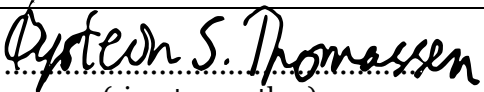




FACULTY OF SCIENCE AND TECHNOLOGY

MASTER THESIS

Study programme / specialisation: Computer Science – Reliable and Secure Systems	The spring semester, 2022 Open
Author: Øystein Sunde Thomassen	 (signature author)
Course coordinator: Ketil Oppedal Supervisor(s): Ketil Oppedal and Álvaro Fernández Quílez	
Thesis title: Improving Prostate Cancer Diagnostic Pathway with Transformers and a Generative Self-Supervised Approach	
Credits (ECTS): 30	
Keywords: Masked Autoencoder, Deep Learning, Classification, Biomedical Image, Prostate Cancer, Self-Supervised Learning, Transfer Learning	Pages: 86 + ThesisCode.zip: Stavanger, 15 June 2022 date/year



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

Improving Prostate Cancer Diagnostic Pathway with Transformers and a Generative Self-Supervised Approach

Master's Thesis in Computer Science
by

Øystein Sunde Thomassen

Internal Supervisors

Ketil Oppedal

Alvaro Fernandez Quilez

June 15, 2022

Abstract

Prostate cancer is the second most occurring cancer and the sixth leading cause of cancer among men worldwide. The number of cases are expected to increase due to population growth and increase in expected lifetime. The screening test for prostate cancer is not without risk and there is chance of over-diagnosis with the traditional screening. The magnetic resonance imaging (MRI) examination is an essential tool and a comfortable method for diagnosing cancer. If some of the screening methods can be replaced by accurate MRI examinations it will be more comfortable for the patients.

This thesis will explore how masked autoencoders with a generative self-supervised approach can improve diagnosis of prostate cancer from MRI images. Computer vision is used in many ways in the field of medical imaging. Hopefully computer vision based on deep learning and transformers can improve the field of medical imaging. However, the state of the art deep learning models require pre-training on huge amounts of unlabeled data and fine-tuning on large amounts of data. This thesis investigates if a masked vision transformer can learn to predict prostate cancer lesions on a limited amount of data. The final results suggest that a simple model of small size is not sufficient to for prediction prostate cancer.

Acknowledgements

This thesis marks the end of my Master's Degree in Computer Science at the Department of Electrical Engineering and Computer Science at the University of Stavanger.

I want to thank my supervisors Ketil Oppedal and Álvaro Fernández Quíles for letting me investigate this exciting topic in my last semester. Special thanks to Álvaro for his guidance on all parts of the thesis. Providing vital insight on theory and practice in machine learning and the medical aspect. The thesis could not have been accomplished without his motivation and help during the implementation of the model.

I would like to give a special thanks to my fellow students for making it a tremendous two years at university, even with the covid pandemic. The cooperation and sharing of knowledge made it possible. Thank you for the good times solving projects in the ISI room and especially for the good moments during breaks and after school.

Contents

Abstract	iii
Acknowledgements	v
Abbreviations	xi
1 Introduction	1
1.1 Motivation	1
1.2 Problem Definition	2
1.2.1 Objectives	2
1.3 Outline	3
2 Medical Background	5
2.1 Prostate Cancer	5
2.2 Prostate Cancer Examination Methods	6
2.2.1 Prostate-Specific Antigen Test	7
2.2.2 Digital Rectal Exam	7
2.2.3 Prostate Biopsy	7
2.2.4 Magnetic Resonance Imaging	9
2.2.5 Gleason Grade Group	9
3 Technical Background and Theory	11
3.1 Neural Network	11
3.1.1 Backpropagation and Gradients	13
3.1.2 Optimizer algorithms	14
3.2 Activation Function	17
3.2.1 Sigmoid	17
3.2.2 Softmax	18
3.2.3 GELU	18
3.3 Residual Learning	20
3.4 Supervised, Unsupervised, Self-Supervised and Semi-supervised Learning .	21
3.5 Autoencoder	21
3.5.1 Regularized Autoencoders	23
3.6 Transformers	24
3.6.1 Transformer Architecture	25
3.6.2 Vision Transformers	27

3.7	Masked Autoencoder for Computer Vision	30
3.7.1	Masking in Vision vs. Text	31
3.8	Metrics and Loss Function	32
3.8.1	Loss Functions	32
3.8.2	Categorical Crossentropy	33
3.8.3	Metrics	34
3.9	Software and Packages	36
3.9.1	TensorFlow	37
3.9.2	Keras	37
3.9.3	Numerical Python	37
3.9.4	OpenCV - cv2	37
4	Data Set and Image Pre-Processing	39
4.1	Data Set	39
4.2	Data Filtering	40
4.2.1	Downloading Data	40
4.2.2	T2-Weigthed Ground Truth	40
4.3	Image Pre-Processing	42
4.3.1	Image Resizing	42
4.3.2	Grey Scale Filtering and Image Normalization	43
4.3.3	Organizing Data Sets	44
5	Solution Approach and Configuration	45
5.1	Model Architecture	45
5.1.1	Patches	46
5.1.2	Patch Encoding and Masking	46
5.1.3	Autoencoder Architecture	48
5.1.4	Downstream Model Architecture	50
5.2	Configuration	51
5.2.1	Width and depth of the network	51
5.3	Augmentation	53
5.4	Hyperparameters	53
5.5	Limitations	55
6	Experimental Evaluation	57
6.1	Experimental Setup	57
6.2	Experimental results	58
6.2.1	Result of Pre-Training	58
6.2.2	Results from Downstream Model	63
6.2.3	Cropping the Image Around Prostate Gland	67
7	Discussion	69
7.1	Approach and Model	69
7.2	Image Pre-Processing and Augmentation	70
7.3	Results and Classification	70
7.4	Limitations	72
7.4.1	Model size	72
7.4.2	Data Set Size	72

7.4.3 Other Approaches with 2.5D and 3D MAE	72
8 Conclusion and Future Direction	75
8.1 Future Directions	76
List of Figures	76
List of Tables	79
Bibliography	81

Abbreviations

2D	Two-Dimensional
3D	Three-Dimensional
AFS	Anterior Fibromuscular Stroma
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Network
CZ	Central Zone
DCE	Dynamic Contrast Enhancement
DW	Diffusion Weighted
DRE	Digital Rectal Examination
FN	False Negative
FP	False Positive
GELU	Gaussian Error Linear Units
GGG	Gleason Grade Group
MAE	Masked Autoencoder
ML	Machine Learning
MLP	Multilayer Perception
MRI	Magnetic Resonance Imageing
MSE	Mean Square Error
NLP	Natural Language Processing
NN	Neural Network
PD-W	Proton Density-Weighed
PSA	Prostate-Specific Antigen
PZ	Peripheral Zone
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent

T2W	T2-Weighted
TN	True Negative
TP	True Positive
TZ	Transition Zone
TRUS	Transrectal Ultrasound
ViT	Vision Transformers
ViViT	Video Vision Transformers

Chapter 1

Introduction

1.1 Motivation

Prostate cancer is the second most common type among men and the fourth most common among both sexes in 2020 [1]. In 2020 it was estimated a total of 1 414 259 new cases of prostate cancer and 375 307 deaths worldwide. Prostate cancer is a severe disease for the male population, and it is expected that the worldwide prostate cancer will increase to around 2.3 million new cases and 740 000 deaths by 2040 due to the growth and aging of the population [2]

Because prostate cancer is a widely spread cancer around the world, it is normal to perform screening on men in their fifties. The goal is to detect prostate cancer in an early stage to offer treatment before it turns life-threatening. Most medical organizations encourage men above fifty to discuss screening and the general pros and cons with their general practitioner (GP). There are several screening tests performed, which include digital rectal examination (DRE), prostate-specific antigen (PSA) test, and transrectal ultrasound (TRUS)-guided biopsy [3]. If any of the screening tests detect any abnormalities, the GP may refer to additional tests like magnetic resonance imaging (MRI) or biopsy to determine if cancer is present.

The current screening tests and biopsies are not free of complication and may result in harms that can be considered minor to major [3]. Common minor harms from screening include bleeding, bruising and anxiety. Common major harms include over-diagnosis and over-treatment, infection, and erectile dysfunction. MRI does not expose patients to the

complications that can be experienced from the normal screening methods or a biopsy. Recent studies show that risk assessment using MRI and MRI-targeted biopsy for men with suspicion of prostate cancer is found to be superior to the TRUS-guided biopsy [4]. Increased use of MRI has made the examination more comfortable and resulted in an improved prostate gland examination and detection of malignant tumors. MRI can also be more accurate in determining the extent and stage of the tumor compared to traditional screening [5].

Radiologists interpret MRIs, and the results are as good as the radiologist that view the MRI. The results and diagnosis may vary depending on the experience of the radiologist who interprets the image. Today there are a lot of algorithms involved to help radiologists. Software help to make the best contrast for viewing the MRI, and segmentation can help mark out areas of special interest. The next step is to introduce computer vision in MRI for prostate cancer diagnosis to help improve lesion localization and classification. Machine learning (ML) might help classify patients with significant prostate cancer that need the attention of specialists and the insignificant cases where no further treatment is required. ML might help reduce over-diagnosing and over-treatment and potentially make biopsy a redundant test.

