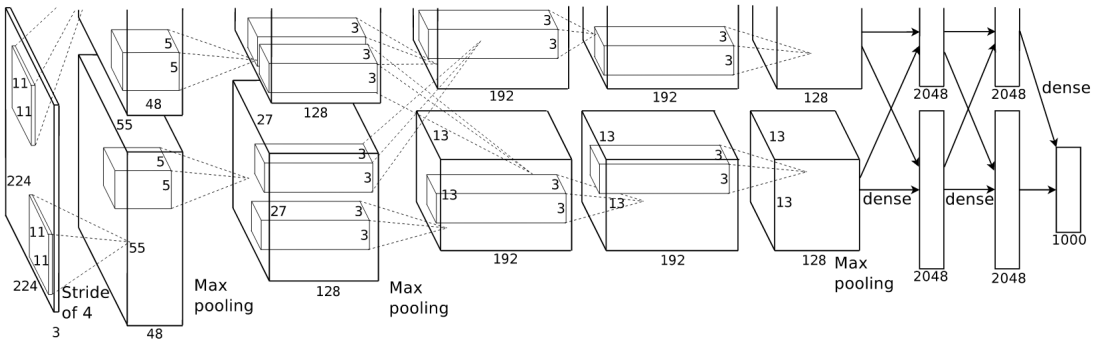


Figure 1.5: AlexNet, architecture of the convolutional neural network that won the ImageNet competition in 2012[3] [26].

[2]Reprinted according to general guidelines from IEEE

[3]Figure free to reprint without permission as according to Alex Krizhevsky webpape `www.cs.toronto.edu/~kriz/`

A summary of the key historical moments are presented in Figure 1.6. This brief history of artificial intelligence is by far inadequate of mentioning all events that have occurred and people who have contributed to the field.

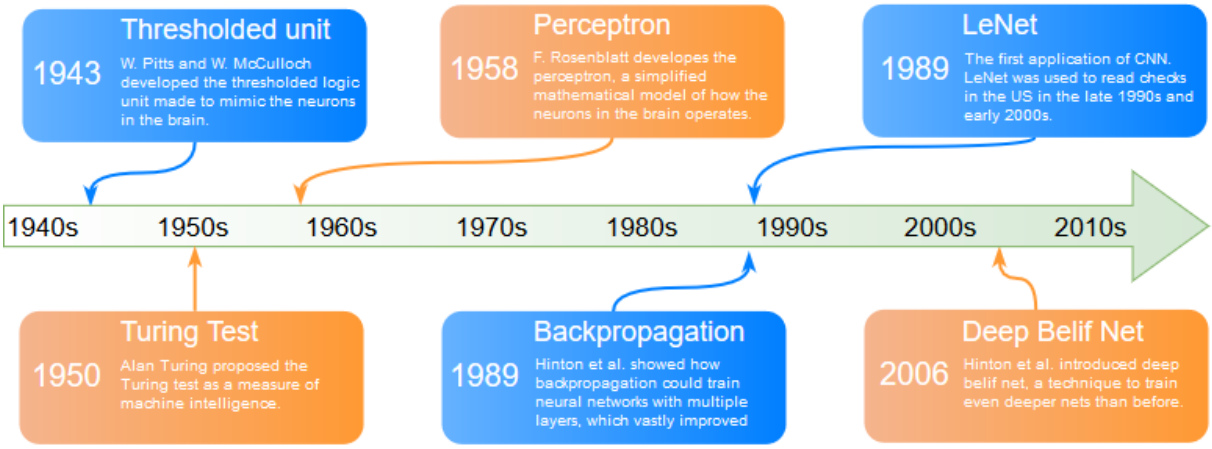


Figure 1.6: Timeline of artificial intelligence showing some of the key moments

1.4 Deep learning in medicine

The article "Dermatologist-level classification of skin cancer with deep neural networks" [27] trained a deep convolutional neural network on 129,450 clinical images of skin lesions. The system achieves performance equal to a dermatologist with AUC results ranging from 91 % to 96 % [27].

Another interesting topic is a new research project in Norway named *DoMore!*. The Norwegian Research Council has selected the project as one of the three winners of the prestigious Lighthouse Project grants [28]. The project says they "*Will teach computers, through Deep Learning and Big Data utilization to establish more robust grading systems in cancer types where pathology has failed. We will do so in an objective and reproducible way, reducing human error and removing subjective analyses, suboptimal diagnosis, and ultimately suboptimal treatment of cancer.*" [29, p.1].

A study using deep learning to automatically detect metastatic breast cancer in whole-slide images compared the results against diagnosis assigned by a pathologist. The software obtained an area under the receiver operating curve (AUC) of 0.925 for whole-slide classification, while the pathologist got a AUC of 0.966. They then combined the pathologist with the software and stated: "*the errors made by our deep learning system were not strongly correlated with the errors made by a human pathologist. Thus, although the pathologist alone is currently superior to our deep learning system alone, combining deep learning with the pathologist produced a major reduction in pathologist error rate*" [30, p.6]. Combining the pathologist and software increased the AUC to 0.995. It concludes with "*These results demonstrate*

the power of using deep learning to produce significant improvements in the accuracy of pathological diagnoses" [30, p.1].

Even though the goal for this thesis is to make a deep learning system to be used alone on histological images, the long-term aim should probably not be to try and replace the pathologist with a computer software, but rather provide them with the correct tools to improve their work.

1.5 Thesis objective

The primary objective is to make a system which utilizes deep learning techniques to automatically predict bladder cancer grade, stage, recurrence and disease progression based on the histological images.

A proposed system consisting of six dependent steps are presented in this thesis. Step one to four will be the main focus of this thesis, with step five and six as relevant future work.

The histological images mainly consist of cancer cells. However, some unwanted parts, which may influence the prediction of cancer grade and stage in a negative matter, are also present. Examples of these unwanted parts are damaged tissue, connective tissue, muscle tissue, blood, background, debris and similar. The main objective of step one to four is to design a system which is capable of distinguishing between these classes, and thus separate out all classes consisting of cancer cells.

1.6 Thesis structure

Chapter 2 - Background theory
This chapter provides an overview of relevant background theory used in this thesis. Bladder cancer and various deep learning techniques are reviewed. In chapter 2.3 the data material used in the thesis is presented.

Chapter 3 - Method
An overview of the proposed system developed during this thesis work is presented.

Chapter 4 - Experiments and results
This chapter presents the experiments conducted. The choice of each experiment is based on the result of the previous experiment, thus both experiment and results are presented together.

Chapter 5 - Discussion
An analysis of the Python script used is presented here. Afterward, the experimen-

tal results from the previous chapter are discussed. Suggested improvements and recommendations for further work are also included here.

Chapter 6 - Conclusion
The final conclusions of this thesis work is presented in Chapter 6.

2 Background theory

This chapter provides an overview of relevant background theory used in this thesis. Bladder cancer and various deep learning techniques are reviewed. In chapter 2.3 the data material used in the thesis is presented.

2.1 Bladder cancer

Bladder cancer is a disease in which abnormal cells multiply without control and form tumors in the urinary bladder. Tumors may be found anywhere within the bladder, but are most common along the lateral walls [2]. The majority of bladder cancer incidents are urothelial carcinoma with as much as 90 % in some regions. Other, and less common, bladder cancer types are squamous cell carcinomas, adenocarcinomas and neuroendocrine carcinomas [2].

When a patient is diagnosed with urothelial carcinoma, the whole tumor or suspicious area is removed. This procedure is called an excisional biopsy, and the extracted tumor tissue is then examined under a microscope by a pathologist to determine both which stage and grade the cancer is at [2].

When determining the correct treatment for the patient, several factors called biomarkers are taken into account. The cancer stage and grade are two of these biomarkers and play a major role. If wrong diagnosis is determined, it could lead to under- or over-treatment of the patient, as well as unnecessary suffering and cost [4].

2.1.1 TNM Stage

To determining the current stage of the cancer the classification of malignant tumors (TNM) system is used. The stage of the tumor is determined based on its size and whether it has invaded nearby tissue. Figure 2.1 shows the different stages a tumor may have.

The tumors may form papillary protrusions into the bladder lumen, solid nodules, or grow diffusely within the bladder wall. However, approximately 70 % of patients have non-muscle-invasive tumors (Ta or T1) [2].

When a surgeon removes the tumor, it is normal that some of the tissue close to the tumor is also extracted. Muscle, connective tissue and other are quite common to see in the histological images.

[4]Image by Cancer Research UK, used under Creative Commons BY-SA 4.0 license

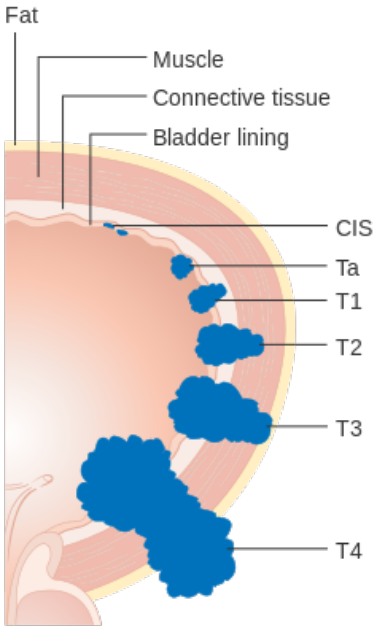


Figure 2.1: The urinary bladder with each of the T-stage tumors[4] [31].

2.1.2 WHO Grading

Another biomarker used to diagnose bladder cancer, is grading of the tumor according to the WHO grading system. There are two grading system which both are in use today, WHO73 and WHO04.

The grade of each system is based on the tissue architecture, nuclear arrangement, proliferation and nuclear atypia. Each of these categories has several subcategories to describe the tumor in detail. All of these subcategories are examined to determine the final grade of the tumor [2].

The WHO73 system classifies the tumors as grade 1, 2 or 3, while WHO04 classifies the tumors as PUNLMPs, low or high grade. There are some correlation between the two systems, but they are not directly interchangeable, so both systems coexist [2].

2.2 Neural networks

This chapter will introduce all the individual building blocks that a neural network consists of, and how to measure their performance.

2.2.1 Artificial vs. biological neurons

Artificial neural networks were developed to mimic the learning process of the human brain. The idea is to try and understand how a single biological neuron works mathematically, and then group them together in a large interconnected network similar to the biological networks in the brain [32].

W. McCulloch and W. Pitts were the first to introduce such an analogy between the biological neurons and a logical gate. This idea was further developed by Frank Rosenblatt who published the first concept of the perceptron learning rule. Artificial neurons are often referred to as perceptrons.

Biological neuron
The human brain consists of a large interconnected network of biological neurons. A neuron has multiple inputs called dendrites, and one primary output called an axon. Each neuron receives electrical input signals from several other neurons through its dendrites. The neuron controls the contribution of each input and accumulates them in the cell body, and if the resulting signal exceeds some threshold the neuron fires. When a neuron fires, a signal is sent through its axon to its boutons. The boutons are connected to thousands of other neurons using connections called synapses [33, 32].

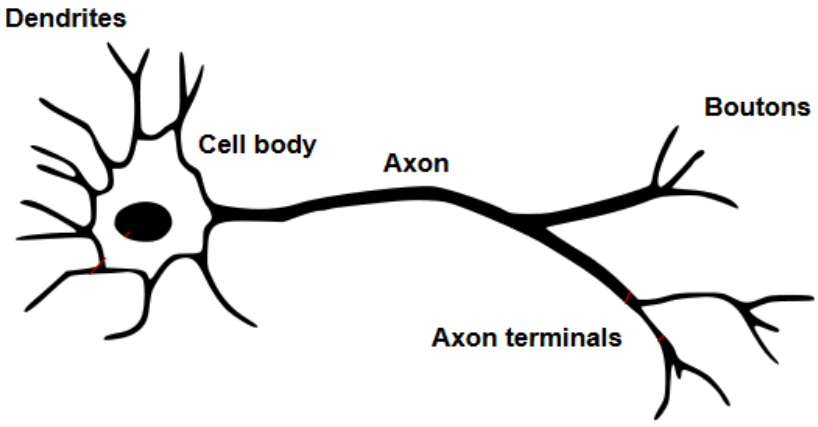


Figure 2.2: The biological neuron is the inspiration for the artificial neuron[5] [34].

[5]Image by Notjim and Looxix, used under Creative Commons BY-SA 4.0 license, text have been altered from original.

Artificial neuron
The construction of the artificial neuron is quite similar to its biological counterpart. It consists of several weighted inputs and a primary output according to Figure 2.3. The inputs are summed together and fed through an activation function which has a threshold to determine if the output should be weak or strong. The weights are then adjusted to minimize the error, which effectively emulates the strengthening and weakening of the synaptic connections found in the brain [32].

Frank Rosenblatt's idea of the perceptron was to create an algorithm that would learn the weights for the input signals in order to draw linear decision boundaries. The original perceptron used a step function as the activation function, which only allowed it to produce linear decision boundaries. As we will see later in Chapter 2.2.5 non-linear activation functions will be introduced which makes it possible to produce non-linear decision boundaries.

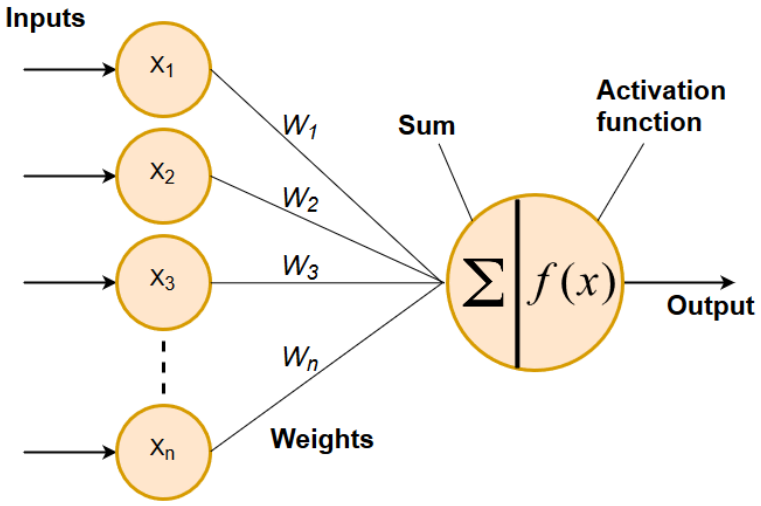


Figure 2.3: The artificial neuron, also called a perceptron.

2.2.2 Convolutional layers

Convolutional neural networks are a particular kind of neural network for processing multidimensional data. Images are multidimensional arrays consisting of height, width and depth. Height and width described the image size, and depth describes the color information. The images used in this thesis are RGB images consisting of three color channels, red, green and blue. Each color channel corresponds to one depth layer, meaning an RGB image has three depth layers.

In conventional neural networks, the input is a vector. If an image should be used as input for such a network, the image array has to be reshaped into a vector. The disadvantage of this is that the reshape operation would remove the spatial structure of input data. Instead, convolutional neural network utilizes the spatial correlation in the image and uses a small filter kernel which slides over the input

image. Example of this sliding operation is shown in Figure 2.4.

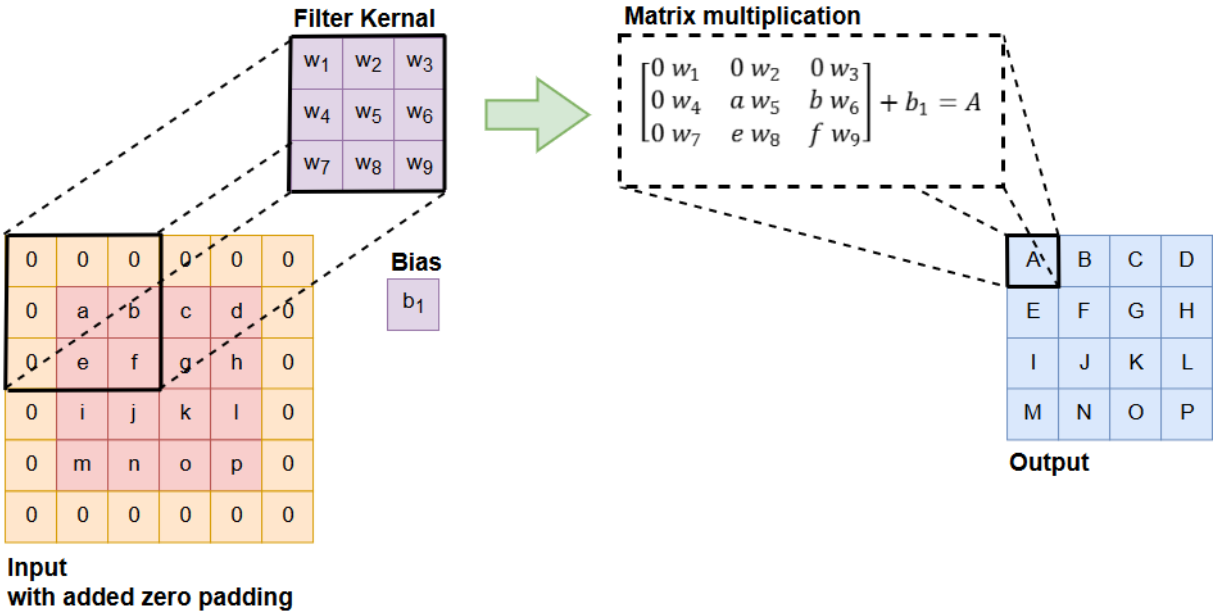


Figure 2.4: Example of convolution on an image. 3x3 filter kernel using stride of 1x1 applied to an 4x4 input image padded with a 1x1 boarder of zeros.

Biological

Continuing from the previous chapter regarding the biological analogy, convolutional neural networks are inspired by the animal visual perception, and thus can be applied to visual recognition tasks.

Neurophysiologists D. Hubel and T. Wiesel worked together for many years in the 1950s and 1960s to figure out the mystery of the animal visual cortex. Three of their published papers [35, 36, 37] studied the visual perception of cats and monkeys. They observed how neurons in the brain responded to images projected in precise locations on a screen [11].

They found that the part of the brain which process visual information called the visual cortex, contained neurons that individually responded in specific regions of the visual field known as the receptive field. These neurons responded only to the presence of edges of a certain orientation, e.g. horizontal, vertical or diagonal edges [11].

Their 1968 paper identified two basic visual cell types in the brain referred to as 'simple cells' and 'complex cells'. These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in images [11].

Their accomplishments were eventually recognized with a Nobel prize in 1981 "*for their discoveries concerning information processing in the visual system*" [38].

Convolution operator
When applying convolutional neural networks, a multidimensional discrete convolution operator is applied to the input and filter kernel. It is necessary to use discrete convolution because both the input image and filter kernel are discrete.