1.2 Problem Definition

This thesis aims to classify prostate cancer tumors between those that are malignant and those that are benign by use of T2 weighted MRI of the prostate. The approach will be a simple, generative self-supervised approach, where the model is comparing reconstructed images from an autoencoder with the original input. Due to the limited amount of data, the first stage will be to perform self-supervised learning on the data set. In the second stage, transfer learning will be applied with labeled data to train the model to distinguish between malignant and benign tumors.

1.2.1 Objectives

- Implement a model for masked autoencoder to perform self supervised-training of the neural network. The goal is for the encoder to learn important features from T2-weighted MRI.

- Apply transfer learning on the encoder part of the model. See if the model can distinguish between clinically significant and insignificant lesions.

1.3 Outline

The thesis is structured into the following chapters, where the outline is presented in the following description

- The first chapter gives a brief introduction and the motivation behind the thesis.
- Chapter two gives a short description of the medical background which is relevant for understanding the motivation behind the subject and why machine learning is relevant for medical images.
- Chapter three gives the technical background and contains most of the important background theory related to the implementation of the model.
- Chapter four describes the data set and the image pre-processing performed on the data set.
- Chapter five describes the solution approach and configuration of the model. How was the model implemented and what are the limitations.
- Chapter six introduces the experimental setup and results of the model
- Chapter seven presents a discussion of the results and the limitations of the thesis.
- Chapter eight is the last one and presents the conclusion and further direction.

Chapter 2

Medical Background

This chapter will present the medical background, terminologies and concepts necessary to understand the motivation behind the thesis and the goal of the thesis.

2.1 Prostate Cancer

Prostate cancer is the second most common type of cancer among men and the fourth most commonly occurring cancer among both sexes worldwide. In 2020 it was estimated a total of 1 414 259 new cases of prostate cancer and 375 307 deaths worldwide in 2020 [1]. Prostate cancer accounts for 7.3% of all incidents worldwide and 3.8% of all mortalities related to cancer. The number is estimated to increase to approximately 2.3 million new cases by 2040 due to population growth and increased life expectancy[2].

Normal functional cells grow and divide to produce new cells in all living organisms. When normal cells die, the human body disposes of the cells. The defining feature of cancer is the rapid creation of abnormal cells that grow beyond the boundaries of normal cells and spread to other parts of the body. In other words, cancer is the result of normal cells that become abnormal and continue to live even if they are obsolete to the human body. Damaged cells continue to grow and form a tumor. A tumor can be benign or malignant. A benign tumor is an abnormal collection of cells but noncancerous. It can develop anywhere on or within the body when cells multiply more than they should or continue to live when they are abnormal. A benign tumor grows slowly, has even and defined borders, and doesn't spread to other parts of the body. Many benign tumors are

not dangerous and don't require any treatment. Malignant tumors are cancerous. They can grow quickly and have irregular borders, which again do not confine them to the boundaries of normal cells. Malignant tumors often invade surrounding tissue and can spread to other parts of the body.

The prostate is a male sex organ for reproductive anatomy located below the bladder. It is a small and soft organ with the average size of a walnut. There are four basic zones of the prostate which are relevant for prostate cancer, see figure 2.1. These are the peripheral zone (PZ), the central zone (CZ), the transition zone (TZ), and the anterior fibromuscular stroma (AFS). Most prostate tumors are found in the peripheral zone, where 70-75% of the tumors arise [6]. In the transition zone, there are around 20-25%, while only 10% arise in the central zone. It is very rare for tumors to arise in the fibromuscular stroma.

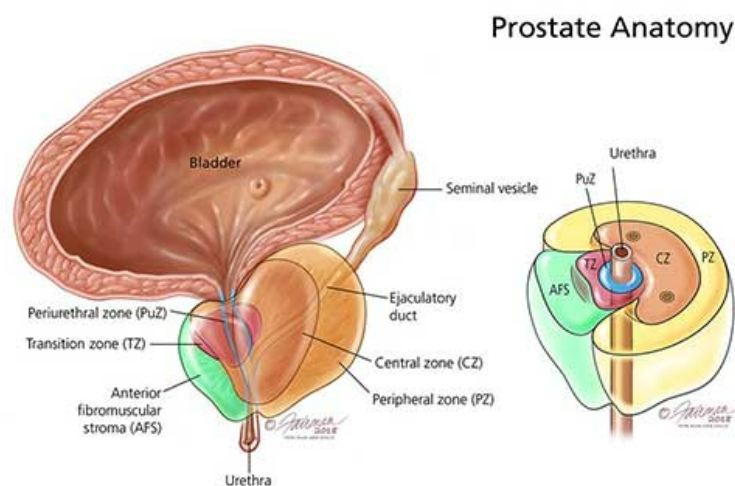


Figure 2.1: Prostate Anatomy [7]

2.2 Prostate Cancer Examination Methods

There is no definitive method for detecting prostate cancer. The following methods may be used in combination to detect prostate cancer if a patient is experiencing symptoms or as part of a screening process. For a screening process, the patient and the GP should discuss the benefits and the drawbacks, including the risks for the different methods.

2.2.1 Prostate-Specific Antigen Test

Prostate-specific antigen (PSA) test aims to detect prostate cancer in an early stage by taking a blood sample to determine the amount of PSA in the patient's blood. PSA is a substance that is excreted in small amounts from the prostate gland and released into the semen and bloodstream, and is measured in nanograms per milliliter (ng/mL) units. The original limit for normal PSA was 4 ng/mL based on an analysis of levels in 860 men and women without prostate cancer [8]. Prostate size increase with age and thus, the concentration also rises with age, which suggests new ranges for normal PSA should be made [9]. Suggested values are shown in table 2.1.

40 to 49 years	0 to 2.5 ng/mL
50 to 59 years	0 to 3.5 ng/mL
60 to 69 years	0 to 4.5 ng/mL
70 to 79 years	0 to 6.5 ng/mL

Table 2.1: Suggested age specific normal values of PSA [9]

2.2.2 Digital Rectal Exam

A digital rectal exam (DRE) is an early-stage prostate cancer screening test performed when patients show symptoms of prostate cancer. It is most commonly done after a PSA test. DRE can, in some cases, detect prostate cancer in men with normal PSA levels, making it a viable extra method if PSA levels are normal with patients showing symptoms. During a DRE, the doctor inserts a lubricated gloved finger into the rectum of the patient and feels for abnormalities, see figure 2.2. The doctor will feel the size of the prostate gland and for bumps, soft or hard spots, or other abnormalities.

2.2.3 Prostate Biopsy

If the GP suspects prostate cancer from either PSA or DRE, then the patient is referred to a urologist for a biopsy of the prostate. The procedure is known as the transrectal ultrasound (TRUS) guided biopsy. During TRUS, the doctor will guide an ultrasonic probe into the rectum, see figure 2.3. The ultrasonic probe produces images of the prostate to help the doctor guide where to take the samples from. A separate needle is inserted next to the probe, and it is usually taken 10 to 12 small samples of tissue from

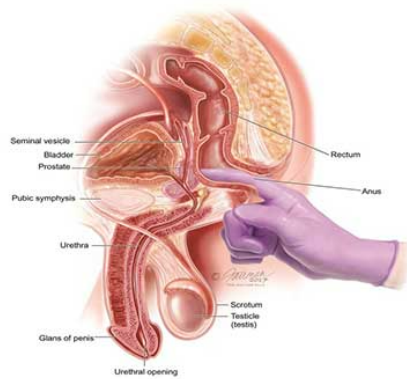


Figure 2.2: Digital rectal exam [7]

different areas of the prostate [10]. TRUS biopsy is still very much non-targeted and directed towards the peripheral zone. With 70-75% of prostate cancer finding place in the peripheral zone, tumors arising in other zones may be missed. TRUS has a negative predictive value of 70-80%, meaning that up to 20-30% of patients with a negative biopsy may have prostate cancer [11]. The biopsy is not without risk of complications [3]. Minor harms may include bleeding and bruising, while major harms may include infection and erectile dysfunction.