The images are 3-dimensional with height, width and depth. But the convolution operation is only applied to one depth channel at the time, which results is a 2-dimensional convolution. A distinct kernel is used for each depth channel. A 2-dimensional convolution applied to an image I with a filter kernel K is defined as

$$s(i,j) = I * K = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{1}$$

The output of an convolutional operation results in the same dimension as the inputs, e.g. 2-dimensional in this case. The convolutional operation is typically denoted with an asterisk: $s(i,j) = K * I$ [11].

The convolution operation essentially calculates the dot product between the filter kernel and parts of the input image. In machine learning libraries, like Tensorflow, usually takes advantages of this and implement the calculations as matrix multiplication [11].

After the filter kernel has convoluted across the entire image and the bias have been added and put through the activation function, the final output produces what is called a feature map. One feature map will be produced for each filter and stacked together along the depth dimension to produce a volume [11, 39].

Each filter kernel consists of weights which are initialized randomly, and then updated through the learning process. All weights used in each layer are generated randomly from a normal distribution with mean zero and standard deviation 1, except values which are more than two standard deviations from the mean are dropped and re-picked. The biases are all initially set as zero [40].

Parameters
There are several parameters that needs to be determined for the convolutional layer. The filter kernel size, number of filters, zero padding and stride all needs to be set.

Kernel size is usually square, with typical size of 3x3 or 5x5, but other sizes are also used in advanced networks. The number of filters determines how many feature maps is created. Zero padding can be put around the border. Stride is a measure of how much the kernel is translated in each step across the image [39].

To calculate the output size of the convolution layer all of these parameters needs to be taken into account. Output size can be calculated as follows:

$$Output\ size = \frac{Input - filter + 2 \times padding}{stride} + 1 \tag{2}$$

As an example, consider an input image with size 128x128x3 convoluted with a 5x5 filter kernel with a stride of 2 and no zero padding. The output would become:

$$\frac{128 - 5 + 2 \times 0}{2} + 1 = 62.5 \tag{3}$$

This output size is not an integer, and therefore not a valid size. Implementing this solution would result in an error. So either the input size or the parameters will have to change until a valid result is achieved. Changing the stride to 1 would result in:

$$\frac{128 - 5 + 2 \times 0}{1} + 1 = 124 \tag{4}$$

Which is a valid output size. This example has illustrated two things. Finding the right set of parameters can sometimes be challenging, specially if a certain output size is needed. And secondly, a minor change in one parameter may have a large impact on the resulting size.

Deconvolution
Deconvolution is the reverse operation of convolution, and is used to reverse or undo the effect of a previous convolution operation [41, 42].

There are several mathemathically techniques described to do this, e.g. Richardson–Lucy deconvolution method. However, Tensorflow which is the machine learning library used in the experiments, they state that " *This operation is sometimes called "deconvolution" after Deconvolutional Networks [41], but is actually the transpose (gradient) of 2-dimensional convolution rather than an actual deconvolution.*" [43, p.1].

2.2.3 Pooling layers

Pooling layers are used for down-sampling the images. This is done to both reduce the amount of parameters and to prevent overfitting. The pooling layers are of particular importance in an autoencoder where reducing the size of the input is essential.

Pooling layers have two parameters to control their behavior, filter size and stride. Filter size is the size of the kernel, and the stride is how far the filter kernel is moved across the input. The most common setting for these is to use a filter of size 2x2 with a stride of two.

There are several different kinds of pooling layers used. There are average pooling, L2-norm pooling or a weighted average based on the distance from the central pixel, but the most common are max pooling. Max pooling keeps the maximum value within a neighborhood and discards the rest. Using a filter size of 2x2, 75 % of the input are discarded. An example of this is shown in Figure 2.5. The reduction in size results in efficiency for the network as well as reduced memory requirements for storing the parameters [11].

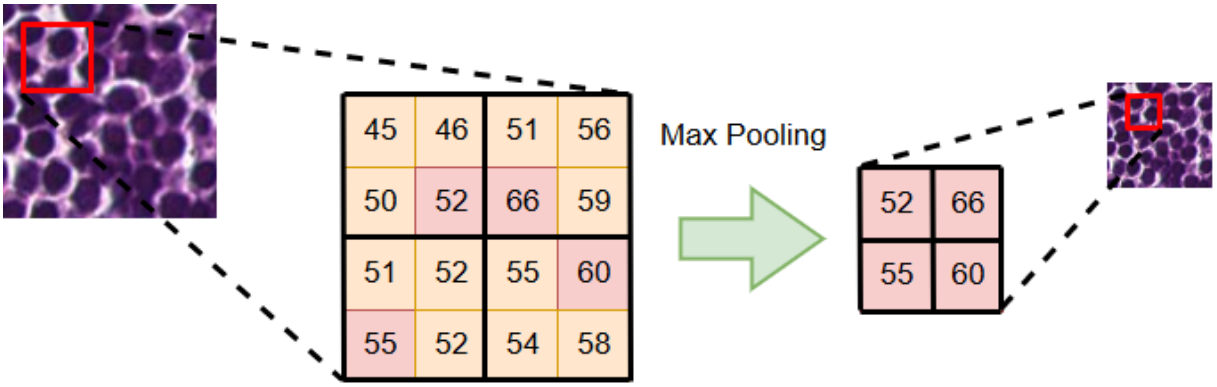


Figure 2.5: Example of max pooling with a 2x2 filter and stride of 2

Unpooling
Some networks structures, like the decoder part of the autoencoder, needs to add information to the input instead of discarding it. As the pooling function is not invertible [11], there is no such thing as an un-pooling function available in Tensorflow (the machine learning library used to program the neural networks).

2.2.4 Fully-connected layers

A fully-connected layer is like a conventional neural network. Each input node is connected to each output. Figure 2.6 illustrates a simple fully-connected network.

Fully-connected layers are none-spatial functions, meaning they do not take a local neighbourhood into account like a convolutional operation. Because of this, fully-connected layers have to be added after all the convolutional layers to not destroy the semantic information in the image before convolution. Fully-Connected layers are therefore always located in the last layers of a deep neural network [39].

Several fully-connected layers can be stacked after one another. Each layer can have different size, meaning that fully-connected layers can be used to both compress and expand the data.

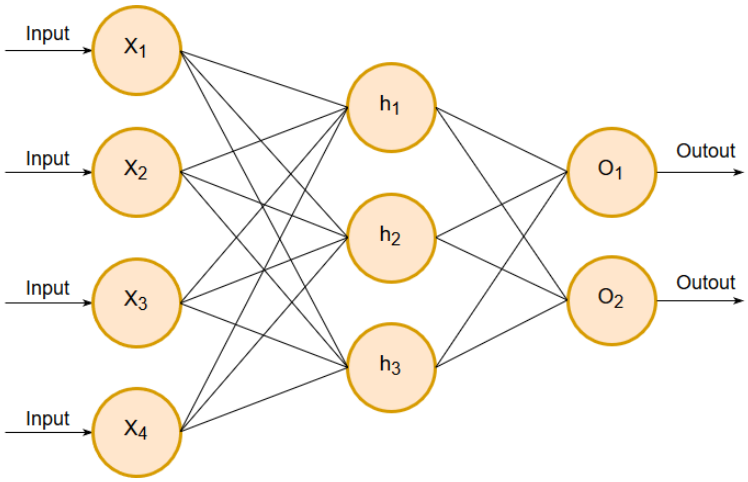


Figure 2.6: Example of a fully-connected neural network with four input nodes, one hidden layer and two output nodes. Each connection have a weight associated with it.

2.2.5 Activation function

The last part of an individual node is called the activation function. All the inputs are summed together and then put into the activation function to determine the output of the node. If the sum of the input were sent directly to the output, it would be a linear activation function. This was the main problem of Frank Rosenblatt's perceptron back in 1959. Minsky et al. proved that it was theoretically impossible for it to learn the XOR function, which is non-linear.

This makes non-linearity an important property of the activation function. Another desirable property is for the function to be continuously differentiable to be able to use its gradient based optimization methods. Together with some other necessary properties, a list of activation functions has unfolded over the years. A common connection between them is that they are all inspired by the biological workings of neurons in the brain, in addition to possessing some different mathematical properties to make them mathematical convenient to use in neural networks [11, 44].

ReLU

In modern neural networks, and especially convolutional networks, the default recommendation is to use the rectified linear unit called ReLU [11, 45]. The ReLU function adds non-linearity to the equation and allows the network to compute non-trivial problems [11]. The ReLU activation function is given as:

$$f(x) = max(0, x) \tag{5}$$

Softmax
Softmax is another useful activation function most often used as the output of a multiclass classifier. The softmax function makes sure that each element of the output lays between 0 and 1, and the entire vector sums to 1. These properties makes the output represents a valid probability distribution. The following formula gives the softmax function:

$$Softmax(x)_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \quad for\ i = 1, .., K \tag{6}$$

2.2.6 Neural network Learning

In the context of neural networks, learning refers to the process of updating a set of parameters. The parameters determine the output of the system, which is used to calculate the error. The parameters are then updated in such a way that it reduces this error. The learning process is an iterative process performed multiple times until convergence. The most common learning technique is stochastic gradient descent algorithm which calculates the gradient and uses this to determine how to update the parameters. Learning is divided into three subcategories; supervised, unsupervised and reinforcement learning, where the two former play a major role in this thesis.

In supervised learning the dataset is labeled, meaning that each sample that goes into the system has a label or integer assign to itself referring to which category it belongs. This label is used as the target for the system, and the error between the system output and target is computed. This error is then used to tell the system how to update its weights [46, 11]. In this thesis, supervised learning is used to train the classifier.

The largest drawback of supervised learning is the process of labeling the dataset, which is very time-consuming. Gathering hundred thousands of images takes a lot of time, but going through every single image and assigning each of them to a category is unbearable.

In unsupervised learning, the input samples don't have any labels assigned to them. The system only has the input to work with and its features [46]. Unsupervised learning is used to train the autoencoder in this thesis, which is the majority of this project.

In May 2015, an article by Y. LeCun, Y. Bengio and G. Hinton wrote a review of deep learning. In their conclusion they mentioned "*..we expect unsupervised learning to become far more important in the longer term. Human and animal learning is largely unsupervised: we discover the structure of the world by observing it, not by being told the name of every object*" [47, p.7].

The dataset used in this thesis is labeled according to their cancer grade, stage and

if the patients turn out to have recurrence and potential progression at a later stage in time. However, in this thesis the classifier will choose between images based on their texture, which is not labeled. Therefore the dataset is considered unlabeled, with only a proportion of it labeled. This means that majority of the thesis consist of unsupervised learning, with some supervised learning to fine-tune the classifier.

2.2.7 Autoencoder

An autoencoder is a neural network with a special structure. It receives an image as an input, compresses it, and then reconstructs it. An autoencoder consists of two main parts; the encoder and the decoder. The encoder part will transform the input image into a latent vector. A latent vector is one which is not directly observable, meaning it can not instantly be reconstructed into an image. To reconstruct it, the decoder part is needed. The latent vector is a representation of the input image, but of a much lower dimensional space. The main idea of an autoencoder is for it to extract the most important features of the image, and preserve these in the latent vector [11].

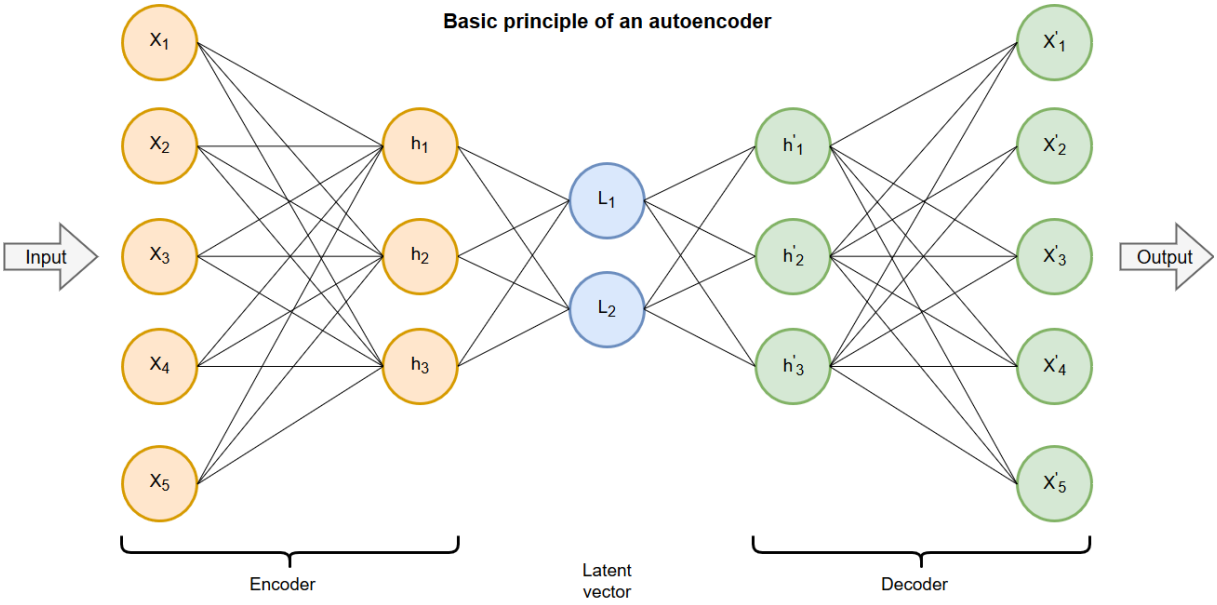


Figure 2.7: Basic principle of an autoencoder

An autoencoder which reconstructs the image almost perfectly can easily be constructed by setting the latent vector large enough. However, the network will learn all of the images features, and the latent vector space will not be of a low dimensional space relative to the input image. Such a latent vector would perform poorly on a classification task afterward.

Instead, a small latent space is chosen. This will force the network to compress the input image during training and learn to keep the most important features. One

way to reduce the size of a representation is to find and remove redundancies. Identifying and removing more redundancy enables the dimensional reduction algorithm to achieve more compression while discarding less information [11].

Figure 2.8 is a visualization of the latent vector, and how it relates to the encoder, decoder and classifier. The latent vector contains all the features from the image and is used as input to the classifier.

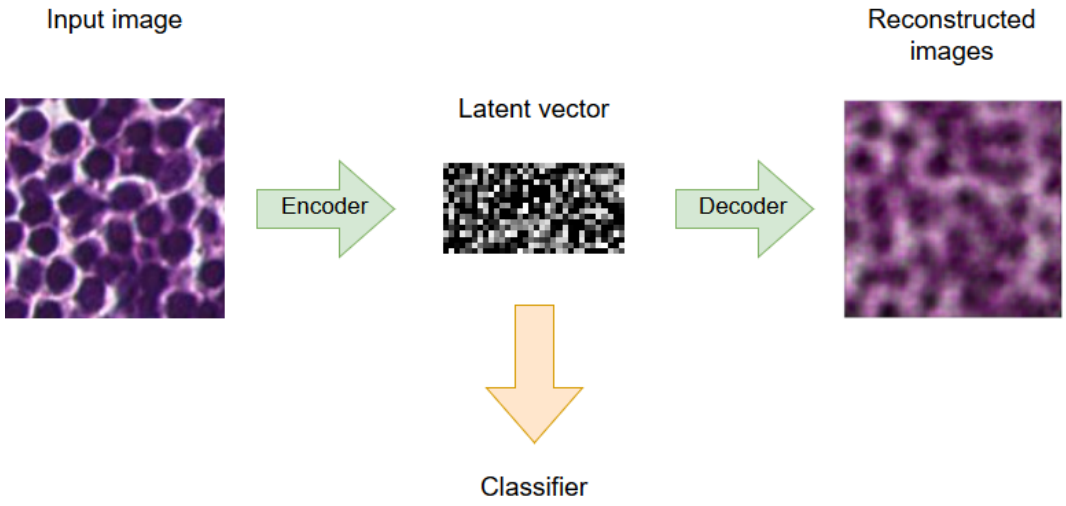


Figure 2.8: The input image is compressed by the encoder into a latent vector. The decoder will then reconstruct the image using the latent vector. The latent vector is also used as the input to the classifier. This is the actual latent vector to the image shown, but have been scaled up for more convenient visualization.

During training, the network looks at the squared difference between the input image and the reconstructed output image as given by the loss function:

$$Loss = \sum (input - output)^2 \tag{7}$$

After training, the encoder have learnt to extract the most important features of the input images. These features are now stored in the latent vector. To do classification, the structure of the network is altered. The decoder part is discarded and exchanged with a classifier. The classifier usually consists of several fully-connected layers connected to the output of the encoder. These layers need to be trained as well to be able to classify input images.

2.2.8 Classifier

In short terms, a classifier is a function that takes an unlabeled input and maps it to an labeled instance. The input to the classifier is the feature vector provided by

the feature extractor. The feature extractor in this case being the autoencoder. The classifiers task is to assign the input object to a particular class or category [46].

Because perfect classification performance is often impossible, it is often more reasonable to determine the probability for each of the possible categories [46]. This is why the softmax activation function is used on the output of the classifier. The classifier observes several random objects $\mathbf{x}$, which has assigned a label $\mathbf{y}$. The classifier then learns to predict $\mathbf{y}$ from $\mathbf{x}$ by estimating $p(\mathbf{y}|\mathbf{x})$ [11].

Examples of a binary classifier is a system used to determine if an incoming email is 'mail' or 'spam'. There are also multiclass classifiers, which have more than two categories to chose between. An example of this is a system to determine the blood type ('A', 'B', 'AB' or 'O').

2.2.9 Cross-validation

When a model is trained multiple times on a dataset, the model is optimized to fit that data. When new data is introduced to the model, the performance may be poor. In such a case the model may be overfitted, meaning that the model fits the training data well, but does not fit the validation data. This is particularly likely to happen when the training dataset is small, or when the model consists of a vast number of parameters.

Validation is used to estimate how well the model generalizes to new independent data. To compute this estimate, the dataset has to be partitioned into subsets. Conventional validation would be to split the dataset into two subsets, one training set (e.g. 70 %) and validation set (30 %). The model would then be trained on the training set and evaluated on the validation set to check the model's performance.

One of several drawbacks of this method is that the model is not trained on the validation data, and may not learn patterns or features that only appear there. Also with small datasets, there may not be enough data to be able to split the dataset without losing significant modeling or testing capability.

A better technique is to use K-fold cross-validation. This method will randomly shuffle the data and divide the dataset into K separate subsets of approximately equal size. One subset will be chosen as validation data and the other K-1 subsets as training data. The model is trained on the training data and evaluated on the validation data. After training and evaluation the process starts over again, but now with a new training and validation set. This process is repeated K times until all elements in the dataset have been part of the validation data once as shown in Figure 2.9.