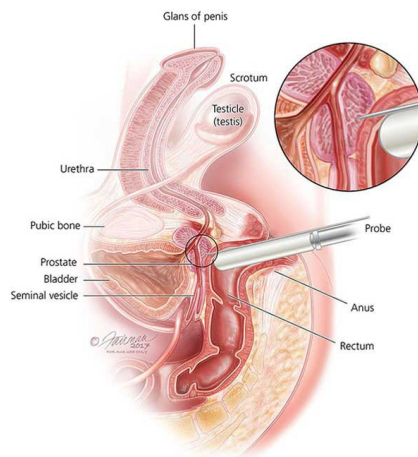


Figure 2.3: Transrectal ultrasound guided biopsy [7]

2.2.4 Magnetic Resonance Imaging

Recent advancements in magnetic resonance imaging (MRI) have led to including MRI in prostate cancer detection. MRI might be used to locate abnormalities in the prostate and help the urologist with areas to collect tissue samples from during TRUS. From MRI images, the radiologist can make decisions on whether further investigations are necessary at the time. Today, the radiologist predicts the probability of a lesion to be clinically significant (malignant) based on findings from a multiparametric MRI. The scoring is based on T2-weighted (T2W), diffusion-weighted imaging/apparent diffusion coefficient (DWI/ADC), and the dynamic contrast enhancement (DCE) sequence. The images are classified according to the Prostate Imaging Reporting and Data System (PI-RADS) scoring system. The latest version, PI-RADS v2.1, was released in 2019 [12]. PI-RADS gives scores dependent on the zone the lesion is located. In the previous version, the assessment criteria were provided for PZ and TZ, which accounts for most cases of prostate cancer. Lesion located in the PZ is mainly determined by DWI/ADC, while lesion located in TZ is mainly determined by T2W. In v2.1, there are updated specifications for CZ and AFMS. For CZ and AFMS, it is an assessment based on T2W and/or DWI/ADC. DCE has generally not had a very big role in scoring prostate cancer. It may assist in the detection of prostate cancer in PZ and TZ. MRI before biopsy can reduce the number of unnecessary biopsies and give better staging accuracy and help determine the need for further treatment [11].

2.2.5 Gleason Grade Group

Donald Floyd Gleason developed and introduced the first pathological-based scoring system for prostate cancer in 1966 [13]. The system was based on samples from 270 patients, and originally, there were a total of nine grades. Further development through follow-up studies reduced the number to five grading categories based on the cell patterns in the tissue. The Gleason score is the sum of the two most widespread patterns. This gives a maximum score of ten. In 2014, the International Society of Urological Pathology (ISUP) introduced an updated grading system with five Gleason grade groups (GGG) [14]. GGG was introduced to simplify the prostate cancer grading prognosis. The GGG groups and the corresponding Gleason scores are shown in table 2.2

GGG	Gleason Score	Clinically Significant
Grade Group 1	Gleason Score ≤ 6	False
Grade Group 2	Gleason Score 7 (3+4)	True
Grade Group 3	Gleason Score 7 (4+3) 6	True
Grade Group 4	Gleason Score 8	True
Grade Group 5	Gleason Score 9 and 10	True

Table 2.2: Gleason score and ISUP-Grading [14]

We see from the table that Gleason score of 6 or GGG of 1 is the lower bound for being clinically significant. GGG 1 has features similar to normal tissue samples. The cells have even and defined borders. GGG 4 and 5 have patterns with features indicating the presence of aggressive cancer cells. The borders are irregular, and the risk of spreading to other parts of the body is present.

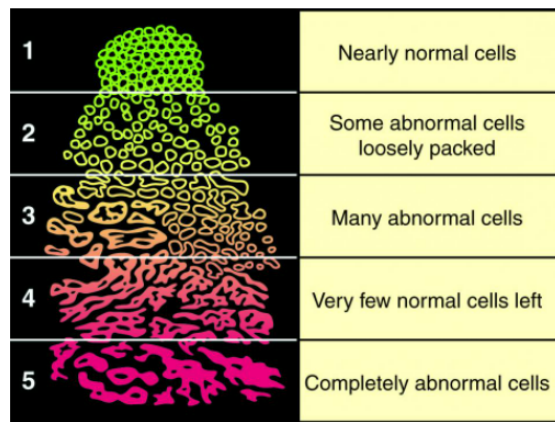


Figure 2.4: Illustration of Gleason score for different cancer cell patterns [15]

Chapter 3

Technical Background and Theory

This chapter will present the technical background and theory which applies to the thesis and the model implemented for prostate cancer diagnostics.

3.1 Neural Network

Neural network (NN), in the sense of machine learning, is a network composed of artificial neurons or nodes. The concept of NN for computers was created in 1943 when Warren Sturgis McCulloch and Walter Pitts created a computational model for neural networks [16]. The model was based on input to a neuron and an activation function to determine the output. The model was called logical threshold based on an activation function of threshold. NN is inspired by the electrical signals between neurons in the human brain nervous system, hence the name neural network. The backbone for the deep learning algorithms used today is the deep feedforward network, also known as multilayer perceptron (MLP). For classification, the feedforward network approximates some function f^* by mapping an input x to a category or label y [17, p.163-166]. A NN contains several neurons in a network of layers that react to an input before transmitting an output. Figure 3.1 illustrates a small feedforward NN with three input values ($\mathbf{x} = [x_1, x_2, x_3]$), one hidden layer with four neurons, and one output layer. The input value \mathbf{x} contains the initial information that is fed to the NN and which propagates through the layers to predict the output values \hat{y} .

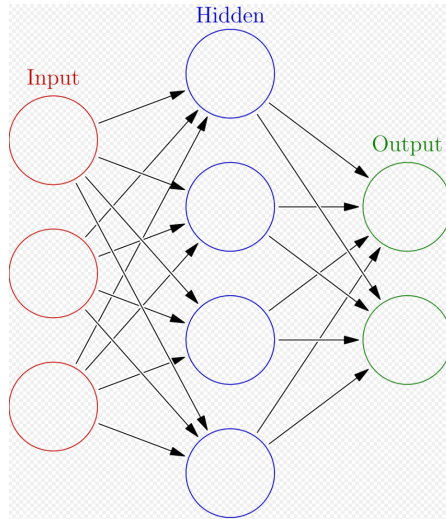


Figure 3.1: Illustration of neural network with one input layer, one hidden layer and one output layer. Each circular node represents an artificial neuron and an arrow represents a connection from the output of one artificial neuron to the input of another [18]

Figure 3.2 illustrates a neuron and the mathematical functions included to calculate the output. A neuron has n numbers of input values received through n connections to the neuron. Each connection from the input to the neuron has specific weights $w = [w_1, w_2, \dots, w_n]$, which are weights learned by the algorithm during training. It is also common to have a bias (b) as input to the neuron. The bias is independent of the input and allows the activation to be shifted to adjust the output value **NEED SOME REFERENCES**. The neuron applies the activation function f on the weighted sum of the inputs plus the bias. The output of the activation function is the input to the next layer.

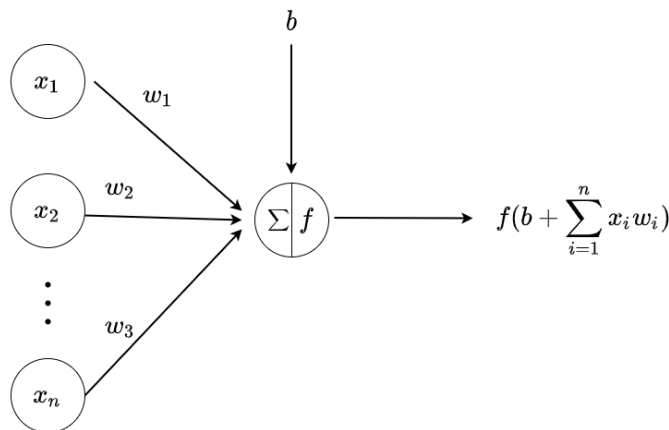


Figure 3.2: Input, activation and output of a neuron

Equation 3.1 is the calculation of the output value of the final hidden layer. Here f is the

activation function, and z corresponds to the bias plus the weighted sum of the inputs. The input, in this case, is the output of the previous layer.

$$\hat{y} = f(z) = f\left(b + \sum_{i=1}^n x_i w_i\right) \quad (3.1)$$

3.1.1 Backpropagation and Gradients

A feedforward NN learns by minimizing a loss value and adjusting the prediction by changing the weights for each neuron and the bias. When the neural network is initialized, the weights are set to a random value. The goal of the learning algorithm is to adjust the weights so that the predicted output is as close to, or similar to the expected value for each input. The loss function will be some function that describes the difference between the predicted output and the expected output. An example is the minimum square error function given in equation 3.2.

$$L(w) = \frac{1}{2} \sum_{k=1}^c (\hat{y}_k - y_k)^2 \quad (3.2)$$

Here \hat{y} is the predicted value, and y is the expected value. In a feedforward network, the flow is from input to output without any feedback connections. The method of backpropagation became popular after David E. Rumelhart et al. showed that the loss could be propagated back through the network to compute the gradient [19]. By utilizing the chain rule, they progressed from knowing how a change in the total input to output would affect the error, to seeing how the error is changing by affecting the states and weights in the feedforward network. The method to compute the gradient uses the chain rule for derivatives that compute the gradient of the loss function $L(w)$ with respect to the weights w as shown in equation 3.4.

$$\frac{\partial L(w)}{\partial (w_{ij})} = \frac{\partial L(w)}{\partial (x_j)} * \frac{\partial x_j}{\partial (w_{ij})} \quad (3.3)$$

Here, x_j corresponds to the j -th input, and w_{ij} is the weight for the i -th output towards the j -th input. The backpropagation learning rule is based on gradient descent.

Having the method to calculate the gradients of the loss function, the weights can be changed in the direction that reduces the error. This gives the backpropagation learning rule. The weights are initialized with random values and are changed in the direction that will reduce the error,

$$\Delta \mathbf{w} = -\epsilon \frac{\partial L}{\partial \mathbf{w}} \quad (3.4)$$

where ϵ is the learning rate which indicates the relative size of the change in weights.

3.1.2 Optimizer algorithms

Deep learning algorithms involve optimization in many contexts. One of the most important is the training as it is common to invest a long time to solve an instance of the neural network training. The time can be days to months, depending on the input and model size. Because the problem of training is both important and expensive, it is developed optimizing techniques and algorithms for solving the problem. For deep learning, the optimization problem relates to finding the parameters θ of a neural network that reduces a cost function $J(\theta)$ [17, p.267-268]. The cost function can again be written as an average over the training set as

$$J(\theta) = E_{(x,y) \sim \hat{p}_{data}} L(f(\mathbf{x}; \theta), y) = \frac{1}{m} \sum_{i=1}^m L(f(x^{(i)}; \theta), y^{(i)}) \quad (3.5)$$

where L is the loss function, f is the activation function, $f(x; \theta)$ is the predicted output when the input is x and \hat{p}_{data} is the empirical distribution defined by the training set. θ will, in our case, be the weights w and the bias b if it is included. A machine learning problem can be converted to an optimization problem by minimizing the expected loss on the training set. In this thesis, optimization is utilizing an optimizer algorithm to minimize the loss function during training. The optimizer is gradient-based and uses the gradient calculated by backpropagation to learn features that minimize the loss.

The goal of optimization is to find the global minimum. Figure 3.3 shows a function with one global minimum but several local minimums. The global minimum is the lowest

value of $f(x)$ while the local minimum is a point where $f(x)$ is lower than all neighboring points on both sides of the curve.

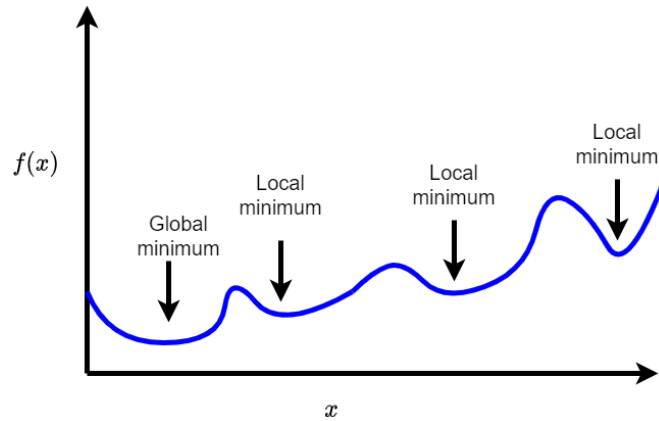


Figure 3.3: Function $f(x)$ including one global minimum and several local minimums

A neural network will have many local minimums, and finding the global minimum will be an exhaustive process during learning. During training, the optimization algorithm may find a local minimum. Finding a local minimum can be problematic if it has a high cost compared to the global minimum. However, today it is not regarded as a big problem for sufficiently large neural networks, and most local minima are regarded to be sufficient[20]. In this thesis, we will use the Adam optimizer algorithm, which can be seen as a combination of two optimizer algorithms named stochastic gradient descent (SGD) with momentum and an algorithm with an adaptive learning rate named RMSProp.

Stochastic Gradient Descent with Momentum

Stochastic gradient descent with momentum is considered a basic algorithm for optimization. The method of momentum [21] is designed to accelerate the learning when facing high curvature, small but consistent gradients, or noisy gradients. The original SGD is an iterative algorithm that starts with an initial guess for θ and updates θ as long as the stopping criteria is not met. A vital part of the SGD is decreasing the learning rate over time because the gradient estimator is based on a random sampling of m training samples that introduces noise. The algorithm is popular for machine learning but can be slow.

In SGD with momentum, an additional variable v is introduced. The algorithm accumulates an exponentially decreasing moving average of previous gradients and proceeds

in that direction [17, p.288-289]. This speeds the learning rate as the direction of the update is more towards a minimum compared to standard SGD. SGD with momentum does not require decreasing learning rate to converge. The updates during iteration is given by equation 3.6 and 3.9.

$$v \leftarrow \alpha v - \epsilon \nabla_{\theta} \left(\frac{1}{m} \sum_{i=1}^m L(f(x^{(i)}; \theta), y^{(i)}) \right) \quad (3.6)$$

$$\theta \leftarrow \theta + v \quad (3.7)$$

The parameter α determines how quickly the contribution of the previous gradient exponentially decays. In other words, the larger the α is relative to ϵ , the more the previous gradient affects the current direction.

RMSProp

RMSProp is an optimizer algorithm with an adaptive learning rate. The learning rate is adapted for each of the parameters, and the idea is to have a moving average of the magnitude of previous gradients for each weight. RRMSProp is an unpublished method proposed by Geoff Hinton in Lecture 6e of his Coursera Class [22]. An advantage of using RMSProp is the opportunity to use mini-batches. Mini-batches are a smaller batch of the training data used to update the parameters during learning.

The moving average of the squared gradient for each weight is given by

$$MeanSquare(w, t) = 0.9 * MeanSquare(w, t - 1) + 0.1 \left(\frac{\partial E}{\partial w^t} \right)^2 \quad (3.8)$$

where w corresponds to weights and t resembles time or iterations. The factors 0.9 and 0.1 is weighted factors for the average squared gradient and the current gradient.

The updates of the weights are given by

$$w \leftarrow w - \frac{\epsilon}{MeanSquare(w, t)} \Delta E(w) \quad (3.9)$$

Adam

The name "Adam" originates from the phrase "adaptive moments" and is a combination of the two previously mentioned optimizer algorithms. The momentum in Adam optimizer is included directly as an estimate of the first-order moment of the gradient. Adam includes a bias to the estimates of the first-order moments and the second-order moments, considering their initialization at the origin [17]. The algorithm and pseudo-code to describe the algorithm is presented in the original paper [23].

3.2 Activation Function

The activation function in an artificial neural network is used to transform an input value into an output value which in turn is fed to the next layer in the neural network. The input to the activation function is the weighted sum of all outputs from the previous layer (see section 3.1). The activation function can be both linear and non-linear. There is no manual for which activation function to use in a neural network, but the most commonly used activation functions are non-linear. A neural network's prediction accuracy is defined by the activation function used [24]. If there is no activation function, then the neural network works as a linear regression model where the predicted output is the same as the weighted sum of the input. For a linear activation function, the network can only learn and adapt to linear changes in the input. In the real world, there are non-linear relations, and this is why non-linear activation functions are preferred.

3.2.1 Sigmoid

The sigmoid activation function gives an output between 0 and 1. It is commonly used in machine learning because its range is $(0, 1)$, which lies in the valid range of probability. It is also referred to as the logistic sigmoid function and has an s-shaped curve, see figure 3.4. The sigmoid function saturates most of the argument. Large positive values saturate towards one and negative values towards zero. The function, therefore, becomes very flat and insensitive to small changes in the input, except around zero. The sigmoid function is continuously differentiable and given by

$$f(z) = \frac{1}{1 + e^{-z}} \quad (3.10)$$

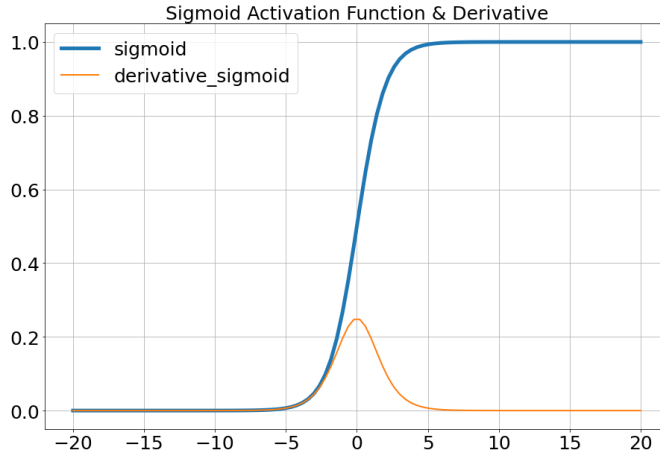


Figure 3.4: Sigmoid activation function

3.2.2 Softmax

The softmax function is a combination of several sigmoid functions. The sigmoid function returns values in the range 0 to 1 and is often used for binary classification problems. The softmax activation function can be used for multiclass classification problems. The function returns the probability for every data point of all the individual classes. The softmax function is given by

$$f(\mathbf{z})_j = \frac{1 + e^{z_j}}{\sum_{k=1}^K e^{-z_k}} \quad \text{for } j = 1, \dots, K \quad (3.11)$$

3.2.3 GELU

One of the most widely used activation functions in deep learning is the Rectified Linear Unit (ReLU) [25]. ReLU is defined as $f(x) = \max(x, 0)$. It is a straightforward function where the output is equal to the input for positive numbers and zero for negative numbers. Deep neural networks with ReLU are more easily optimized and can train several times faster than traditional activation functions like sigmoid or tanh units [26]. The Gaussian Error Linear Unit (GELU) was introduced in 2016 as an alternative to the highly popular ReLU [27]. Both functions is plotted in 3.5.