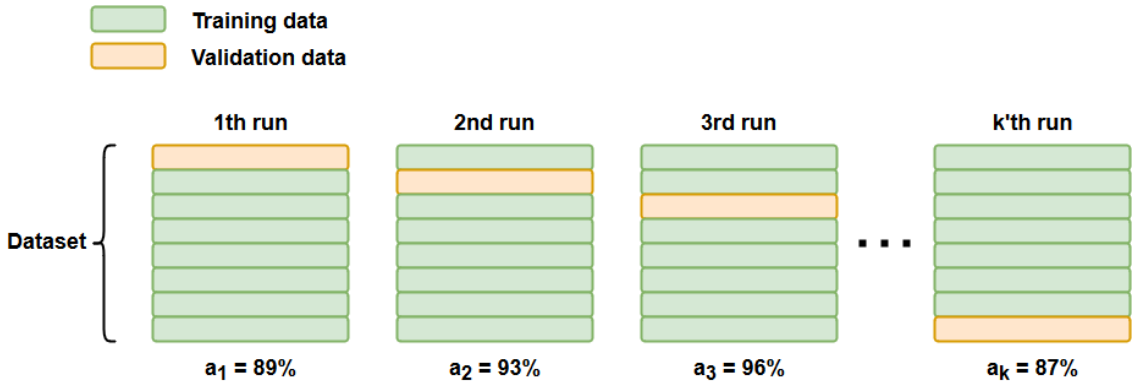


Figure 2.9: Example of how to split the dataset between training and validation set when using k-fold cross validation. The accuracy results are arbitrary numbers for illustration only.

After each individual run the accuracy is stored. These are shown in Figure 2.9 as a_1, a_2 etc. After all K runs are finished, the accuracy is estimated as the average of all K runs according to Formula 8. In addition, it is common practice to accommodate the accuracy with a standard deviation based on the results from each run.

$$Accuracy_{CV} = \frac{1}{K}\sum_{j=1}^{K} a_j \tag{8}$$

The main advantage of K-fold cross-validation is that all data is used both as training and validation. The downside however, is that it requires K times longer to calculate the estimated accuracy. When training deep neural networks which may require many hours or even days to finish, multiplying this time by e.g. ten times is notable to say at least.

The paper "*A study of cross-validation and bootstrap for accuracy estimation and model selection*" [48] compares several validation techniques on large scale real-world dataset. For estimating the accuracy of a classifier, an estimation method with low bias and low variance is preferable. Results in the paper showed that k-fold cross-validation was pessimistically biased for low values of K. "*Most of the estimates are reasonably good at 10 folds and at 20 folds they are almost unbiased*" [48, p.5]. It further states that "*there is almost no change in the variance of the cross-validation estimate when the number of folds is varied*" [48, p.3].

They then conclude with the following statement "*Our results indicate that for real-world datasets similar to ours, the best method to use for model selection is 10-fold stratified cross-validation, even if computation power allows using more folds.*" [48, p.1].

Stratified cross-validation means that the folds are stratified so that they contain approximately the same proportions of labels as the original dataset. This has not

been tried to accomplish in this thesis, but the recommendation to use 10-folds is followed.

2.2.10 Confusion matrix

To easily visualize the result and performance of a classifier, a confusion matrix is often used. This uses a specific table layout to present the data from the classifier. It is useful whenever supervised learning is used, as the true label of each class is needed. The true classes are located along the rows of the table, and the predicted classes along the columns as showed in Figure 2.10.

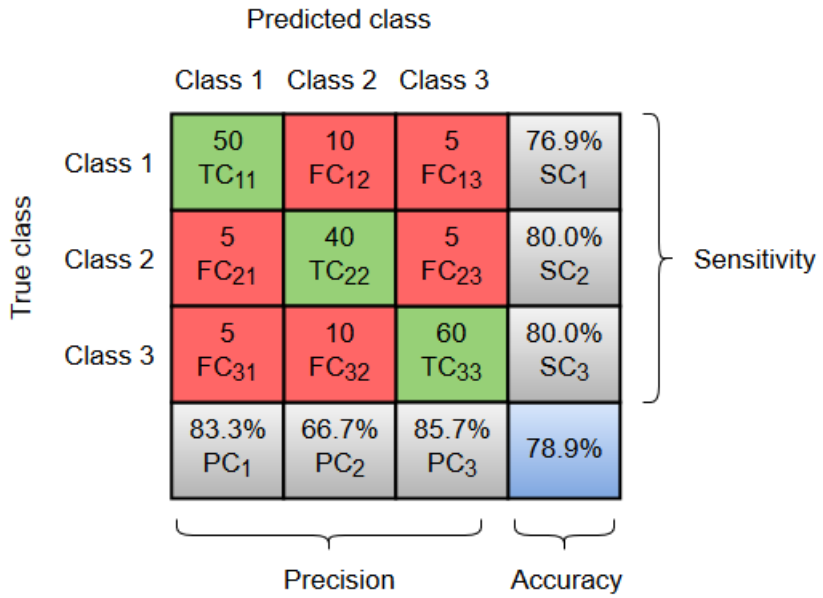


Figure 2.10: Confusion matrix example. TC = True Class. FC = False Class. SC = Sensitivity Class. PC = Precision Class.

The green fields along the diagonal indicates the number of correctly identified items of each class. Whereas the red fields indicate wrong classification, and also shows what class it is wrongly classified as. The gray and blue fields are sensitivity, precision and accuracy which all help measure the performance of the classifier. The two former values are calculated for each class to give a more detailed analysis of the individual class performance, as accuracy alone can sometimes give a misleading result, especially if the datasets are unbalanced [46].

An easy example of this is as follows, given a two-class problem with 95 items of class 1 and five items of class 2. If the classifier is biased and classifies all items as class 1 the accuracy becomes 95 % which sounds good, but this does not reflect the 0 % classification for class 2. A confusion matrix makes it easy to see if an algorithm confuses two or more classes, meaning commonly mislabeling one class as another, hence the name *confusion matrix* [46].

Sensitivity
Sensitivity is a measure of the proportion of a given class that is correctly predicted as such. The percentage in the gray box is how large proportion of e.g. class 1 that is predicted as class 1. Sensitivity is also known as True Positive Rate (TPR) or Recall, and is calculated according to Formula 9 [46].

$$Sensitivity\ class\ 1\ (SC_1) = \frac{TC_{11}}{TC_{11} + FC_{12} + FC_{13}} \tag{9}$$

Precision
Precision is a measure of how the proportion of predicted classes is correctly identified within one class. Precision looks at all values that the classifier has classified as e.g. class 1, and then calculates how large proportion of these that are correctly predicted. Calculation of precision is shown in Formula 10 [46].

$$Precision\ class\ 1\ (PC_1) = \frac{TC_{11}}{TC_{11} + FC_{21} + FC_{31}} \tag{10}$$

Accuracy
Accuracy is the proportion of total number of correctly predicted classes, and is an overall measure of how well the classifier is. Accuracy can also be used as the probability of correctly classifying a randomly selected instance. Accuracy is calculated as shown in Formula 11 [46].

$$Accuracy = \frac{TC_{11} + TC_{22} + TC_{33}}{Total\ population} \tag{11}$$

2.2.11 Tensorflow

Tensorflow is an machine learning library developed by Google. It were realesed as an open-source package under the Apache 2.0 license in November, 2015. Operations of neural networks are done on multidimensional data arrays called tensors, hence the name Tensorflow. It is used both by researchers and production at Google. TensorFlow provides a Python API, as well as C++, Haskell, Java and Go APIs. In this thesis, the Python API has been used [40].

Figure 2.11: Tensorflow logo [40]

2.3 Material

This chapter gives an small overview of the material used in this thesis, the histological images. In addition, a review of the file format and the data augmentation scheme is presented.

2.3.1 Dataset

The dataset consist of histological images from about 360 patients taken between 2002 and 2010. Digitalization of the tissue samples were done at the Department of Pathology at Stavanger University Hospital using an SCN400 histological slide scanner from Leica. In addition to the images, metadata regarding cancer grade, stage, recurrence and disease progression are also available.

Tissue classes
During biopsy, several other parts surrounding the cancer tissue are also extracted. These parts are visible in the images, and creates small regions of the individual parts. The histological images may have regions with cancer cells grouped together, next to it a region with blood and another region with damages tissue. Table 1 lists the five main classes used in this thesis. Note that the images may contain other classes not listed here.

Table 1: An overview of the different classes within the dataset

Class	Name	Description
Class 1	Cancer tissue	Tissue consisting of cancer cells
Class 2	Other tissue	Other tissue like connective tissue, muscle tissue or similar.
Class 3	Damage tissue	Tissue that have been damage due to e.g. heat or physical
Class 4	Blood	Red blood cells
Class 5	Background	Tiles of background with small parts of debris, tissue or similar

Only class 1 is useful when trying to diagnose the cancer grade/stage. Class 2-5

may be regarded as noise, as they do not provide any information relating to the cancer grade/stage.

Figure 2.12 shows some example tiles of each class.

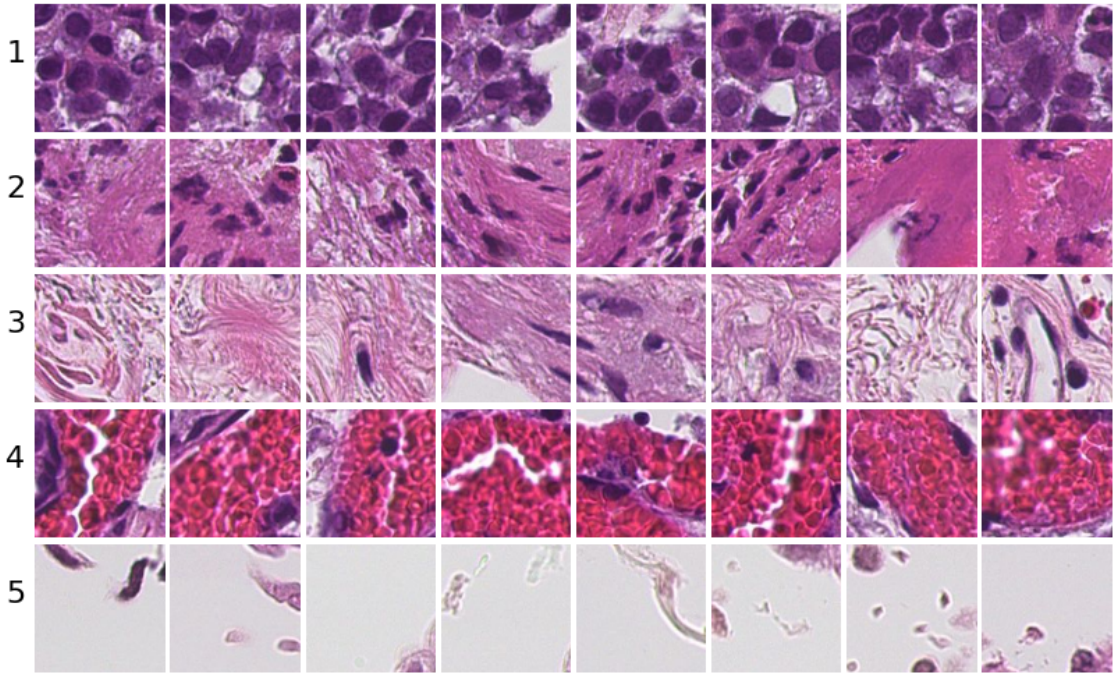


Figure 2.12: Whole-slide histological images consists of multiple classes each with different textures. 1) Cancer tissue. 2) Other tissue. 3) Damage tissue. 4) Blood. 5) Background.

2.3.2 SCN image format

After the tissue has been removed from the patient and placed on a microscope slide, it is scanned using a Leica SCN400 Slide Scanner. This scanner saves the image using Leica's own SCN image format. The SCN format is a single-file pyramidal tiled BigTIFF image. The bigTIFF format is the same as the tiff format, but uses 64-bit offset rather than 32-bit to be able to save larger files. The images being pyramidal tiled means they are deep zoom capable, meaning it is possible to view the slide at zero magnification, or zoom all the way in at 400x magnification [49].

To open and view an SCN image Leica's ImageScope SCN viewer, or another SCN viewer program, is needed. To be able to process them, OpenSlide is used. OpenSlide is a vendor-neutral software designed for digital pathology. It supports several medical image formats, including SCN. OpenSlide is released as an open-source software under the LGPL v2.1 license [50].

To be able to do image processing on the SCN images, another library named Vips

(VASARI Image Processing System) was chosen. Vips can not read the SCN images directly, but uses the OpenSlide library for this. Vips is also an open-source software released under the LGPL license [51, 52].

Vips was introduced by J. Cupitt and K. Martinez [51, 52], and is a result of several EU-funded projects (VASARI (1989-1992), MARC (1992-1995), ACOHIR, Viseum) whose primary objective was to build a system capable of measuring long-term color change in old master paintings. In 2005 the research and development effort of Vips was changed to medical images, and are currently being used for scientific analysis, general research and development [51].

Vips handle large images very memory efficient. Usually when doing image processing, the whole image is loaded into the computer memory. Due to the size of the SCN images, this is not possible. The Vips library only loads the part of the image that is currently being processed into memory. In addition to being memory efficient, it is also very fast. This is primarily due to its architecture which automatically parallelises the workflows. In a benchmark comparison Vips showed to be 5.6 times faster that Pillow (Python Imaging Library) and 6.7 times faster that OpenCV [53].

2.3.3 Preprocessing

Tile size
A histological image is far to large to be feed into the autoencoder, and has to be split into smaller tiles. If the size of the tiles is chosen to be small, the amount of weights necessary in the autoencoder is lower, which requires less memory to store. However, a smaller amount of information is present in the tile and it may not be possible to learn any features regarding the grade of the cancer type. If a larger tile size is chosen, more cancer cell is present in each tile, but larger memory space is required.

A similar study by Litjens et al. [54] using deep learning on histological images made the following statement regarding tile size: "*Patch size in pixels was determined empirically during initial experiments. We tried 64 x 64, 128 x 128 and 256 x 256 pixel patches. The 64 x 64 sized patches performed substantially worse on patch-based accuracy and 256 x 256 sized patches limited convolutional networks depth due to memory limitations of the GPU. As such, we settled on a patch size of 128 x 128.*" [54, p. 8].

Based on the conclusion of Litjens et al. [54], a tile size of 128 x 128 was chosen for the experimental work of this thesis.

Removing background
Whole-slide images consist of a lot of background. To reduce unnecessary computational time in future steps, most of the background is removed during preprocessing.

The background has a uniformly distributed gray color, and several advanced techniques exist to remove this with high precision. Due to the vast amount of data, the main issue is computational time rather than accuracy. In fact, it is important to let some of the background images through, or else the autoencoder would not be able to learn its features.

When a tile is extracted from the whole-slide image, the algorithm has to determine if the current tile consists of mostly background or tissue. To do this, the histogram of the tile is computed. Next, the threshold where 10 % of the histogram is located, is calculated. All of this can easily be calculated in Vips using the command *Vips.percent(10)*. The command can calculate the threshold for any percent value, but 10 % was found to produce the best result.

In Figure 2.13 this threshold is shown with the red arrow. For a tile containing tissue like 2.13a the histogram is evenly distributed across the specter and the 10 % threshold is usually somewhere in the middle. For the tile in 2.13b which includes mostly background, the histogram is heavily shifted towards the right end of the specter which will also move the threshold in the same direction.

If the threshold lands within the region marked with the black arrow, the tile is considered as background and therefore not saved.

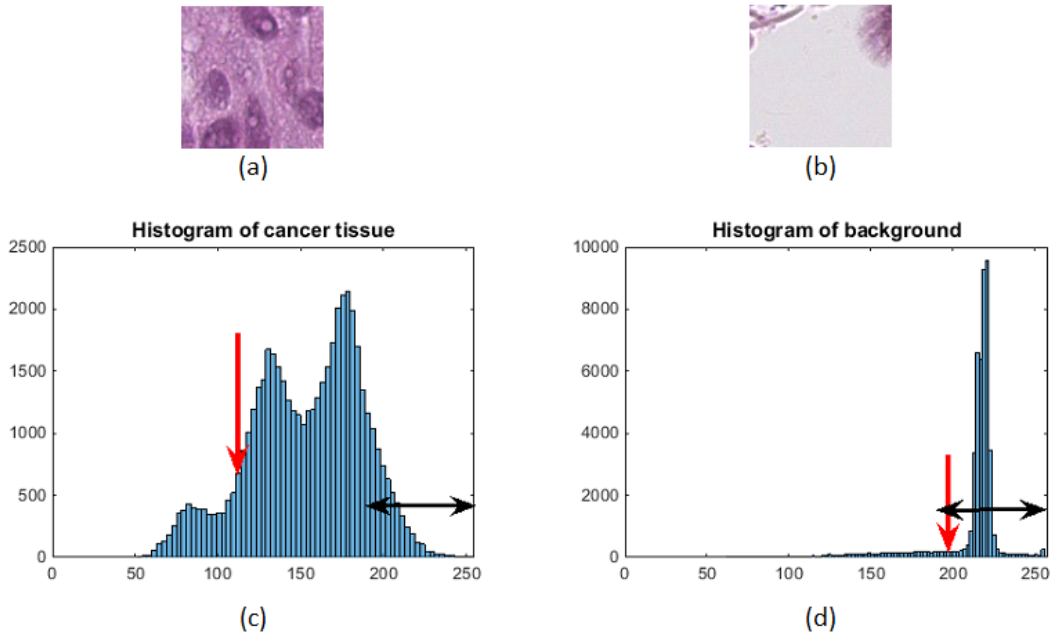


Figure 2.13: Comparison of histogram from two different tiles. Red arrow indicates where the 10 % threshold is. The black arrow indicates the interval which, if the red arrow falls within, that tile will be considered as background and therefor discarded.

A similar setup is used to determine if the current tile consists of mostly black,

which indicated that a binary mask has been applied to that part of the image. Since black has a pixel value of 0, most of the histogram is located to the far left of the plot, and the 10 % threshold will be located here as well, and these tiles are easily filtered out.

2.3.4 Data augmentation

Data augmentation is useful when the available dataset is too small. Common augmentation techniques are to either systematically or randomly rotate and flip the images. This has two advantages. First, more training data is produced. Secondly, the system is trained to become rotational invariant because the texture in the images can be found at any angle.

The semantic information of a histological image is not altered by flipping of rotating the image. As seen in Figure 2.14 some combinations of rotation and flipping produce the same results and are therefore excluded. This augmentation scheme results in an 8x increase in data.

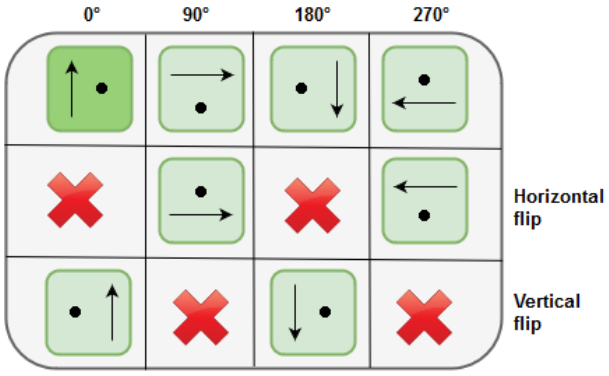


Figure 2.14: Augmentation scheme. The dark-green square marks the input image. This image is then rotated and flipped to produce more data. Some combinations of rotation/flipping produces redundant images and are excluded.

3 Method

This chapter explains how the individual building blocks from the previous chapter are put together to produce the system. First the proposed system is presented, which consists of six steps. Next the preprocessing, autoencoder and classifier will be explained in details.

3.1 Proposed system overview

The proposed system consists of six parts that have to be executed separately. Each step in the process builds on the previous step, so the order is also important. To best illustrate the system, several figures are used to present the system. As an example, the figures shows how the system can be used to predict cancer stage, but the system is also capable of predicting cancer grade, recurrence or disease progression. How the system performs in each case needs to be evaluated by experiments, but is not a part of this thesis.

Step 1-4 is the basis for this thesis. Step 5 and 6 are not part of this master thesis work, but can be considered as relevant future work. They are still included so the reader gets the full picture of the system.

Step 1 - Preprocess images
The first step of the system is to preprocess the input image. Due to its large size, it is not possible to feed the image directly into the autoencoder. The preprocessing algorithm turns the large input image into smaller tiles of size 128x128 px. It also checks each tile and removes them if they consist of mostly background. The cancer grade for the input image is also stored for later references.

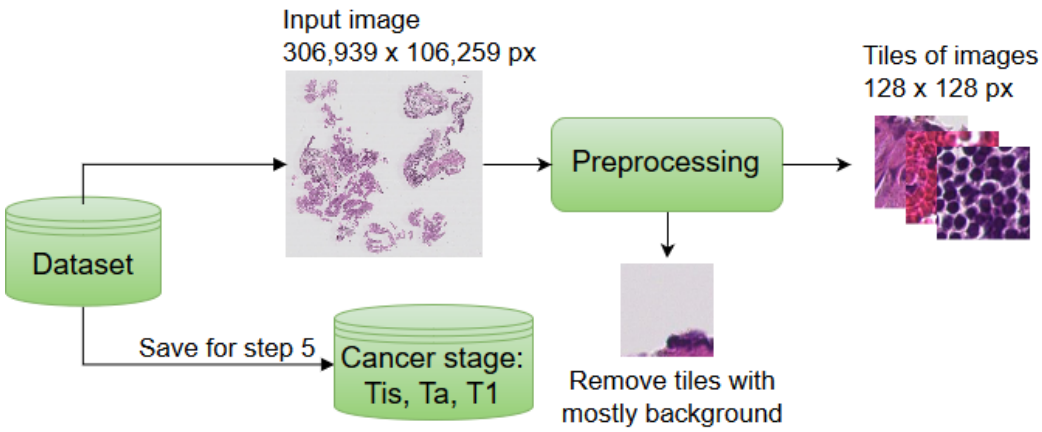


Figure 3.1: Preprocess input image. Note that the input image is shown here without the white boarder, but the pixel size is with the boarder.

Step 2 - Train autoencoder
The output tiles from step 1 is now used as input images to the autoencoder. Each image will go through both the encoder and decoder. The output of the autoencoder is the reconstructed image of the original input. These two images are then compared to each other using the loss function described by equation 7. The autoencoder will train itself to be able to reconstruct the images. Note that the reconstructed images in Figure 3.2 are slightly blurred. This is because the autoencoder acts as a lossy compression algorithm and some of the information is lost. Since the input images don't have any label, this is called unsupervised learning.

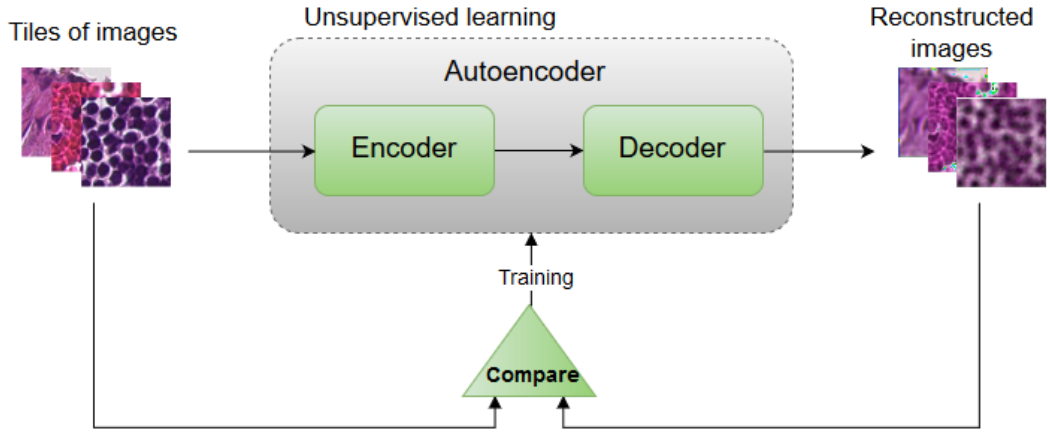


Figure 3.2: Train autoencoder

Step 3 - Train classifier
After the autoencoder have been trained to reconstruct the input images, the next step is to train the classifier. In this step the classifier has to know what each input image is, to be able to learn to recognize the different images. The input dataset now consist of several hundred-thousand of tiles, and it is not possible to label them all, so only a small subsample dataset is created with a label for each image. This small labeled dataset is then used to train the classifier. Each input image is fed through the encoder part of the autoencoder, and the output of the encoder is then fed into the classifier. The classifier will give a prediction whether to which class it thinks the current image belongs to. This prediction is compared to the true class of the image and the classifier will update its weights accordingly.

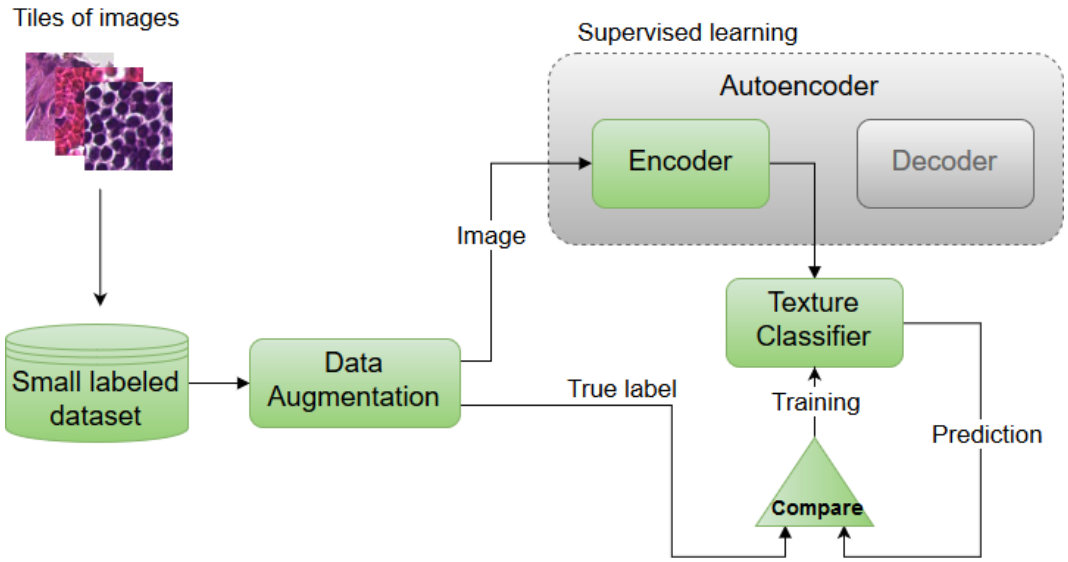


Figure 3.3: Train classifier

Step 4 - Categorize tiles using texture classifier
After the autoencoder has been trained and the classifier has been fine-tuned, the next step is to categorize each input image based on its texture. Each image is fed trough the system, and the classifier will predict which class the current image belongs. Only images which are classified as cancer tissue are saved, all other images are discarded.

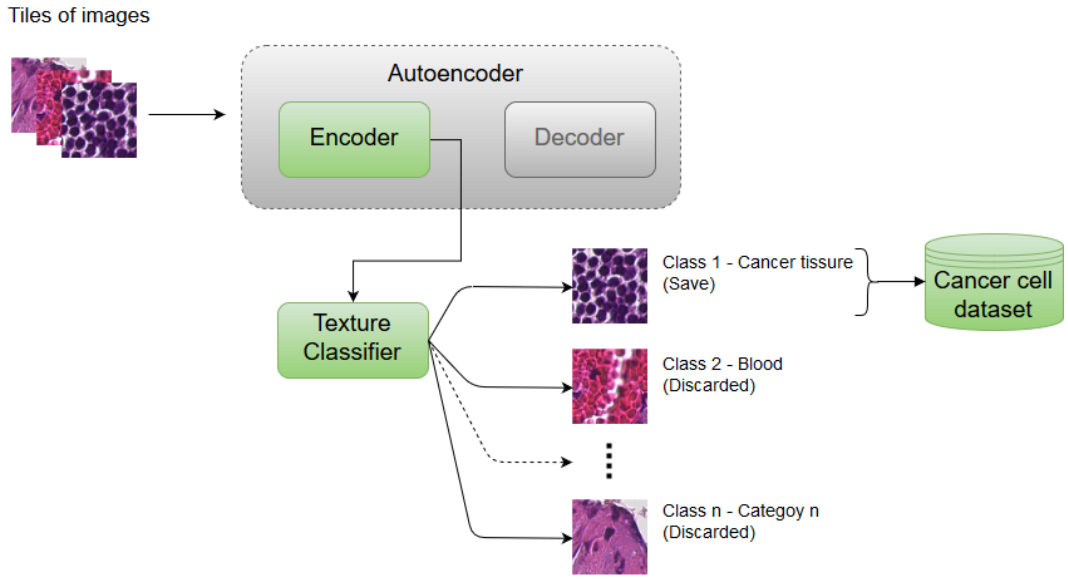


Figure 3.4: Sort tiles using texture classifier

Step 5 - Train convolution neural network
This step is not part of this master thesis work, but can be considered as relevant future work. The cancer type label collected in step 1 and the images classified as cancer tissue in step 4 can be combined and used to train a convolution neural network. The images are first augmented, meaning they are both rotated and flipped to produce more data out of the dataset. This process also makes sure the network becomes rotational invariant.

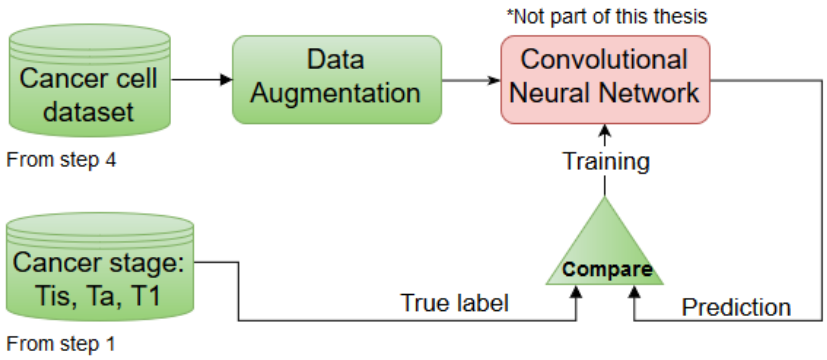


Figure 3.5: Train convolution neural network

Step 6 - Use system on new histological images
This step is not part of this master thesis work, but can be considered as relevant future work. The last step can be used to predict cancer type on new images. Whenever a biopsy is taken from a new patient, the tumor is sliced and scanned to produce the whole-slide image. This image can be fed through the system suggested in Figure 3.6. The system will then provide a prediction of the cancer grade.

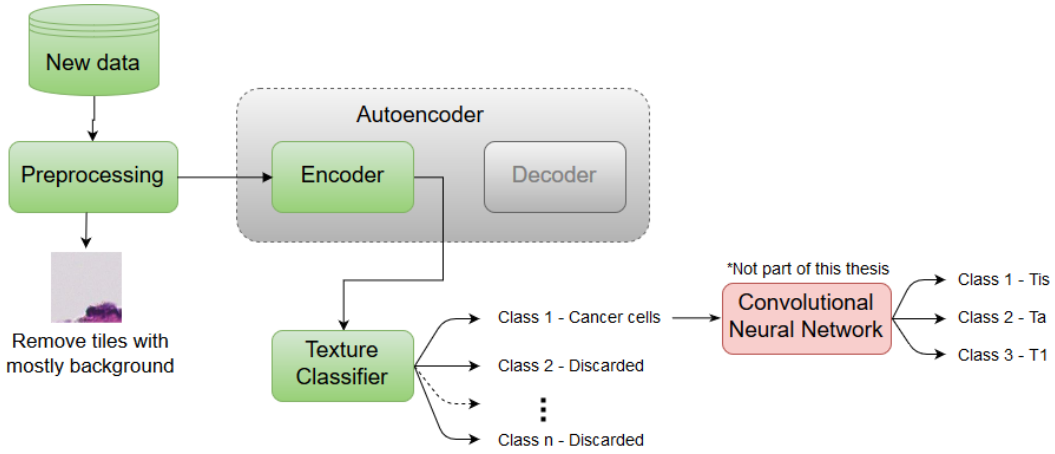


Figure 3.6: Use system on new histological images

3.2 Preprocessing

A histological image is far too large to be feed directly into the autoencoder, and needs to be divided into several smaller images. To do this an automated program has been developed. An overview of the preprocessing program is shown in Figure 3.7.

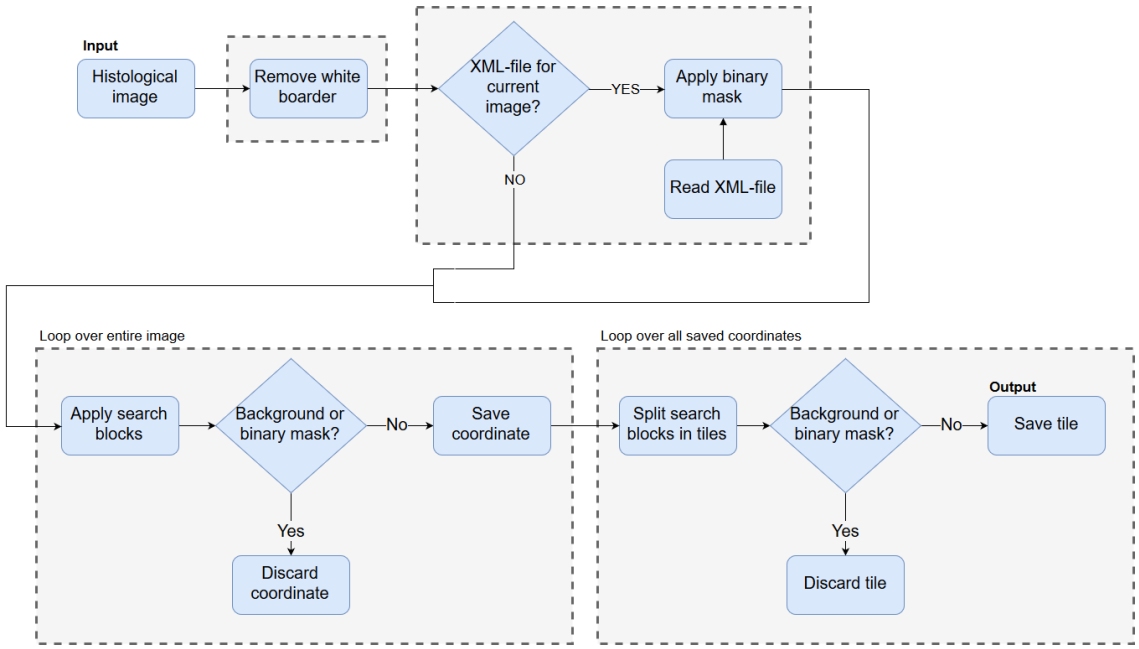


Figure 3.7: Overview of the preprocessing program

Around the histological images there is a large white boarder which contains no information. This is not visible if the image is opened in an SCN-viewer program, but has to be taken into account when working with the images in Python. This white border exists because the scanning area is larger than the microscope slide (its the same thing as when a receipt is scanned using a traditional scanner which is set to scan in A4 format, a white border will appear around the receipt). A binary search algorithm has been implemented which searched from the edge towards the center of the image of each of the four sides. This algorithm searches for the transition between the white border and the actual histological image. When this border has been found the image is cropped to contain only the histological image.

It is possible to mark unwanted areas of the image. This is done using the ImageScope SCN viewer from Leica. This program contains tools to draw freehand polygons in the image. Coordinates of these polygons are stored in an XML-file together with the image. The program reads the XML-file and creates a binary mask with the same size as the histological image and consists of only 1's. The polygons in the XML-file is then transferred to the binary mask and given pixel values of 0. The binary mask is then multiplied element-wise with the image which will set the marked areas as black polygons in the image. These are then removed later in

the program. There is an option in the program to inverse this function, which will mark everything outside the polygon instead of inside. This makes it possible to mark out regions of interest in the image. This function was useful when making the labeled dataset, as will be explained later in Chapter 4.5.

When a histological image is split into tiles of size 128 x 128, one image can consist of as much as 2,887,680 different tiles (using image H3395 as an example). Each of these tiles needs to be both cropped out, processed and saved which all adds up and takes a lot of time. And since the majority of the histological image consist of only background, a search block is first used to filter out large areas of background. The search block has a much larger size (1024x1024) than the individual tiles, and therefore can cover the whole image in less time. The search block loops through each row and column of the image in a systematically fashion and checks each block. If the current search block position consists of only background, that position is never processed again. If any tissue is present, the coordinated are saved and the tiles will check it more closely afterward.

Next, all the saved search blocks are divided into tiles and checked. Tiles that consists of background will be discarded, while all other tiles will be saved. The tiles are saved as JPEG images which is a lossy compression format, where the compression rate is controlled by the Q-factor. Q-factor is a value chosen between 0-100 which determines how much compression to apply to an image. A low Q-factor will compress the image a lot so it takes up less storage space, but will remove most of the details in the image. To preserve as much of the raw pixel values as possible the Q-factor is therefore set to 100.

3.3 Autoencoder

One of the main parts of the system is the autoencoder. This has been implemented using the Tensorflow library and Python programming language.

As mentioned in Chapter 2.2.3, there is no unpooling function available in the Tensorflow library. To solve this issue, regular image resizing with bilinear interpolation was used to expand the inputs by a factor of two in both directions. The name 'unpooling' is still kept to better represent that the operation is a counterpart of the pooling operation applied earlier.

An autoencoder consists of several convolutional, pooling and fully-connected layers. Using these layers, the autoencoder will first compress the input image down into a small vector called latent vector. Afterward, it will decode and reconstruct the image from the latent vector. An example of an autoencoder is shown in Figure 3.8.