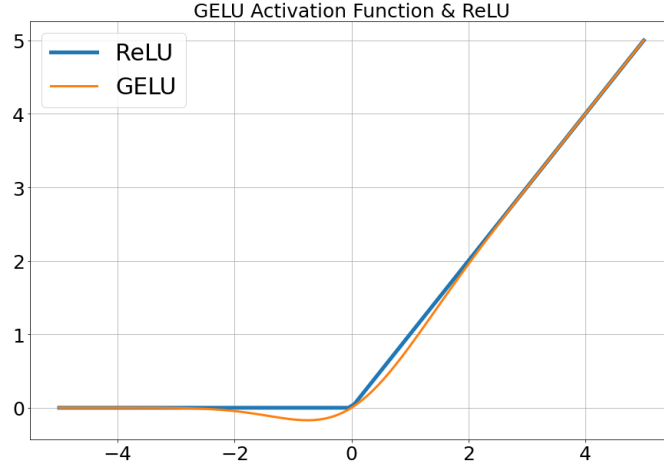


Figure 3.5: GELU and ReLU activation function

The GELU activation function is defined as

$$GELU(x) = xP(X \leq x) = x\Phi(x) = x\frac{1}{2}[1 + erf(x/\sqrt{2})] \quad (3.12)$$

Where erf is the error function. We can approximate the GELU with

$$GELU(x) \approx 0.5x(1 + \tanh[\sqrt{2/\pi}(x + 0.044715x^3)]) \quad (3.13)$$

We can see that the GELU activation function is similar to ReLU from the plot, where the difference is approximately in the range of -2 to 2. It has a negative coefficient that shifts toward positive. The interesting difference is in the derivative of the two functions. For ReLU the derivative is 0 for negative numbers and 1 for positive numbers as the derivative of 0 is 0, and the derivative of x is 1. This is also the reason for the easy optimization because the gradients are able to flow when the input to the ReLU function is positive. However, if several of the inputs are negative, the components of the network are not updated during training. The derivative of the GELU is a hyperbolic function shown in figure 3.6. It is continuous and is not equal to zero for small negative numbers. The GELU activation function was shown to be state-of-the-art in natural language processing (NLP) through bidirectional encoder representations from transformers (BERT) for language understanding [28].

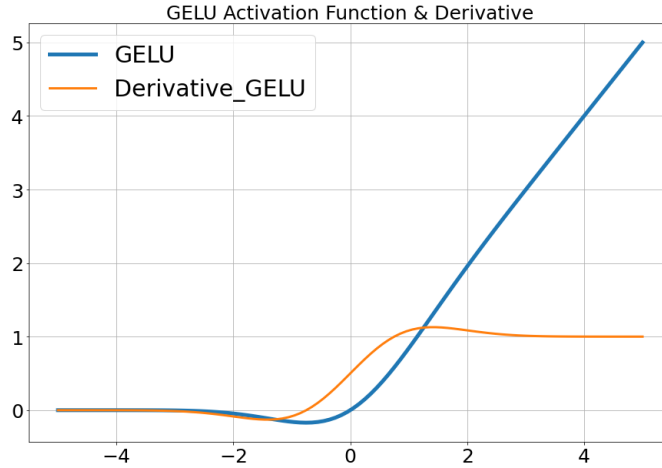


Figure 3.6: GELU and ReLU activation function

3.3 Residual Learning

Deep convolutional neural networks led to a series of breakthroughs for image classification [26, 29]. Deep networks can find features and relations on a lower and higher lever compared to a network with one or two layers [30]. Learning better networks is not as simple as just stacking more layers on top of each other. Experiments showed that the training error increased with an increasing number of training layers, which was also the reason for investigating deep residual learning [31]. Instead of letting each stack in the layer directly fit an underlying mapping, the layers fit a residual mapping. The idea is illustrated in figure 3.7

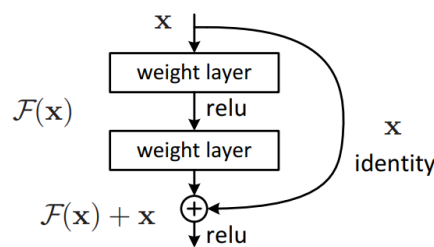


Figure 3.7: Residual learning building block. *The figure is reprinted in unaltered form from the paper written by Kaiming He et al. named "Deep residual learning for image recognition" [31]*

If the desired underlying mapping is $H(x)$, the stacked nonlinear layers fit another mapping of $F(x) := H(x) - x$. The original mapping is recast into $F(x) + x$. $F(x) + x$ can be achieved by skipping one or more layers. Residual learning can reduce the training error, and results showed that the training error for large numbers of layers was close to a smaller number of layers with residual learning [31]

3.4 Supervised, Unsupervised, Self-Supervised and Semi-supervised Learning

There are different methods of learning in machine learning. Every machine learning model needs input information to learn, but the method applied for learning depends on how structured the data is. Structuring data can be a major job, and in many cases, we are working with unstructured data.

The term supervised learning corresponds to the process where the input x has an expected output y and tries to predict the output \hat{y} equal to y . In this case, the expected output y is labeled. The machine learning model is told what it is supposed to learn. A typical example is feeding a model with many pictures of cats and dogs, and each picture has a label that says cat or dog. After the model is trained, it is supposed to predict if a new picture it has not seen is a cat or dog.

In unsupervised learning, the model is fed unstructured data. The data has no label to tell the model what it is trying to predict. The model seeks to automatically discover and learn patterns and features in the input data to produce the output. Unsupervised is often associated with clustering, grouping, and dimensionality reduction. The model learns to find similarities in the unstructured data and assigns the input data that are similar into a cluster. Self-supervised learning can be considered a subcategory of unsupervised learning. The model is fed unstructured data, but the goal is to label the data itself.

Semi-supervised learning is a combination of supervised and unsupervised learning. Generally, a machine learning model needs a huge amount of data to learn well. If most of the data is unstructured, but some of the data is labeled, then we can apply semi-supervised learning.

3.5 Autoencoder

Autoencoder is a part of deep learning where the neural network is trained to attempt to copy the input to its output. The model is designed to encode the input into a compressed yet meaningful representation and then decode it back so that the reconstructed output is as similar as possible to the original input.

The general structure of an autoencoder is simple to describe mathematically [17, p.493-502]. There is a mapping from an input x to an output r (reconstructed) through an internal representation h . An autoencoder has two components: the encoder f which maps x to h and the decoder g that maps h to r . This is illustrated in figure 3.8.

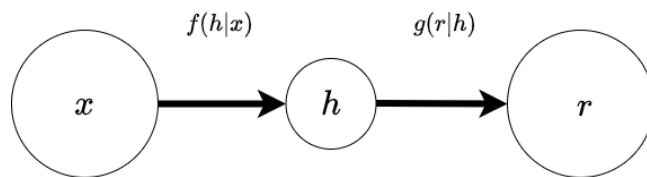


Figure 3.8: Mathematical illustration of autoencoder

A more common description of the autoencoder is illustrated in figure 3.9. An input, in this case, an image of a number, is run through the encoder, which outputs a compressed representation of the data. The compressed data is a latent representation that would not make much sense if trying to visualize it. The decoder takes the compressed representation of the output and makes a reconstruction of the input.

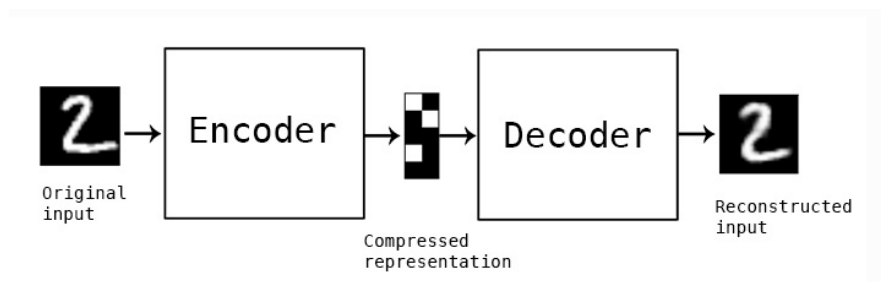


Figure 3.9: Illustration of autoencoder [32]

An autoencoder that simply learns to make a perfect mapping from x to r is not very useful for deep learning tasks. The goal is that the encoder learns to represent features and important properties to the compressed representation h . This is why the autoencoders are restricted, so the model is forced to learn useful properties to represent the input. There are several techniques for restricting the model to enforce learning of useful properties [17, p.493-502][33]. An undercomplete autoencoder h is constrained to have a smaller dimension than the input x and is the simplest form of restriction. For an under complete autoencoder the learning process is minimizing the loss function $L(\mathbf{x}, g(f(\mathbf{x})))$ by penalizing $g(f(\mathbf{x}))$ for not being similar enough to \mathbf{x} . Undercomplete autoencoders fail to learn useful properties if the encoder and decoder is given too much capacity.