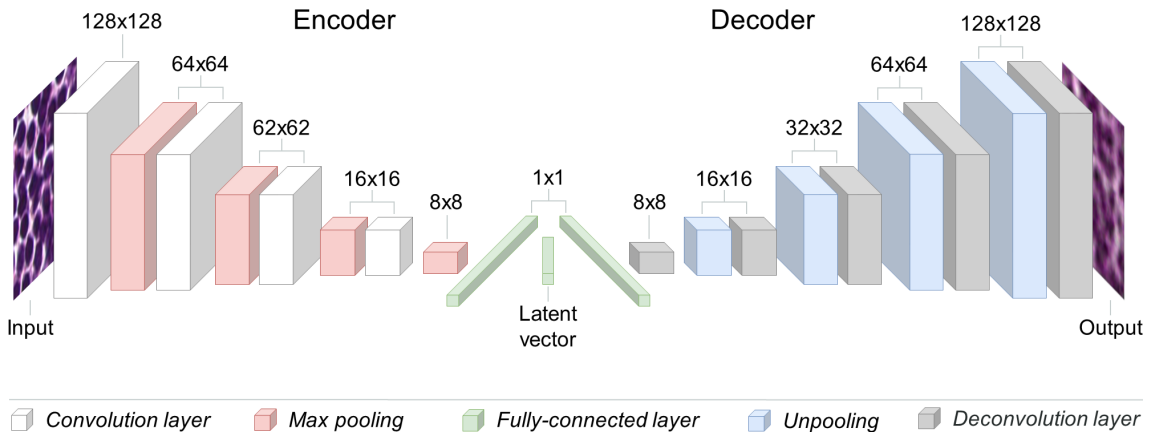


Figure 3.8: One possible architecture of an autoencoder. The encoder will compress the input image into the latent vector. The decoder is a mirror copy of the encoder, and will reconstruct the image from the latent vector. Data flows from left and to the right.

Latent vector size

One of the importaint criteria of the autoencoder, is the size of the latent vector. If this is too large or small, the autoencoder may not properly learn the features. The optimal size is highly dependent on the input images, and how many features that are present. It is therefore not straightforward setting a useful size.

The paper "*Learning deconvolution network for semantic segmentation*" [42] uses an autoencoder for feature extraction. Table 2 on page 9 contains a detailed list of the size of each layer. The paper uses a different input size of the images than used in this thesis, but the ratio can be calculated and applied. In data compression, the *space saving* is often useful, and defines the reduction in size relative to the uncompressed size. The paper uses input images of $224x224x3 = 150,528$, and the latent vector has the size 4096. The space saving is then given as:

$$Space\ saving\ used\ in\ [42] = 1 - \frac{4096}{150528} = 0.973 \tag{12}$$

This gives a 97 % compression of the input image, and will be the area of aim for the autoencoders used in the experiments in this thesis. The input images used in this thesis have the dimension $128x128x3 = 49,152$. A 97 % reduction in size results in:

$$49152 \times (1 - 0.973) = 1327 \tag{13}$$

This exact size is hard to achieve since the reduction between layers are determined by the pooling and fully-connected layers. But it acts as a guideline to how many layers to use. Both smaller and larger vectors will be experimented on.

Architecture
The convolutional layers contain the weights and biases which are the parameters that learns the features in the image. Pooling layers are used for reducing the size in the system. The reshape layer is a functions that maps a 3-dimensional volume into a 1-dimensional vector. And the fully-connected layers are network where all outputs are connected to all the inputs of the next layer.

Autoencoder hyperparameters
The convolutional layers could also have been used to reduce the size if the parameters (stride, filter kernel etc.) had been chosen accordingly. However, the program developed in this thesis is made to automatically test a multitude of different models with different hyperparameters. To avoid any architectural problems, zero-padding is added to the convolutional layers, and only pooling and fully-connected layers is used to reduce the size. Both stride and kernel size is kept constant at 1 and 3 respectively.

In addition, some other limitations have been chosen. All input images are square and of size 128. The filter kernel size is also square. Stride is equal along both axis. And the same padding is added along both axis. Based on these restrictions, the output size of a convolutional layer is:

$$Convolution\ output\ size = \frac{128 - 3 + 2 \times 1}{1} + 1 = 128 \tag{14}$$

Which means that the size is preserved, the output size is the same as the input size.

The most common settings for pooling layers are to use max pooling with kernel size $2x2$ and stride 2. This is also used in this system.

Layer structure
When combining the different layer types into a structure, the order they are stacked together may influence the result. It is not trivial to find the best order, and therefore several different autoencoder-structures are made and experimented on.

The layers were stacked together in an altering fashion, where each model had one more layer than the previous model. In total 19 different structures were designed in the beginning. A complete list of these are located in Appendix B.

The green half of the structures in Appendix B follows the pattern convolution-pooling pairs, followed by fully-connected layers. This is a basic convolutional neural network pattern often used, and was used by LeCun in 1989 in his LeNet as shown in Figure 1.4.

The other blue half follows the pattern convolution-convolution-pooling pairs, followed by fully-connected layers. This structure is used in e.g. [42], and AlexNet

shown in Figure 1.5 also have several connected convolution layers before pooling.

Next, four of these structures were chosen to experiment on. Most of the shallow structures were discarded because they only contain a few pooling layers, resulting in a large latent vector. Two structures from each half was chosen. Within each half, one structure with many conv-pool pairs and few fully-connected pairs was chosen. And one structure with fewer conv-pool pairs, and more fully-connected pairs was chosen.

This way a multitude of different structures is experimented on. Worried that some structures may give a bad result, experimenting on several kinds may raise the probability to find a good model. In Figure 3.9 below the four chosen structures are shown. Note that only the encoder structure is shown. The decoder structure is a clean mirror copy of the encoder, but uses deconvolution and unpooling instead of convolution and pooling.

Model	Input layer	Encoder										
A1	Input	conv	pool	conv	pool	reshape	FC	FC	FC			
	128x128x3	128x128x10	64x64x10	64x64x10	32x32x10	10240	5120	2560	1280			
A2	Input	conv	pool	conv	pool	conv	pool	conv	pool	reshape	FC	
	128x128x3	128x128x32	64x64x32	64x64x32	32x32x32	32x32x32	16x16x32	16x16x32	8x8x32	2560	1280	
A3	Input	conv	conv	pool	conv	conv	pool	reshape	FC	FC	FC	
	128x128x3	128x128x10	128x128x10	64x64x10	64x64x10	64x64x10	32x32x10	10240	5120	2560	1280	
A4	Input	conv	conv	pool	conv	conv	pool	conv	conv	pool	reshape	FC
	128x128x3	128x128x10	128x128x10	64x64x10	64x64x10	64x64x10	32x32x10	32x32x10	32x32x10	16x16x10	2560	1280

Figure 3.9: Architecture of the encoder part of the four different autoencoders used in the experiments. The decoder is a mirror copy of the encoder. Conv = convolutional layers, pool = pooling layers, FC = fully-connected layers.

By changing the number of filters used in the convolution layers, the volume changes which again result in a different latent vector size. By adjusting the number of filters, the latent vector can be adjusted. Since the correct latent vector size is unknown, several different numbers of filters will be experimented on.

For the fully-connected layers, the output size has been chosen to be half the size of the input. Meaning that each fully-connected layer reduces the current vector size by a factor of two.

3.4 Classifier

After the autoencoder has been trained, the structure is slightly altered. The decoder part is disconnected, and a set of three fully-connected layers are connected to the output of the encoder as shown in Figure 3.10. These three layers serve as the classifier and will output a prediction of how confident it is in the input image

belonging to each class. The class with the highest confidence is selected as the class the input belongs. Even though the autoencoder have been trained, these new layers are initiated randomly and need to be trained to be able to classify.

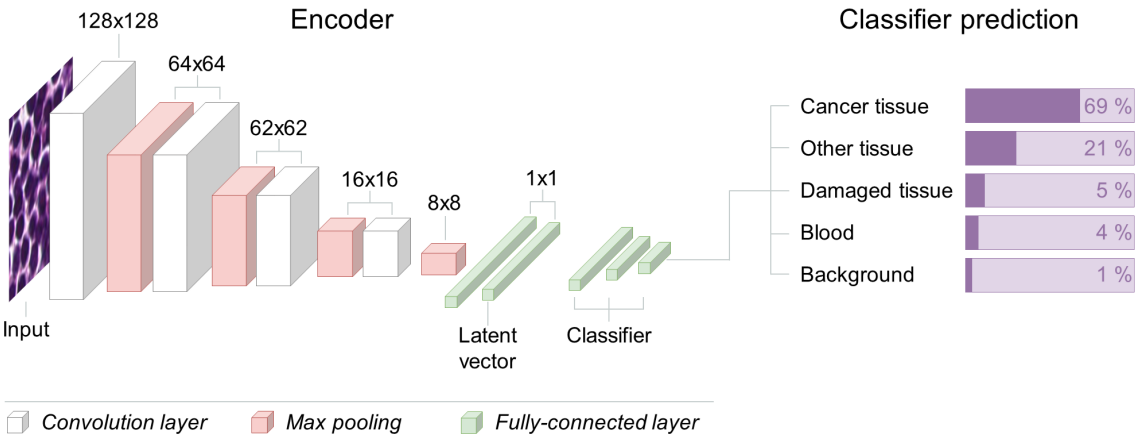


Figure 3.10: Encoder and classifier structure. The classifier will output a prediction of which class the input belongs.

To use three fully-connected layers in the end seems like a typical structure to use. It can be seen on both LeNet in Figure 1.4 and AlexNet in Figure 1.5, it was also used by Hinton in his deep belief net [24]. Even though it is not entirely comparable, as the encoder structures used in this thesis already have some fully-connected layers, as seen in Figure 3.9.

The two first layers are using the ReLU activation function, and the output layer are using the softmax activation function. It is this function that allows the network to predict between classes. The size of the output layer is determined by the number of unique classes in the dataset. The size of the two other layers are unknown and may affect the classification results, and are therefore experimented on.

4 Experiments and results

In this chapter the performance of the proposed system will be tested by conducting a set of experiments. Since the results of one experiment are the foundation of the next, both setup and results are presented in the same chapter. Some of the results will also be discussed here, to elaborate which options to use in the next experiments. All results will also be summarized and discussed in Chapter 5.

4.1 Preprocessing of SCN images

The first step of the proposed system is to preprocess the histological images. The process is shown in Figure 3.1. The SCN image is loaded into the preprocessing program, which divides the image into smaller tiles. Search block size of 1024x1024, and tile size of 128x128 are used. This means that each search blocks size is equal to 64 tiles.

In total seven histological images were preprocessed. Results from each run are displayed below in Table 2.

Table 2: Results after histological images are preprocessed.

Image	**Size (px)**	**Search block saved**	**Search block discarded**	**Tiles saved**	**Tiles discarded**	**Time (H:M:S)**
H671	129,410 x 86,276	291	360	192,640	105,344	03:21:57
H1722	89,474 x 84,228	195	225	117,137	82,543	02:55:37*
H2270	104,548 x 82,114	150	350	91,883	61,717	02:18:41*
H3137	78,252 x 52,610	73	155	47,933	26,819	00:54:14
H3395	108,416 x 85,250	265	255	168,995	102,365	03:54:32*
H5407	92,610 x 77,890	299	119	248,540	57,636	03:54:19
H7191	59,970 x 46,340	110	44	75,999	36,641	01:10:16
Total	N/A	1,383	1,508	943,127	473,065	18:29:36

Times in the table indicated by an asterisk(*) are runs where the discarded tiles is saved as a JPEG file. This is done to evaluate the categorization manually afterward. This extra saving of discarded tiles takes time, and therefore these runs have a higher runtime (relative to the size of the image).

The column *size* in Table 2 are referred to the size of the image after the white border have been removed by the program, and is the actual size of the image. All SCN images in the table have a default size of 306,939 x 106,259 px. before preprocessing.

After preprocessing of all seven images, a total of 943,127 tiles have been extracted

and will be used as dataset for upcoming experiments.

Some example of both saved and discarded tiles are shown below in Figure 4.1.

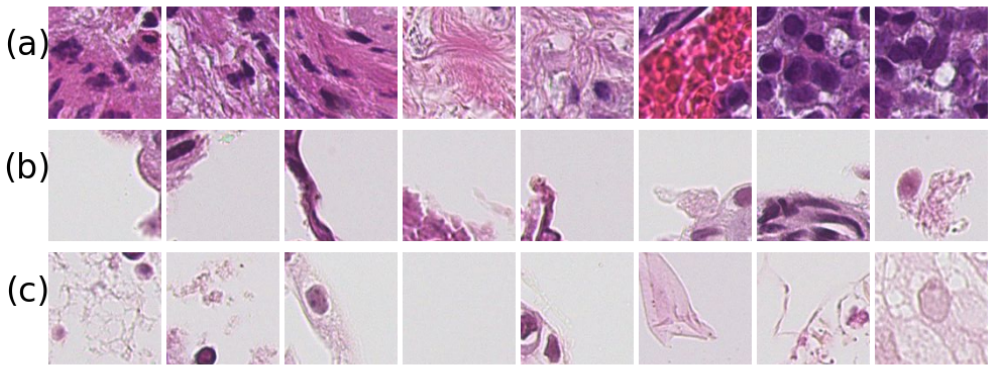


Figure 4.1: (a) and (b) are examples of tiles that are saved. (c) show tiles that are discarded.

4.2 Consistency of autoencoder

Since the weights are initialized at random, it is reasonable to think that the same model will produce different results if run multiple times. To test the consistency, the same model was trained ten times.

The model run in this experiment were structure ID 19 from Appendix B. It was run for 200 epochs on 40,000 tiles using learning rate 0.001, batch size 16 and ten filters. This model was trained ten individual times, where the cost for each epoch was saved.

In Figure 4.2 below, the average of all ten models have been calculated and plotted. In addition, the standard deviation of all models have been computed and are shown as red error bars around the average. For the original plot with all models, see Appendix D.

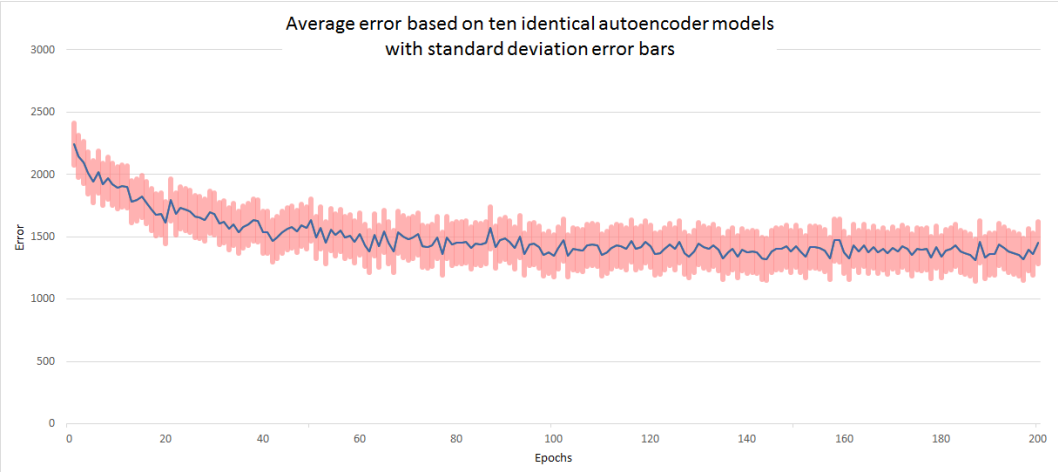


Figure 4.2: Blue curve shows average results from ten autoencoders of the exact same model. The red error bars shows the standard deviation.

4.3 Finding the best autoencoder

When training an autoencoder, several parameters needs to be determined. Some of these were restricted to certain values in Chapter 3.3, while others need to be found via experiments. The four autoencoder models described in Figure 3.9 will be trained with a multitude of different parameters to try and find the best combination of these.

To greatly reduce the computational time when training all of these models, the dataset were reduced to contain only 50,000 images and each model was trained for 100 epochs. The models will not have converged to a final state with this scheme, but it should be enough to separate the good models vs. the poor.

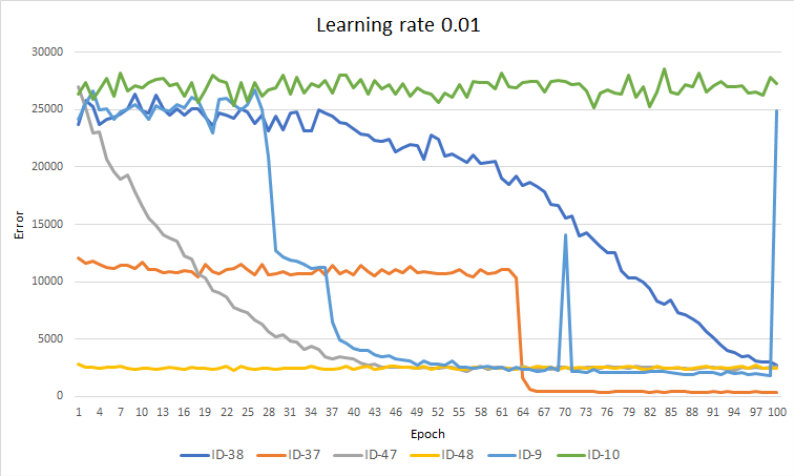
In total 48 different models were trained, all based on the four main models from Figure 3.9. Both learning rate, batch size and number of filters were changed to try and find the best parameters. A complete list of results for each model are shown in Appendix C.

Learning rate

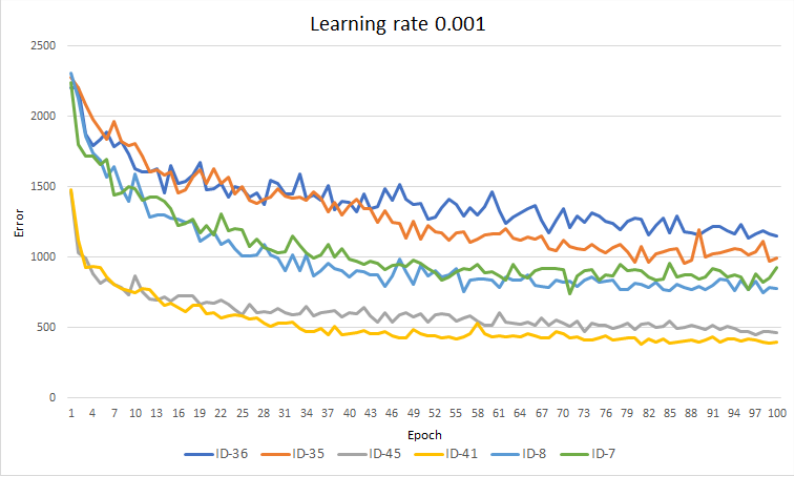
Initial models were trained using learning rate of 0.01 and 0.001. However, due to the poor performance of the highest learning rate, this was abandoned for the benefit of lower learning rates of 0.001 and 0.0001.

To compare the performance, the models from Appendix C with similar parameters are compared. For instance, ID no. 5, 7 and 9 are the same model with the same parameters, except for learning rate. This allows us to evaluate the performance of learning rate. Similar, five other models that were trained on all three learning rates with other parameters constant were selected. The results of these models are

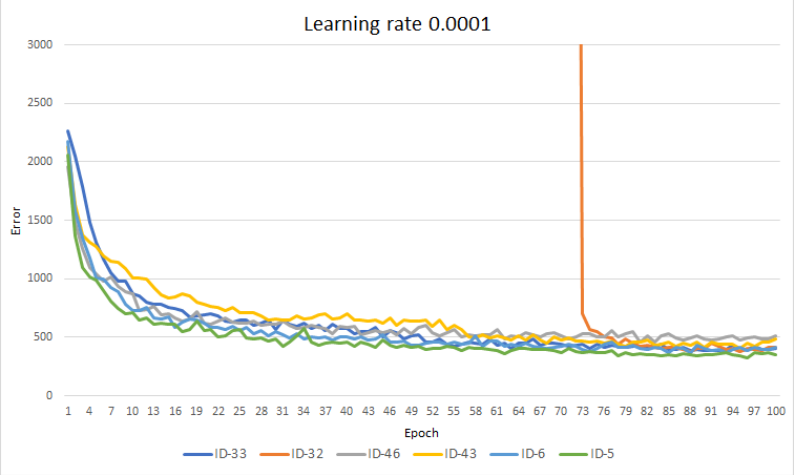
plotted in Figure 4.3.



(a) Learning rate 0.01.



(b) Learning rate 0.001.



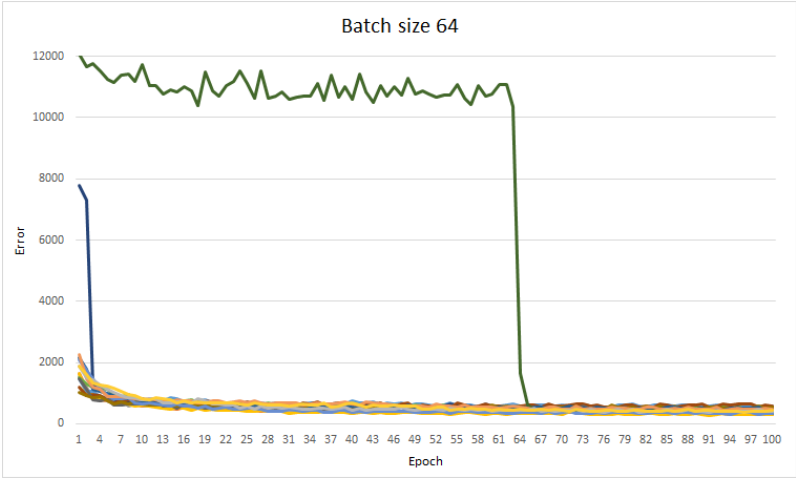
(c) Learning rate 0.0001.

Figure 4.3: Result of autoencoder models trained on different learning rates.

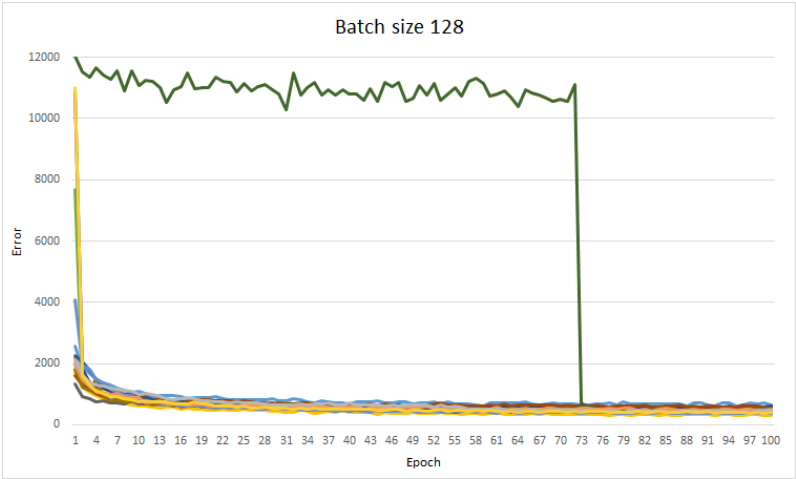
It is quite obvious that a lower learning rate produces better results, and so the learning rate of 0.0001 is chosen for upcoming experiments.

Batch size

Two main batch sizes were used during testing, 64 and 128. In a similar fashion as for learning rate, all models with each batch size is combined together and plotted in Figure 4.4.



(a) Batch size 64



(b) Batch size 128

Figure 4.4: Result of autoencoder models trained with different batch sizes.

The differences are not as distinct as for the learning rate case, but there is still some minor details to notice. For instance, the green model which seem to be stuck in a local minimum and suddenly drops vertically down. This happens in both plots, but for batch size 64 it happens about ten epochs earlier. Also, looking closely at epoch 1, several of the models for 128 batch size are starting with a higher cost than for the batch size 64 case.

These two minor details suggest that using a batch size of 64 is somewhat better that 128.

Number of filters

Comparing the number of filters is a bit differently than the two previous cases. Since a larger amount of filters produces a greater volume, it results in a larger latent vector. The larger the latent vector is, the more information is available for the decoder to produce a more accurate reconstructed image, which will produce a lower error. However, as mentioned in Chapter 2.2.7, it is not necessarily the autoencoder which produces the best reconstructed image which is the best foundation for the classifier.

In the experiment the following number of filters were used; 8, 10, 12, 16, 32 and 40. These, in combination with the different autoencoder models, produced latent vectors of four different sizes; 512, 1024, 1280 and 1536. Each of these vector sizes corresponds to a compression of 98.9 %, 97.9 %, 97.4 %, 96.8 % respectively.

It makes more sense to compare models of equal latent vector size, than an equal amount of filters. To compare these models, an average of all models were calculated and plotted. Seven of the 48 models were not included in the average due to poor performance. For a list of which models were included, see Appendix E. The results are shown in Figure 4.5.

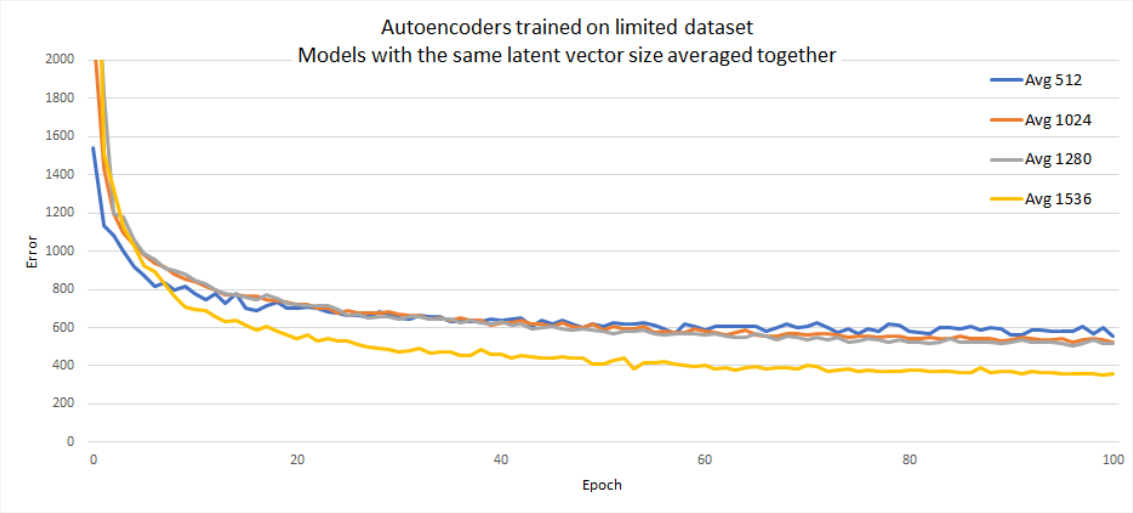


Figure 4.5: Average size of models with four different latent vector sizes. Note that the Y-axis have been reduced to emphasize the lower Y-values.

Not surprisingly, the models with larger latent vector reconstructed the images with a lower error on average. Moreover, for each step up in latent vector size, the average error increased slightly. This experiment shows that all four vector sizes are capable of reconstructing the input images, meaning that the latent vector contains important feature information in all cases. The 1536 and 1280 sizes are therefore

discarded, and further experiments will be conducted on models with latent vector of size 512 and 1024.

Model structure
Until now, a learning rate of 0.0001, batch size of 64 and number of filters which produces latent vector size of 512 and 1024 have been chosen. Figure 4.6 shows the result of each model trained using these parameters (using latent vector 1024 only).

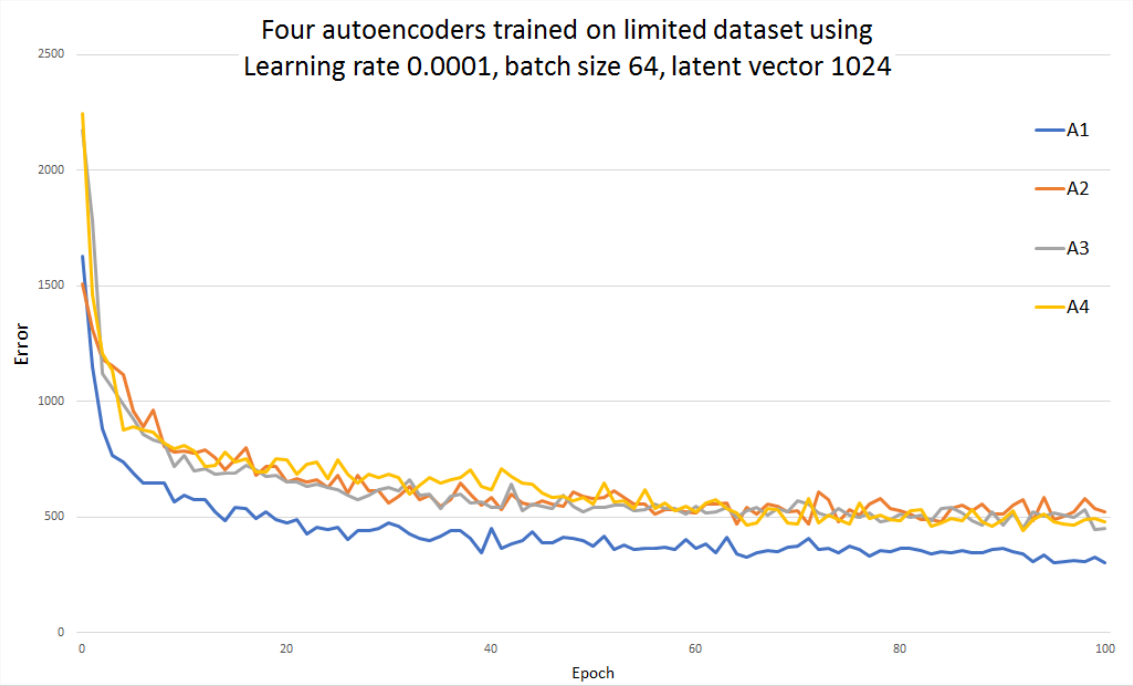


Figure 4.6: Four autoencoders trained using the same parameters. Learning rate 0.0001, batch size 64 and latent vector of size 1024.

It seems like model A1 manages to reconstruct the images with marginally lower error than the other models. However, this is only the result of one trained model for each case. Reproducing the results in Figure 4.6 multiple times could maybe produce another best model. Nevertheless, model A1 is chosen. As shown in Figure 3.9, the four models came in pairs of two. So it is natural to bring along model A2 together with model A1 for further experimenting.

4.4 Training autoencoders

This chapter is a continuing from previous chapter where the parameters for the autoencoders were determined. Training an autoencoder corresponds to step 2 of the proposed system as shown in Figure 3.2.

The four autoencoders were trained for 200 epochs on the full dataset of 943,127 tiles. An overview of the trained autoencoders are shown in Table 3 and the training results are shown in Figure 4.7.

Table 3: Four main autoencoders trained on the full dataset

Model	Learning rate	Batch size	Filters	Latent vector	Epochs	Time
A1	0.0001	64	4	512	200	41:31:05
A2	0.0001	64	16	512	200	58:00:03
A1	0.0001	64	8	1024	200	54:17:23
A2	0.0001	64	32	1024	200	76:35:47

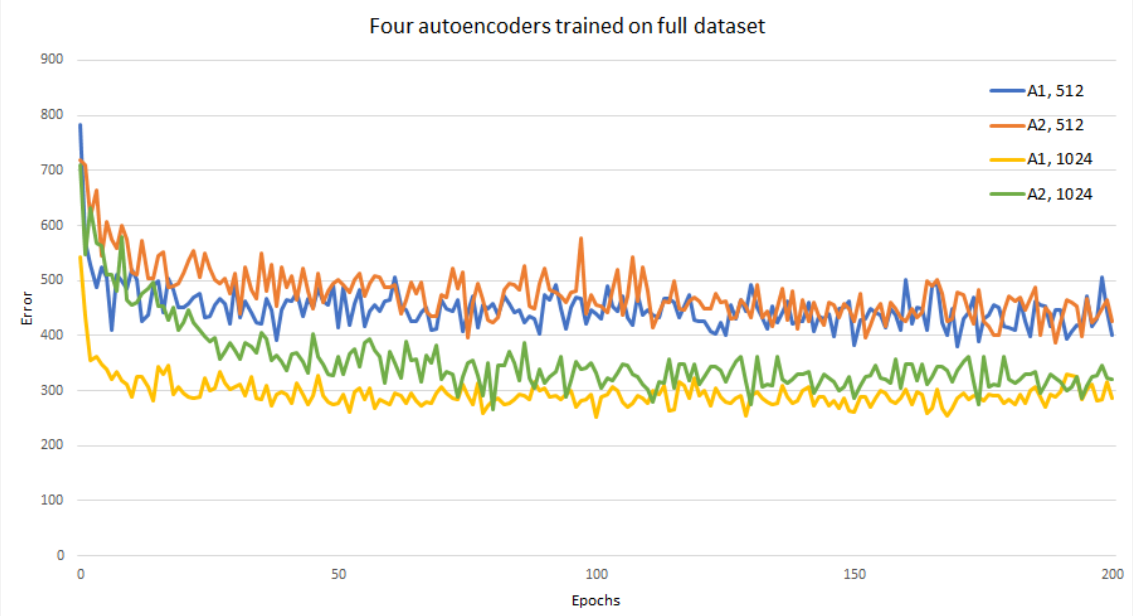


Figure 4.7: Four autoencoders trained on the full dataset. Model A1 and A2 was trained using both latent vector size of 512 and 1024 for this experiment.

4.5 Finding the best classifier

In this chapter the decoder of the autoencoder will be exchanged with a classifier and trained on a labeled dataset. This chapter corresponds to step 3 and 4 of the proposed system as shown in Figure 3.3.

Labeled dataset
As shown in Figure 3.3, a small labeled dataset is necessary to train the classifier. Since the texture inside each class varies quite much, a lot of data is needed to create

a useful classifier. The labeled dataset was manually picked using the ImageScope software to mark small regions consisting of the same class and then using the preprocessing program to extract tiles from this region. Tiles of each class were put into folders, where each folder represented the label for its class. The data were then augmented before used to train the classifier. Total number of labeled tiles are shown in Table 4.

Table 4: Labeled dataset

Name of class	**No. tiles**	**After augmentation**	**Percent of dataset**
Cancer tissue	6,817	54,536	36 %
Other tissue	2,004	16,032	11 %
Damaged tissue	5,535	44,280	29 %
Blood	3,016	24,128	16 %
Background	1,667	13,336	9 %
Total	19,039	152,312	100 %

Calculation of the number of tiles extracted before augmentation in comparison to the full dataset is given as:

$$\frac{19039}{943127} = 0.02 \tag{15}$$

Which shows that the extracted dataset is equivalent to only 2 % of the full dataset.

Classifiers

As described in Chapter 3.4, the classifier consists of three fully-connected layers. The size of the output layer is set equal to the number of classes to predict on, five in this case. The size of the two first layers are unknown and experimented on in this chapter.

The result from training the autoencoder revealed that a lower learning rate performed better, therefore an even lower learning rate of 0.000001 was used to train the classifiers.

Two different sizes will be tested for each of the two layers. This results in four different classifiers to test as shown in Table 5.

Table 5: Four different classifiers used in the experiments

Classifier	Learning rate	Filters 1th	Filters 2nd	Epochs
C1	0.000001	128	256	40
C2	0.000001	128	256	40
C3	0.000001	256	512	40
C4	0.000001	256	512	40

Experiment
With four classifiers trained on each autoencoder results in a total of 16 different models. These classifiers will be trained on the full labeled dataset, using 4-fold cross validation. 4-fold is used instead of 10-fold to save time, and allow us to see how each model perform. The best models will then trained once more using 10-fold cross validation to confirm its performance.

Table 6 shows the result for the 16 classifiers.

Table 6: Results from 16 classifiers trained on full labeled dataset. Average accuracy after 4-fold cross-validation are shown together with standard deviation. Best result is ID 11.

ID	Autoencoder model	Latent vector size	Classifier model	Accuracy	Standard deviation
1	A1	512	C1	72.3 %	± 13.0 %
2	A1	512	C2	29.1 %	± 0.2 %
3	A1	512	C3	33.1 %	± 7.6 %
4	A1	512	C4	70.2 %	± 16.9 %
5	A1	1024	C1	47.0 %	± 4.5 %
6	A1	1024	C2	57.5 %	± 13.2 %
7	A1	1024	C3	29.1 %	± 0.3 %
8	A1	1024	C4	72.9 %	± 7.4 %
9	A2	512	C1	29.1 %	± 0.2 %
10	A2	512	C2	77.5 %	± 23.0 %
11	A2	512	C3	97.9 %	± 0.8 %
12	A2	512	C4	37.0 %	± 9.1 %
13	A2	1024	C1	37.0 %	± 8.8 %
14	A2	1024	C2	47.0 %	± 4.4 %
15	A2	1024	C3	47.3 %	± 25.3 %
16	A2	1024	C4	33.0 %	± 7.8 %

4.6 Verification of best result

This chapter used the best model found in the previous experiment, and trained it once more to verify its performance. The model used is autoencoder model A2 with latent vector size 512, using classifier model C3. The classifier was be trained on full labeled dataset for 80 epochs with 10-fold cross-validation to evaluate it. Result of the model are listed in Table 7.

Table 7: Final result on the best model evaluated using 10-fold cross-validation.

Autoencoder model	**Latent vector size**	**Classifier model**	**Epoch**	**Time (H:M:S)**	**Accuracy**	**Standard deviation**
A2	512	C3	80	18:56:27	97.7 %	± 3.2 %

To see how the model performed on individual inputs, some example prediction by the classifier are shown in Figure 4.8. The blue bars below each input image shown how confident the model is in its prediction. Confidence is shown as 0 % to the left, and rises toward the right end.