3.5.1 Regularized Autoencoders

The goal is to train any architecture of autoencoder by choosing the code dimension and capacity of the encoder and decoder based on the complexity of the distribution to be modeled. Regularized autoencoders give a method for learning useful properties without introducing a small bottleneck like the undercomplete autencoder. Rather than limiting the model capacity by creating shallow encoder and decoder, options for regularization exist that will enforce the autoencoder to learn a different representation of the input. One method is the sparse autoencoder which enforces sparsity on the hidden activations [33]. The training criterion involves a sparsity penalty on the code layer h in addition to the reconstruction error. Here we will present the denoising autoencoder as an introduction to the masked autoencoder used in the thesis.

Denoising Autoencoders

The denoising autoencoder receives a corrupted version of the input and is trained to predict the original, uncorrupted version of the input. The computational graph of a denoising autoencoder is shown in figure 3.10. A training example is sampled from \mathbf{x} and run through a corruption process $C(\tilde{\mathbf{x}}|x)$. The corrupted version is the new training example as input to the model. The model will minimize the loss L by comparing the output $g(h|f(\tilde{\mathbf{x}}))$ and the original input x .

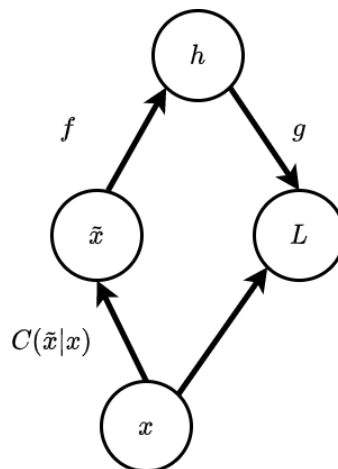


Figure 3.10: Computational graph for denoising autoencoder

After the successful approach for training deep belief networks with more than one or two layers [34], Vincent et al. proposed a denoising autoencoder in 2008 for extracting

and composing robust features [35]. The success of deep belief network was based on an unsupervised training criterion to perform a layer-by-layer initialization. Each layer is first trained to produce a higher-level representation of the patterns based on the result received from the previous layer. The results proved better than random initialization of the layers. The denoising autoencoder was developed to investigate what is a good representation for initializing deep architecture. Some information of the input needs to be presented while still being constrained in some form. The idea of partially destroyed input came from humans being able to recognize partially corrupted images. A good representation should be able to capture dependencies from a combination of many inputs and hence reconstruct a partially corrupted image. The corruption used by Vincent et al. was to set a certain number of the input pixels to zero. The empirical results support that unsupervised initialization of layers with denoising helps capture important features, and in turn, helps transfer learning to new tasks like supervised classification. The corruption process can involve many techniques like random dropping part of the image, randomly replacing part of the image with other images, color distortion, blurring, and more [36].

3.6 Transformers

A transformer is a deep learning model that adopts the mechanism of self-attention, differentially weighting the significance of each part of the input data. The concept of transformers was introduced by a team at google brain in 2017 [37] and the greatest success is probably in natural language processing (NLP) with the implementation of BERT [28]. The goal of the transformer architecture was to eliminate the issue of sequential computation. For CNN the number of operations required to relate signals from two arbitrary input or output positions grows with the distance between two positions. This makes it harder to learn dependencies between distant positions [38]. Self-attention is a mechanism for relating different positions of a single sequence in order to compute a representation of the sequence. In transformers, the sequential computation is reduced to a constant number of operations [37]. In other words, transformers process the entire input at once, and the attention mechanism provides context for any position in the input sequence. For transformers in natural language processing, each word in the input is a token, and each token is embedded to a vector of numbers to represent the word and each token has positional embedding to represent the position in the text.

3.6.1 Transformer Architecture

The original transformer architecture is given in the article, "Attention Is All You Need" [37]. The transformer model architecture is shown in figure 3.11. It is built with an encoder and a decoder structure where the encoder maps an input sequence of symbol representation (x_1, \dots, x_n) to a sequence of continuous representation $\mathbf{z} = (z_1, \dots, z_n)$. The decoder then generates an output sequence (y_1, \dots, y_n) of symbols, one element at a time. The figure show that the decoder takes the output of the encoder as additional input after the first operation. The transformer has a fixed self-attention and point-wise architecture for the encoder and decoder that are stacked in fully connected layers.

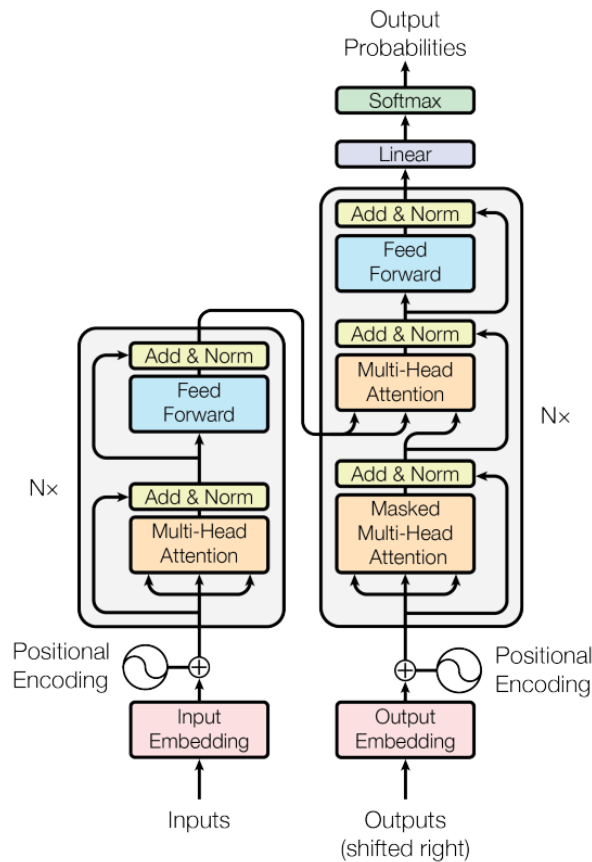


Figure 3.11: Illustration of the original transformer architecture. *The figure is reprinted in unaltered form from the original paper written by Ashish Vaswani et al. named "Attention is all you need" [37]*

Encoder and Decoder Stack

The figure shows that the architecture of the encoder and decoder is similar. The encoder consists of two sublayers, and the decoder consists of three sublayers. For the original transformer, both the encoder and decoder were composed of a stack of $N = 6$ layers. Each layer in the encoder is identical, and the same goes for the decoder. The first layer in the encoder is a multi-head attention, and the second layer is a position-wise, fully connected feed-forward network. Each of the two sublayers has a residual connection before normalization. The output of each layer is $LayerNorm(x + Sublayer(x))$. The decoder has three sublayers, where there is one additional multi-head attention taking the output of the encoder in addition to the first multi-head attention.

Multi-Head Attention

Multi-head attention is several attention layers running in parallel. The attention module splits the input parameters N -ways and passes each split independently through a separate head. All the calculations in each head are then concatenated together to produce the final attention. The multi-head attention gives the transformer greater power to encode multiple relationships for each word in the input. Figure 3.12 shows an illustration of multi-head attention with h heads.

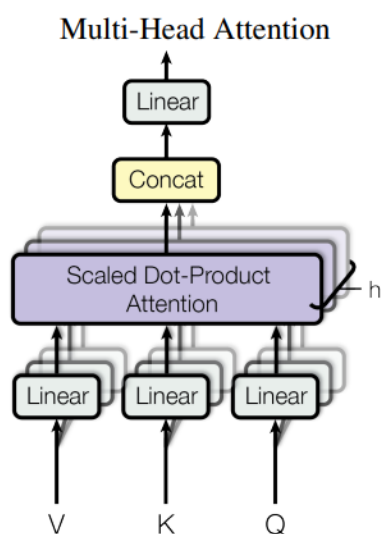


Figure 3.12: Multi-head attention with h heads. *The figure is reprinted in unaltered form from the original paper written by Ashish Vaswani et al. named "Attention is all you need" [37]*

The illustration for multi-head attention is for natural language processing where the input is a set of queries packed into a matrix, Q , and the keys and value are represented by the matrices K and V . This is split into h heads.