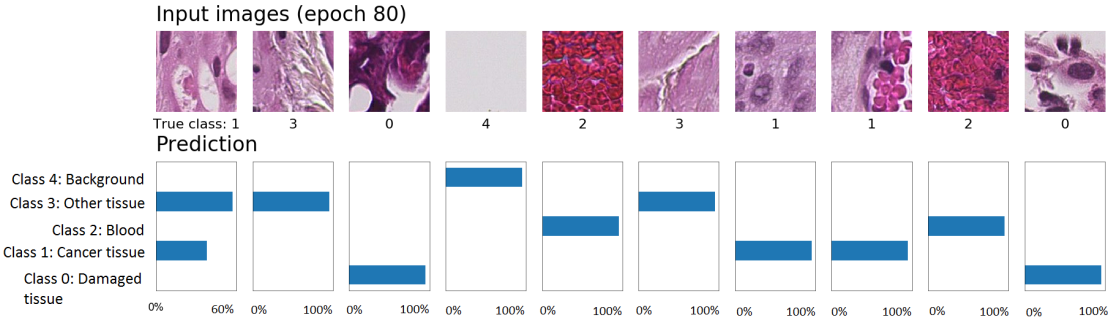


Figure 4.8: The classifier outputs a confidence percentage for each input image. If the blue bar is further to the right, the model is more confident in its prediction.

To further analyze the models performance the confusion matrix is shown in Figure 4.9. The confusion matrix is the total confusion matrix, meaning that the results from each of the ten runs are combined.

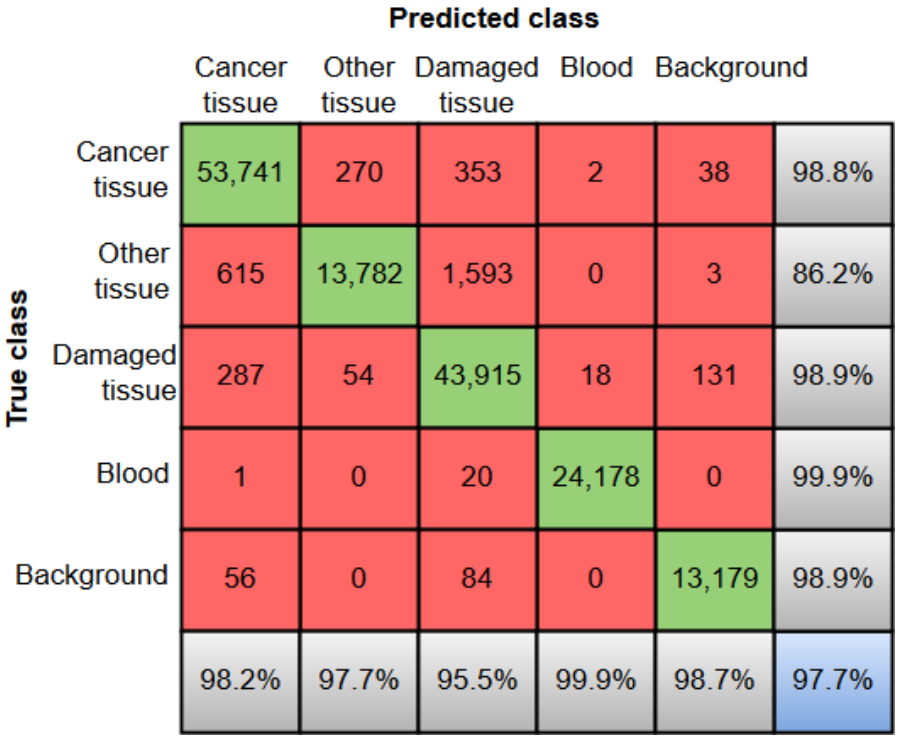


Figure 4.9: Confusion matrix for autoencoder model A2 with latent vector size 512, using classifier model C3 and 10-fold cross-validation.

5 Discussion

This chapter includes a short review of the python scripts used in the experiments. The experimental results and the system performance is summarized and discussed. Suggested improvements and future work will also be presented.

5.1 Analysis of the Python scripts

This chapter contains a small review of the Python scripts written during this thesis. Appendix A contains a complete list of all features available for the Python scripts.

5.1.1 Preprocessing

The preprocessing program is very flexible and can be used on any image format of any size. A small amount of user input is necessary to be able to run the program. As the program is running, it is continuous giving feedback to the user about the current step and an estimate of how long until the program is finished. Going over the entire image with block search greatly reduces the total running time of each image.

The major problem is when using binary masks, as used to extract a region of interests to make the labeled dataset. It is a computational heavy and inefficient operation to run on the large SCN images, and crashed the program on the largest images. To avoid this, the images were first cropped to reduce its size, and then the regions where extracted.

5.1.2 Autoencoder and classifier

The program used to train both the autoencoder and classifier has proven to work very well. A considerable amount of time has been dedicated to making the program able to run uninterrupted for extended periods of time. The program can train multiple models in sequence, automatically restarts the program where it stopped if an interrupt should occur, and e-mail notification to inform the user of progress have been set up.

5.2 Experimental results

The chapter will summarize and discuss the experimental results.

5.2.1 Preprocessing

Seven histological images were preprocessed, resulting in a total of 943,127 saved tiles.

Three of the preprocessing runs saved the discarded tiles in a separate folder. These were manually studied to see if any tiles with a lot of tissue had wrongly been discarded. The program did a decent job, and no tiles were found that could be considered as a useful tile.

The folder containing the saved tiles was also manually studied, and a small proportion of background tiles had wrongly been saved as useful tiles. Figure 4.1 illustrates some example tiles of each case. The tiles which are incorrectly saved as useful tiles are quite similar to the ones that were discarded. However, they have debris/tissue of a darker color. This dark color shifts the histogram of the tile, and the program therefore saves it. This is not a critical issue, as the autoencoder needs background tiles to be able to learn its features.

5.2.2 Consistency of autoencoder

One model was trained ten individual times to check the consistency of an autoencoder. Average results with standard deviation error bars are shown in Figure 4.2, and a plot with all ten models are shown in Appendix D.

The standard deviation stays approximately the same throughout all epochs. Meaning that on average, the models does not slide further away from each other, or converges closer to each other. It seems like the models which start out with a lower error compared to the other models also ends up with a smaller error after 200 epochs compared to the other models, and vice verse for the models with a higher error.

5.2.3 Selecting the best autoencoder

In total 19 models were originally designed, a list of these can be found in Appendix B. From these, four models were chosen for further investigation. To find the best combination of hyperparameters, 48 different models were trained on a limited dataset. Results for all 48 models are listed in Appendix C.

An analysis of these models showed that the best hyperparameters were learning rate of 0.0001 and batch size of 64. The 48 models produced results with four different latent vector size; 512, 1024, 1280 and 1536. The two smallest sizes were chosen for further experimentation.

Model A1 looked slightly more promising than the other and was therefore chosen. Together with model A1, model A2 was also selected as these two models have the

same structure, but different amount of convolutional/fully-connected layers. Model A1 and A2 were trained on the full dataset for 200 epochs using two latent vector sizes, producing four models in total.

Analyzing the training graph in Figure 4.7, all four models did a similar and good job. Three of the models training is almost equal, but model A1 with latent vector 1024 is slightly better than the others with a smaller error. However, as emphasized earlier, a smaller reconstruction error does not necessarily mean better classification results.

5.2.4 Selecting the best classifier

For each of the four autoencoders, four different classifiers were trained to produce a total of 16 different classifiers. Each classifier was validated using 4-fold cross-validation. Results are presented in Table 6 and show that majority of the classifiers performed poorly, with 11 out of the 16 classifiers getting an accuracy below 60 %. Four of the classifiers performed mildly better, with accuracy around 70-79 %.

The best model got an accuracy of 97.9 % and a standard deviation of only 0.8 %. This model consists of autoencoder model A2 with latent space 512 and classifier model C3.

5.2.5 Verification of the best model

Since the previous experiments were only conducted using 4-fold cross-validation, the best model was run once more to verify its performance. This time the model was evaluated using 10-fold cross-validation.

In this re-evaluation the model got an average accuracy of 97.7 % with a standard deviation of 3.2 %.

Some individual predictions are shown in Figure 4.8. On nine of the examples the model shows a 100 % confident in its prediction, and was correct on these samples. On the first case however, the model gives about a 40/60 % confident to class 1 and 3 respectively. The model predicts the class with the highest confidence, so it picks class 3. The true class however was class 1, so the model was wrong in this case.

These are only a few samples of the prediction done by the model. To see the overall performance of predictions done by the model, all predictions are combined in the confusion matrix in Figure 4.9. This allows us to analyze the classifier even further.

The classifier did an excellent job classifying tiles of blood, with both a sensitivity and precision of 99.9 %.

Another great performance is classification of cancer and background tiles. Both classes have sensitivity and precision in the range 98.2-98.9 %.

Almost all tiles consisting of damaged tissue was correctly classified as so, giving it a sensitivity of 98.9 %. However, a lot of tiles belonging to other classes were also wrongly classified as damaged tissue, giving it a precision of 95.5 %.

The poorest performance of the classifier was with tiles belonging to the class other tissue were only 86.2 % of the tiles were correctly classified. All wrongly predicted samples were either misclassified as cancer tiles or damaged tiles. The precision of the class was somewhat higher, with a 97.7 %.

5.3 Suggested improvement

As the histological images are split up into tiles during preprocessing, the spatial correlation between tiles is lost. When classifying one tile as cancer tissue, there is a larger probability that the next tile also will be cancer tissue. But the system does not know this.

Maybe this could be solved by implementing a Bayesian probability algorithm. Before classifying a tile, there is a prior knowledge given by the tile's neighborhood. This prior probability is then updated to a posterior probability as new relevant data (the evidence) is presented by the classifier.

One way to solve this, is to classify each tile in the image and only keep the classes where the classifier is confident about the class. All tiles which have an approximately equal probability to be one of several classes are rescanned using the Bayesian approach given the prior knowledge from the first scan. This would be more time consuming, but may help remove misclassification of those difficult-to-classify tiles.

5.4 Future work

New structures
Even though this thesis have experimented with a range of models, it is not obvious that the structure of either autoencoder or classifier is optimal. Experimentation with different structures, or hyperparameters, could reveal better models.

More data
The key to a good model is available training data. Only seven of the total 360 histological images were utilized to create the unlabeled training dataset for the autoencoder. Preprocessing more of the histological images creates a larger dataset, which could contribute to train more robust models.

The classifier was trained on a small labeled dataset compared to the large unlabeled dataset. The classifier from Chapter 4.6 could be used to extract and automatically label more data. This larger labeled dataset could be used to further train the classifier model and maybe reveal even better results.

Create heat maps
Since the classifier made in this thesis is capable of distinguishing between several classes in the histological images, the results could be used to create a heat map of the histological image. If the location of each extracted tile is saved, the class prediction could be linked back to its location. By giving each class a different color, the heat map will give the pathologist a graphical representation of the data and where the different classes are located in the image.

Predict relevant cancer information
This thesis has proposed a system consisting of six steps to classify stage, grade, recurrence or progression of histological images. However, only step one until four have been experimented and tested. Future work involves using the results of this thesis and conducting experiments on step five as shown in Figure 3.5, and finally test step six as shown in Figure 3.6.

6 Conclusion

This thesis proposes a new system to automatically predict cancer's grade, stage, recurrence and progression based on histological images. The system consists of six steps, where four of the steps have been analyzed and experimented on in detail in this thesis and the two remaining steps noted as future work.

The histological images contain texture of several different classes which were categorized as; cancer tissue, damaged tissue, other tissue, blood and background. The aim of this thesis was to make a classifier capable of categorizing tiles into these classes.

A set of experiments was conducted to find an optimal set of parameters and structures of the classifier. The steps of training the classifier consist of first pre-training a deep autoencoder on an unlabeled dataset, and afterward fine-tuning a classifier on a labeled dataset.

The best classifier was evaluated using 10-fold cross-validation and got a final result of 97.7 % accuracy with a standard deviation of only 3.2 %.

Based on the results presented in this thesis, future work on step five and six seems both interesting and promising.

7 References

[1] I. K. Larsen, *Cancer incidence, mortality, survival and prevalence in Norway*, 2016. [Online]. Available: https://www.kreftregisteret.no/globalassets/cancer-in-norway/2015/cin-2015.pdf

[2] O. M. Mangrud, *Identification of patients with high and low risk of progression of urothelial carcinoma of the urinary bladder stage Ta and T1*, 2014.

[3] R. Lozano, M. Naghavi, K. Foreman, and S. Lim, "Global and regional mortality from 235 causes of death for 20 age groups in 1990 and 2010: A systematic analysis for the Global Burden of Disease Study 2010," *The Lancet*, vol. 380, no. 9859, pp. 2095–2128, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140673612617280

[4] E. A. Janssen and V. Kvikstad, "Bedre diagnostikk av TaT1-uroteliale karsinomer i urinblære," 2015.

[5] J. Urdal, *Image processing and classification of urothelial carcinoma using tissue sample images*, 2016. [Online]. Available: https://brage.bibsys.no/xmlui/bitstream/handle/11250/2414014/Urdal_Jarle.pdf?sequence=1

[6] J. Urdal, K. Engan, V. Kvikstad, and E. A. Janssen, "Prognostic prediction of histopathological images by local binary patterns and RUSBoost," *Accepted for publication in Proceedings of EUSIPCO 2017, Greece, August*, 2017.

[7] S. S. Garapati, L. M. Hadjiiski, K. H. Cha, H.-P. Chan, E. M. Caoili, R. H. Cohan, A. Weizer, A. Alva, C. Paramagul, J. Wei, and C. Zhou, "Automatic staging of bladder cancer on CT urography," *SPIE*, 2016. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=2507203

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11263-015-0816-y

[9] A. Munoz, "Machine Learning and Optimization," *Courant Institute of Mathematical Sciences*, 2014. [Online]. Available: https://www.cims.nyu.edu/~munoz/files/ml_optimization.pdf

[10] L. Deng and D. Yu, "Deep Learning: Methods and Applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3-4, pp. 197—-387, 2013. [Online]. Available: https://www.microsoft.com/en-us/research/publication/deep-learning-methods-and-applications/#

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, 2016.

[12] W. S. McCulloch and W. Pitts, "A Logical Calculus of the Idea Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943. [Online]. Available: http://www.cse.chalmers.se/~coquand/AUTOMATA/mcp.pdf

[13] J. L. G. Rosa, "Biologically Plausible Artificial Neural Networks," *InTechOpen*, 2013. [Online]. Available: https://cdn.intechopen.com/pdfs-wm/41965.pdf

[14] P. Husbands and O. Holland, "The Ratio Club: A Hub of British Cybernetics," *The Mechanical Mind in History*, no. September 1949, pp. 91–148, 2008. [Online]. Available: http://users.sussex.ac.uk/~philh/pubs/Ratio2.pdf

[15] A. M. Turing, "Turing. Computing machinery and intelligence," *Mind*, vol. 59, no. 236, pp. 433–460, 1950. [Online]. Available: www.jstor.org/stable/2251299

[16] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.588.3775

[17] M. Minsky and S. Papert, "Perceptrons: an introduction to computational geometry," no. MIT Press, 1969. [Online]. Available: https://mitpress.mit.edu/books/perceptrons

[18] J. Schmidhuber, "Deep Learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015. [Online]. Available: https://arxiv.org/abs/1404.7828

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learining Internal Representations by Error Propagation," pp. 318–362, 1986. [Online]. Available: http://www.cs.toronto.edu/~fritz/absps/pdp8.pdf

[20] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0893608089900208

[21] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991. [Online]. Available: http://zmjones.com/static/statistical-learning/hornik-nn-1991.pdf

[22] G. Piatetsky, "Interview with Yann LeCun, Deep Learning Expert, Director of Facebook AI Lab," 2014. [Online]. Available: http://www.kdnuggets.com/2014/02/exclusive-yann-lecun-deep-learning-facebook-ai-lab.html

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *IEEE*, 1998. [Online]. Available: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

[24] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006. [Online]. Available: http://www.mitpressjournals.org/doi/10.1162/neco.2006.18.7.1527

[25] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pp. 1–8, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1553374.1553486

[26] A. Krizhevsky, I. Sutskever, and H. Geoffrey E., "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems 25 (NIPS2012)*, pp. 1–9, 2012. [Online]. Available: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[27] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, 2017. [Online]. Available: https://www.nature.com/nature/journal/v542/n7639/full/nature21056.html

[28] "DoMore! receives Lighthouse project grant from the Norwegian Research Council," 2016. [Online]. Available: http://oslocancercluster.no/2016/04/27/domore-receives-lighthouse-project-grant-from-the-norwegian-research-council/

[29] "DoMore Project - Improving diagnosis by utilizing Big Data and software-driven automation of pathology." [Online]. Available: http://domore.no/Projects/utilize-big-data

[30] D. Wang, A. Khosla, R. Gargeya, H. Irshad, and A. H. Beck, "Deep Learning for Identifying Metastatic Breast Cancer," pp. 1–6, 2016. [Online]. Available: http://arxiv.org/abs/1606.05718

[31] W. Commons, "T-stage cancer diagram by Cancer Research UK (Original email from CRUK) [CC BY-SA 4.0 (http://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons," 2014. [Online]. Available: https://commons.wikimedia.org/wiki/File:Diagram_showing_the_T_stages_of_bladder_cancer_CRUK_372.svg

[32] T. Dettmers, "The brain vs. deep learning — a comparative analysis," 2015. [Online]. Available: http://timdettmers.com/2015/07/27/brain-vs-deep-learning-singularity/

[33] V. M. Goncalves, K. M. Honorio, and A. B. Ferreira da Silva, "Applications of Artificial Neural Networks in Chemical Problems," 2016. [Online]. Available: https://www.intechopen.com/books/artificial-neural-networks-architectures-and-applications/applications-of-artificial-neural-networks-in-chemical-problems

[34] W. Commons, "Diagram of a neuron [CC BY-SA 4.0 (http://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons," 2008. [Online]. Available: https://commons.wikimedia.org/wiki/File:Neuron_-_annotated.svg

[35] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of Physiology*, vol. 148, no. 3, pp. 574–591, 1959. [Online]. Available: http://doi.wiley.com/10.1113/jphysiol.1959.sp006308

[36] ——, "Receptive of cells in striate cortex of very young , visually inexperienced kittens," 1963. [Online]. Available: http://hubel.med.harvard.edu/papers/HubelWiesel1963Jneurophysiol.pdf

[37] ——, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of Physiology*, vol. 195, no. 1, pp. 215–243, 1968. [Online]. Available: http://doi.wiley.com/10.1113/jphysiol.1968.sp008455

[38] Nobelprize.org, "The Nobel Prize in Physiology or Medicine 1981," 1981. [Online]. Available: http://www.nobelprize.org/nobel_prizes/medicine/laureates/1981/

[39] A. Karpathy, "Convolutional Neural Networks for Visual Recognition - Stanford University," 2017. [Online]. Available: http://cs231n.github.io/convolutional-networks/

[40] GoogleResearch, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: http://download.tensorflow.org/paper/whitepaper2015.pdf

[41] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2528–2535, 2010. [Online]. Available: http://www.matthewzeiler.com/pubs/cvpr2010/cvpr2010.pdf

[42] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 11-18-Dece, pp. 1520–1528, 2016. [Online]. Available: https://arxiv.org/abs/1505.04366

[43] Tensorflow, "Tensorflow deconvolutional operation," 2017. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/layers/conv2d_transpose

[44] P. Sibi, S. Jones, and P. Siddarth, "Analysis of different activation functions using back propagation neural networks," *Journal of theoretical and applied Information Technology*, vol. 47, no. 3, pp. 1264–1268, 2013. [Online]. Available: http://www.jatit.org/volumes/Vol47No3/61Vol47No3.pdf

[45] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," *AISTATS '11: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, vol. 15, pp. 315–323, 2011. [Online]. Available: http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

[46] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, second edition ed., 2001.