Positional Encoding

Since transformers do not have recurrence or convolution, there is a need for some information about the position of the input relative to each other. Positional encoding was implemented as an addition to the input embedding. Every word or token has an embedding which is a vector of numbers. In addition, every token has a positional embedding which is a vector of the same length as the token embedding. The original transformer used sine and cosine functions of different frequencies for the positional encoding of tokens. It was also experienced with learned positional embedding, but they gave similar results. The result is the sum of the input embedding and the positional embedding. An illustration of positional embedding is shown in figure

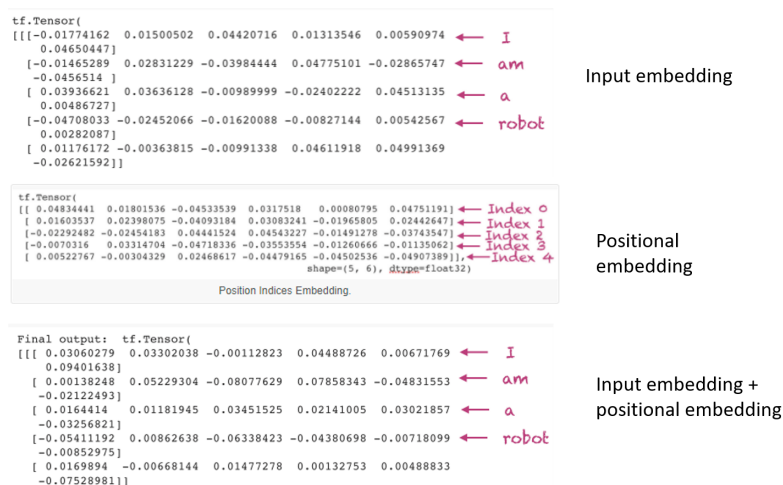


Figure 3.13: Positional Embedding

3.6.2 Vision Transformers

Transformer became the go-to model in natural language processing, but for computer vision, CNN has been the dominant architecture for machine learning models. In 2020 transformers were adapted to computer vision in the paper "An image is worth 16 x 16 words" [39]. Inspired by the scaling success of transformers in NLP, the goal was to apply

the same architecture for images with as few as possible modifications. The resulting vision transformer (ViT) architecture is shown in figure 3.14.

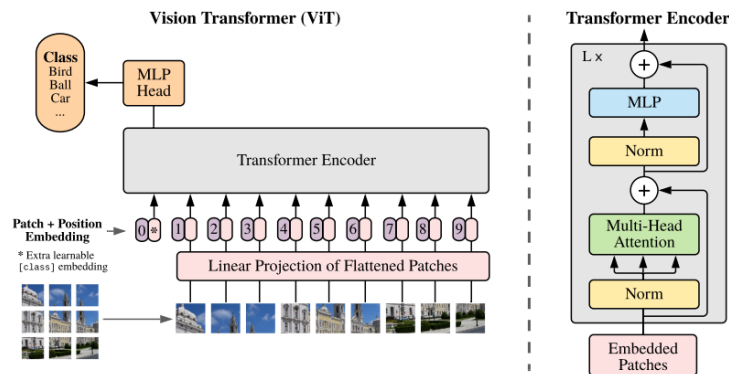


Figure 3.14: Visual transformer architecture. *The figure is reprinted in unaltered form from the original paper written by Alexey Dosovitskiy et al. named "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" [39]*

The ViT only has the encoder, and the architecture is very similar to that of the original transformer. It consists of two sublayers implemented with residual learning, where the first is the multi-head attention and the second is the multi-layer perception (MLP). MLP is a fully connected feed-forward network. For ViT, the MLP has two layers with GELU activation function.

The standard Transformer receives input as a 1D sequence of token embeddings. To apply the same architecture on 2D images, the images are reshaped into a sequence of flattened 2D patches. A 2D image has the shape of $(H \times W \times C)$, where H is the height, W is the width, and C is the number of channels. The sequence of the flattened patch has shape $(N \times (P^2 * C))$, where $P \times P$ is the resolution of each image patch and $N = HW/P^2$ is the number of patches.

The transformer uses a constant latent (often referred to as projected dimension) vector size D through all of the layers. Each flatten patch is mapped to D dimensions with a trainable linear projection. This is the patch embedding for the ViT, which serves the same purpose as token embedding for transformer in NLP, where each token is represented as a vector. Positional embedding is added to the patch embedding to preserve positional information.

Video Vision Transformers

Video Vision Transformers (ViViT) are developed based on the same architecture as ViT [40]. In the case of ViViT the patches can be sampled as flattened 2D patches from all the frames in uniform frame sampling, as shown in figure 3.15.

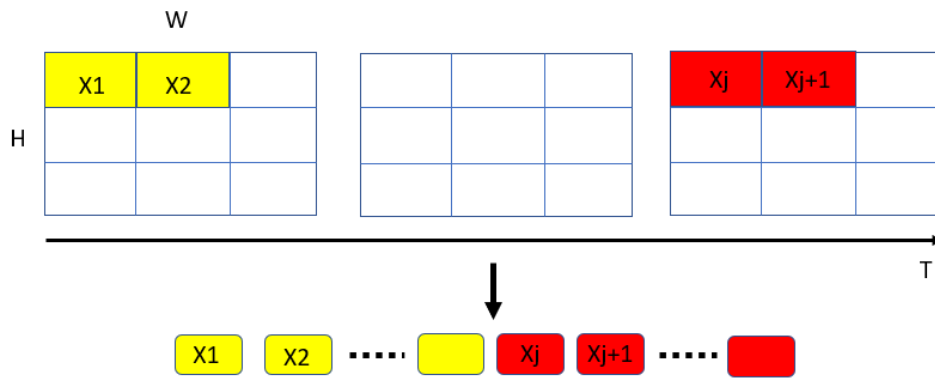


Figure 3.15: Uniform frame sampling from video frames

An alternative method is to extract non-overlapping tubes from the input volume. Tubelets can be achieved by 3D convolution on the input and flattening the 3D tube to a patch. By extracting tubelets, the patches are extracted from the temporal (time), height, and width dimensions of the video. Tubelet embedding is illustrated in figure 3.16

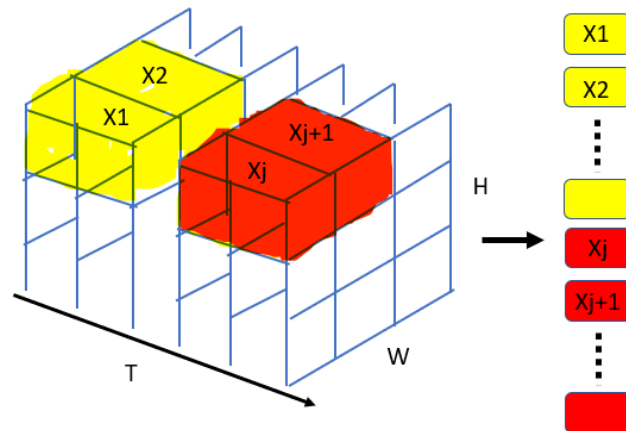


Figure 3.16: Tubelet embedding from video frames

3.7 Masked Autoencoder for Computer Vision

When deep models can overfit one million images and start to demand millions of labeled images, there is a problem with attaining a large enough data set. BERT successfully overcame the problem of demand for huge amounts of labeled data by introducing self-supervised pre-training [28]. In BERT, a portion of the input is masked, and the model learns to predict the masked tokens. The method made it possible for NLP models to perform pre-training on huge amounts of text data without the manual task of labeling all the data.

After the success of BERT, masked autoencoders (MAE) are developed for computer vision on images [41] and videoMAE for videos [42]. The required architecture for positional encoding was addressed in ViT [39]. MAE gives a method for learning features on unlabeled images by masking a proportion of the image and learning by reconstructing the image. The overall architecture for MAE is shown in figure 3.17

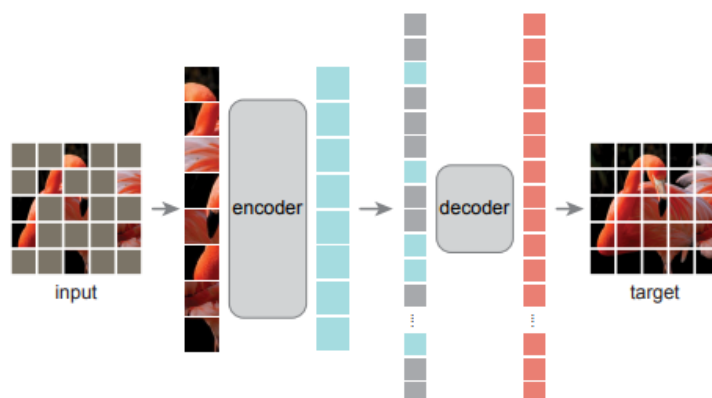


Figure 3.17: MAE architecture [41]. Published under free license.

For MAE the image is split into patches like in ViT. A proportion of the patches is masked, and only the unmasked patches are fed as input to the encoder. The autoencoder learns features by reconstructing the image. The input to the encoder is a portion of the total image, hence it is possible to train very large encoders with a fraction of computational power and memory. The original ViT is implemented with a MLP head for classification after the transformer encoder, as illustrated in figure 3.14. In MAE, there is one layer of normalization to produce the output to the encoder. The input to the decoder is the encoded visible patches and the masked tokens, where each masked token is a vector representing the patch to be predicted. Positional embedding is added to all tokens in the full set to preserve the information about the location in the image.

In MAE, the last layer in the decoder is a linear projection where the number of output channels is the same as the number of pixel values in a patch. The decoder's output is reshaped to form a reconstructed image.

3.7.1 Masking in Vision vs. Text

Languages are human-generated signals with high semantics, and the information is dense, while images are natural signals with high spatial redundancy. With spatial redundancy, we mean elements that are repeated within a structure, and for images, these are areas where the pixel value is similar. Compression of images utilizes the spatial redundancy by removing redundancy while preserving the information. In the same way, patches can be recovered from neighboring patches with little high-level understanding of the different parts in the image. In NLP the model can learn sophisticated language understanding by just masking a few words. To enforce a high-level understanding of the image, a large proportion of the image is masked. Experiments showed that masking 75% of the image gave good results [41]. This reduces the redundancy and enforces learning beyond low-level image statistics. The goal is not a perfect reconstruction of the image but to learn important features.