[47] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14539

[48] R. Kohavi, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," no. March 2001, 2016. [Online]. Available: http://robotics.stanford.edu/~ronnyk/accEst.pdf

[49] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent Models of Visual Attention," *Advances in Neural Information Processing Systems 27*, vol. 27, pp. 1–9, 2014. [Online]. Available: http://papers.nips.cc/paper/5542-recurrent-models-of-visual-attention.pdf

[50] M. Satyanarayanan, A. Goode, B. Gilbert, J. Harkes, and D. Jukic, "OpenSlide: A vendor-neutral software foundation for digital pathology," *Journal of Pathology Informatics*, vol. 4, no. 1, p. 27, 2013. [Online]. Available: http://www.jpathinformatics.org/text.asp?2013/4/1/27/119005

[51] J. Cupitt and K. Martinez, "VIPS: An image processing system for large images," *Proc. SPIE*, vol. 2663, pp. 19–28, 1996. [Online]. Available: https://eprints.soton.ac.uk/252227/1/vipsspie96a.pdf

[52] K. Martinez and J. Cupitt, "VIPS - A highly tuned image processing software architecture," *Proceedings - International Conference on Image Processing, ICIP*, vol. 2, pp. 574–577, 2005. [Online]. Available: https://eprints.soton.ac.uk/262371/1/martinezcupitt13.pdf

[53] J. Cupitt, "Vips - Speed and Memory Use," 2017. [Online]. Available: http://www.vips.ecs.soton.ac.uk/index.php?title=Speed_and_Memory_Use

[54] G. Litjens, C. I. Sánchez, N. Timofeeva, M. Hermsen, I. Nagtegaal, I. Kovacs, C. Hulsbergen - van de Kaa, P. Bult, B. van Ginneken, and J. van der Laak, "Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis," *Scientific Reports*, vol. 6, no. 1, p. 26286, 2016. [Online]. Available: http://www.nature.com/articles/srep26286

A Python code

PreProcessing.py
This file is used to preprocess the whole-slide SCN images. It have several features that can be turned on/off inside the file. The program outputs tiles of the input image. Some of the features available:

- Supported input file format: SCN, JPEG, PNG, TIFF, BMP.
- Specify both tile-size and search block size.
- The program can handle any amount of images in one run. Locate all the images that need processing in the appropriate folder. If the dataset consists of several classes that needs to be separated, create one folder for each class and the program will save the tiles in corresponding folders.
- The program can remove any white border around SCN images.
- If an XML-file is present, the program will read the file and draw the region on the image. The user can specify if the region inside or outside the border should be masked out.
- User can chose to save the deleted tiles or not.
- The program calculates an estimate of how long time remains until finished.
- Useful information about the current state of the program is presented to the user.
- Logging is saved to text-file.

Augmentation.py
This program will use augmentation techniques to increase a dataset 8 times. Specify a input-folder which contains the dataset to augment. The program will augment the data and save them in the output folder specified by the user. Augmentation is applied as explained in augmentation chapter.

Autoencoder.py
This program is used to train both the autoencoder and classifier. Main features of the program:

- Specify an array of different hyperparameters. The program will then train one model for each parameter combination.
- Turn shuffling of the dataset on/off.

- Set requirements for training which will stop the training if not fulfilled.
- Run on single or multiple GPU's.
- Saving and restoring models.
- Supported input file format: JPEG, PNG.
- Support both grayscale and color images.
- Logging is saved to both text- and Excel-files.
- The program produces example images at the end of each epoch.
- K-fold cross validation is implemented when training classifier, K value can be set to any integer larger than 2.
- Confusion matrix is calculated and saved every epoch when training classifier.
- E-mail notification to notify user of progress during program runs.
- A separate watchdog program is implemented that automatically restarts and restore the program if the program should stop due to an unexpected server issue or similar.

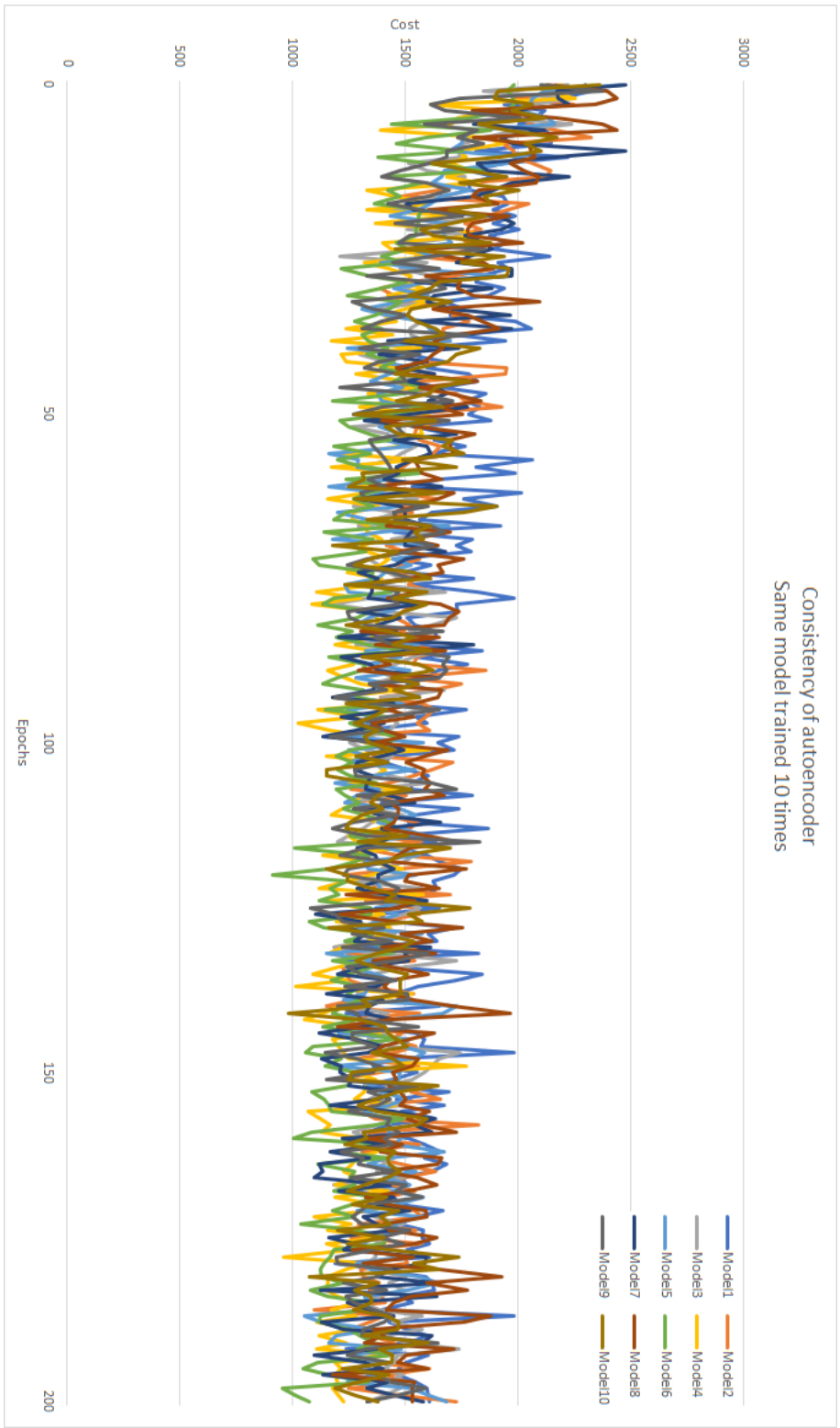
B Encoder structure

ID	INPUT	ENCODER												
1	Input	conv	pool	conv	pool	reshape								
	128x128x3	128x128x10	64x64x10	64x64x10	32x32x10	10240								
2	Input	conv	pool	conv	pool	reshape	FC							
	128x128x3	128x128x10	64x64x10	64x64x10	32x32x10	10240	5120							
3	Input	conv	pool	conv	pool	reshape	FC	FC						
	128x128x3	128x128x10	64x64x10	64x64x10	32x32x10	10240	5120	2560						
4	Input	conv	pool	conv	pool	reshape	FC	FC	FC					
	128x128x3	128x128x10	64x64x10	64x64x10	32x32x10	10240	5120	2560	1280					
5	Input	conv	pool	conv	pool	conv	pool	reshape						
	128x128x3	128x128x10	64x64x10	64x64x10	32x32x10	32x32x10	16x16x10	2560						
6	Input	conv	pool	conv	pool	conv	pool	reshape	FC					
	128x128x3	128x128x10	64x64x10	64x64x10	32x32x10	32x32x10	16x16x10	2560	1280					
7	Input	conv	pool	conv	pool	conv	pool	reshape	FC	FC				
	128x128x3	128x128x10	64x64x10	64x64x10	32x32x10	32x32x10	16x16x10	2560	1280	640				
8	Input	conv	pool	conv	pool	conv	pool	reshape	FC	FC	FC			
	128x128x3	128x128x32	64x64x32	64x64x32	32x32x32	32x32x32	16x16x32	8192	4096	2048	1024			
9	Input	conv	pool	conv	pool	conv	pool	conv	pool	reshape	FC			
	128x128x3	128x128x32	64x64x32	64x64x32	32x32x32	32x32x32	16x16x32	16x16x32	8x8x32	2048	1024			
10	Input	conv	conv	pool	reshape									
	128x128x3	128x128x10	128x128x10	64x64x10	40960									
11	Input	conv	conv	pool	reshape	FC								
	128x128x3	128x128x10	128x128x10	64x64x10	40960	20480								
12	Input	conv	conv	pool	conv	conv	pool	reshape						
	128x128x3	128x128x10	128x128x10	64x64x10	64x64x10	64x64x10	32x32x10	10240						
13	Input	conv	conv	pool	conv	conv	pool	reshape	FC					
	128x128x3	128x128x10	128x128x10	64x64x10	64x64x10	64x64x10	32x32x10	10240	5120					
14	Input	conv	conv	pool	conv	conv	pool	reshape	FC	FC				
	128x128x3	128x128x10	128x128x10	64x64x10	64x64x10	64x64x10	32x32x10	10240	5120	2560				
15	Input	conv	conv	pool	conv	conv	pool	reshape	FC	FC	FC			
	128x128x3	128x128x10	128x128x10	64x64x10	64x64x10	64x64x10	32x32x10	10240	5120	2560	1280			
16	Input	conv	conv	pool	conv	conv	pool	conv	conv	pool	reshape			
	128x128x3	128x128x10	128x128x10	64x64x10	64x64x10	64x64x10	32x32x10	32x32x10	32x32x10	16x16x10	2560			
17	Input	conv	conv	pool	conv	conv	pool	conv	conv	pool	reshape	FC		
	128x128x3	128x128x10	128x128x10	64x64x10	64x64x10	64x64x10	32x32x10	32x32x10	32x32x10	16x16x10	2560	1280		
18	Input	conv	conv	pool	conv	conv	pool	conv	conv	pool	reshape	FC	FC	
	128x128x3	128x128x10	128x128x10	64x64x10	64x64x10	64x64x10	32x32x10	32x32x10	32x32x10	16x16x10	2560	1280	640	
19	Input	conv	conv	pool	conv	conv	pool	conv	conv	pool	reshape	FC	FC	FC
	128x128x3	128x128x10	128x128x10	64x64x10	64x64x10	64x64x10	32x32x10	32x32x10	32x32x10	16x16x10	2560	1280	640	320

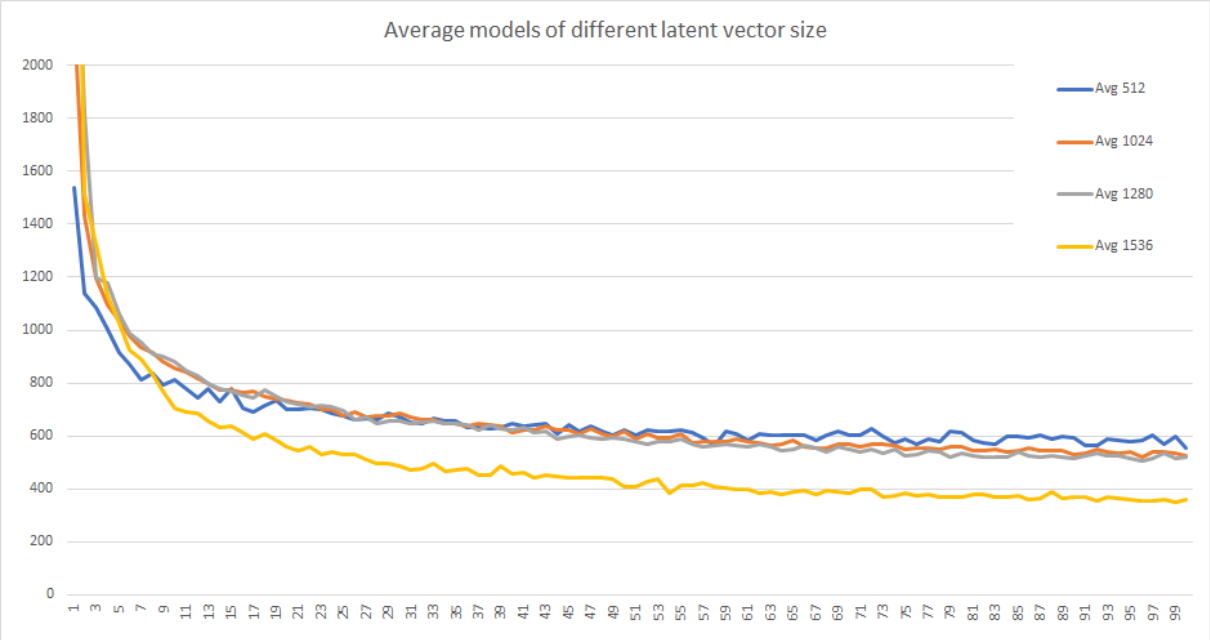
C Autoencoder 48 models

ID	MODEL	LEARNING_RATE	BATCH_SIZE	N_FILTERS	BEST_COST	LAST_LAYER_SIZE	EPOCHS	TIME(H:M:S)
1	Model-1	0.0001	64	12	281	1536	100	04:23:27
2	Model-1	0.0001	128	12	295	1536	100	04:11:10
3	Model-1	0.0001	64	8	304	1024	100	03:34:54
4	Model-1	0.0001	64	10	307	1280	100	04:05:18
5	Model-1	0.0001	128	10	324	1280	100	03:43:36
6	Model-1	0.0001	128	8	367	1024	100	03:33:51
7	Model-1	0.001	128	10	736	1280	100	03:54:11
8	Model-1	0.001	128	8	751	1024	100	03:28:15
9	Model-1	0.01	128	8	1835	1024	100	03:23:32
10	Model-1	0.01	128	10	25152	1280	100	03:39:06
11	Model-2	0.001	32	40	362	1280	100	03:44:33
12	Model-2	0.0001	32	40	375	1280	100	03:50:06
13	Model-2	0.001	32	32	376	1024	100	03:27:40
14	Model-2	0.0001	32	32	402	1024	100	03:59:36
15	Model-2	0.001	64	32	405	1024	100	03:06:12
16	Model-2	0.001	32	16	426	512	100	03:11:01
17	Model-2	0.0001	64	40	439	1280	100	03:39:12
18	Model-2	0.001	128	40	446	1280	100	03:41:14
19	Model-2	0.0001	32	16	457	512	100	03:16:38
20	Model-2	0.001	128	32	458	1024	100	03:25:05
21	Model-2	0.001	64	40	467	1280	100	02:36:31
22	Model-2	0.0001	64	32	471	1024	100	03:25:13
23	Model-2	0.0001	64	16	505	512	100	03:11:13
24	Model-2	0.0001	128	32	509	1024	100	03:58:53
25	Model-2	0.001	64	16	517	512	100	03:11:48
26	Model-2	0.0001	128	40	537	1280	100	03:40:02
27	Model-2	0.001	128	16	555	512	100	03:10:37
28	Model-2	0.0001	128	16	601	512	100	03:08:24
29	Model-3	0.0001	64	12	319	1536	100	00:20:44
30	Model-3	0.0001	64	10	350	1280	100	04:19:51
31	Model-3	0.0001	128	12	362	1536	100	04:24:48
32	Model-3	0.0001	128	10	377	1280	100	04:07:14
33	Model-3	0.0001	128	8	383	1024	100	03:56:47
34	Model-3	0.0001	64	8	445	1024	100	03:56:59
35	Model-3	0.001	128	10	955	1280	100	03:50:35
36	Model-3	0.001	128	8	1137	1024	100	04:00:31
37	Model-3	0.01	128	10	1922	1280	100	02:29:00
38	Model-3	0.01	128	8	2723	1024	100	03:10:10
39	Model-4	0.0001	64	10	363	1280	100	02:12:51
40	Model-4	0.0001	128	12	368	1536	100	03:46:49
41	Model-4	0.001	128	10	384	1280	100	03:34:48
42	Model-4	0.0001	64	12	386	1536	100	03:29:21
43	Model-4	0.0001	128	10	404	1280	100	03:43:23
44	Model-4	0.0001	64	8	442	1024	100	03:35:22
45	Model-4	0.001	128	8	452	1024	100	03:28:23
46	Model-4	0.0001	128	8	456	1024	100	03:32:02
47	Model-4	0.01	128	8	2210	1024	100	01:46:48
48	Model-4	0.01	128	10	2282	1280	100	01:21:01

D Consistency of autoencoder



E Average models of different latent vector size



Latent size	Average is calculated using model ID from Appendix C
512	16, 19, 23, 25, 27, 28
1024	3, 6, 8, 13, 14, 15, 20, 22, 24, 33, 34, 36, 44, 45, 46
1280	4, 5, 7, 11, 12, 17, 18, 21, 26, 30, 32, 35, 39, 41, 43
1536	1, 2, 29, 31, 40, 42

Model ID 9, 10, 30, 32, 37, 38 and 47 were discarded due to poor performance.