A video is made up of several frames of images which gives several alternatives for masking, see figure 3.18. One option is to mask random patches in each frame as performed in MAE (row 2 in the figure). Another option is to mask whole frames and keep a portion of the frames unmasked (row 3 in the figure). The third option is to mask the same patches in every frame, which is referred to as tube masking (row 4 in the figure).

Temporal correlation (correlation between frames in time) makes it easy to reconstruct the missing pixels by finding the corresponding patches in adjacent frames for random masking and frame masking [42]. Tube masking is suggested to prevent learning from temporal correlation. Also, the temporal redundancy can be very high for video, making VideoMAE in favor of an extremely high masking ratio (90 to 95%).

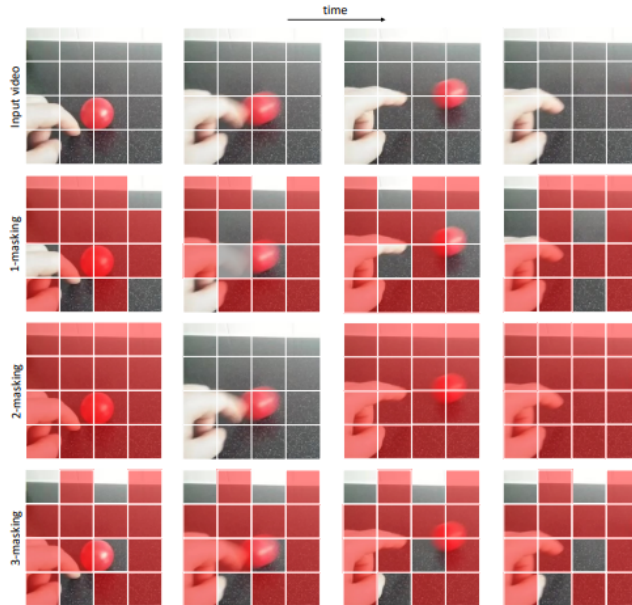


Figure 3.18: Masking of video frame [42]. Published under open license.

3.8 Metrics and Loss Function

A loss function, often referred to as an error function, is a function that maps an event of one or more variables onto a real number that represents some cost associated with the event. In the case of an optimizing problem, we seek to minimize the loss function. Generally, a machine learning problem is an optimizing problem where the model learns by minimizing the loss function. Metrics, on the other hand, is used to measure the performance of the model by measuring the difference between the predicted and expected output. For a classification problem, it is a measure between the predicted label of the test set and the actual label.

3.8.1 Loss Functions

Mean Square Error

The mean square error (MSE) is among the simplest and most common loss functions in machine learning. MSE is calculated by taking the difference between the predicted output and the expected output, squaring it, and averaging it out across the whole data set. MSE is given by

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (3.14)$$

where N is the number of samples we are testing against. MSE will never be negative since the errors are squared.

Binary Crossentropy

Binary crossentropy is used for binary problems with two outcomes. Typical classification problems often have two outcomes that can be reduced to yes or no, or often labeled 0 or 1. For prostate cancer classification, it can be if the lesion is clinically significant or not. The binary crossentropy is given by

$$Loss = -\frac{1}{N} \sum_{i=1}^N y_i * \log(p(y_i)) + (1 - y_i) * \log(1 - p(y_i)) \quad (3.15)$$

where N is the number of samples we are testing against. y_i is the label, and $p(y_i)$ is the predicted probability of the label. We see that the binary cross-entropy first takes the label times the log probability of that label ($\log(p(y_i))$) and adds the other label times the log probability of the other label $\log(1 - p(y_i))$. This is summed for every label and divided by the number of samples to average out.

3.8.2 Categorical Crossentropy

While binary cross entropy is a good loss function for binary classification problems, the categorical loss function is suitable for multiclass classification problems. The categorical crossentropy is given by

$$Loss = -\sum_{i=1}^N y_i * \log(\hat{y}_i) \quad (3.16)$$

where \hat{y}_i is the predicted value from the model output and y_i is the corresponding true labeled value. We see that compared to binary crossentropy the categorical crossentropy only considers the probability of the i-th label and averages over all samples.

3.8.3 Metrics

Confusion Matrix

Many of the metrics measuring the performance of a machine learning model for classification is based on the values in the confusion matrix. The confusion matrix gives a measure of the performance based on the predicted and actual values of the data. Table 6.6 represents a confusion matrix with $n=2$ classes. The term true positive (TP) stands for the number of positive examples classified accurately. Similarly, true negative (TN) stands for the number of predicted negatives that is actually negative. False negative (FN) is the number of predicted negative values that are actually labeled positive and false positive (FP) is the predicted positive values that are actually labeled negative.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive	False Negative
Actual Negative	False Positive	True Negative

Table 3.1: Confusion matrix with $n=2$ classes

The confusion matrix can sometimes seem hard to understand. If we consider the case where a clinical significant case of prostate lesions (cancer) is considered positive and not clinical significant lesions are considered negative, then true positive is the case where the model predicted clinical significant, and the image is clinical significant. True negative will be where the model predicted not clinical significant, and the image was not clinical significant.

When we understand the terms of the confusion matrix, it is easier to understand its relevance to the classification problem. For example, the case of high false negative will be problematic for prostate cancer classification. This means that the model predicts many cases of not clinically significant lesions that are actually clinically significant and require treatment. In the same way, a large amount of false positive may increase over treatment as not clinically significant lesions are classified as clinically significant that require treatment.

Accuracy

Generally, the accuracy measures the ratio of correct predictions over the total number of instances evaluated [43]. The numerator involves the true positive and true negative,

which are all the correct predictions, and the denominator is the sum off all predictions for the data set. Accuracy is given by

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (3.17)$$

Precision

Precision is a measure of the fraction of correctly predicted positive out of all the predicted positive [43]. Precision is given by

$$Precision = \frac{TP}{TP + FP} \quad (3.18)$$

Recall

Recall is used to measure the fraction of positive patterns that are correctly classified [43]. This metric is relevant for cases with an imbalanced data set. Recall will then give an indication of how accurate the model is at correctly classifying relevant predictions. Recall is given by

$$Recall = \frac{TP}{TP + FN} \quad (3.19)$$

F1

F1 represents the harmonic mean between recall and precision [43]. It considers both parameters and gives out a single number as a metric of the performance. It can be difficult to understand what F1 gives compared to precision and recall. Remember that precision is the number of correct positives from all the positives that the model predicted, while recall is the number of correct positives from all positives in the data set. Different tasks can require different performance measurements for precision and recall, but the F1 gives a mean over how good the model is on predicting positive among all the positive labels and how many of the predicted positive were predicted wrongly.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \quad (3.20)$$

Area under the ROC curve (AUC)

AUC is a very popular ranking metric and is considered a better metric of evaluation than accuracy [44]. AUC is defined as

$$AUC = \frac{S_p - n_p(n_n + 1)/2}{n_p n_n} \quad (3.21)$$

where s_p is the sum of all positive examples ranked, while n_p and n_n are the numbers of positive and negative examples. The higher the AUC, the better the model is for distinguishing between positive and negative classes. If AUC=1, the classifier is able to perfectly distinguish between positive and negative classes for every data point. For AUC=0 it will be the opposite, where every data point is classified wrongly. When AUC=0.5, the model is not able to distinguish between the two classes, and it would be like tossing a coin to predict the class. The closer the AUC is to 1, the better the model is at predicting the correct class.

3.9 Software and Packages

The technical part of this thesis is implemented with the programming language named Python. Python is a high-level, general-purpose programming language. A high-level programming language makes the implementation easier by leaving hardware configurations to automated systems. Python is a very popular language for data scientist because of its simplicity and easy integration with powerful packages to speed up computational time and pre-defined functions for machine learning. In the implementation of the model, there is applied several external libraries which comes with premade functions to boost computational time and utilizing GPU hardware.

3.9.1 TensorFlow

TensorFlow is an interface for expressing machine learning algorithms and execute them [45]. The library can be used to implement a wide variety of algorithms for a deep NN, and can be executed on a wide variety of systems. Ranging from mobile devices up to large-scale distributed system and thousands of computational devices such as GPU cards. Tensorflow is applied for its capacity and option to run on GPU.

3.9.2 Keras

The deep learning application programming interface (API) used in this thesis is Keras. The Keras library uses TensorFlow to enable fast implementation and experimentation of deep learning ideas [46].

3.9.3 Numerical Python

The Numerical Python (NumPy) library is developed for Python programming language and supports high-level scientific computing and data analysis for numbers and multi-dimensional array. The library supplies functions for storing large quantity of information in arrays, and fast execution of mathematical operations. This library have been used to generate random integers, generate and store information in arrays, and execute mathematical functions [47].

3.9.4 OpenCV - cv2

OpenCV, cv2, supports NumPy and the objects are returned as NumPy objects [48]. Cv2 is a library for images and includes several functions to manipulate images. Cv2 is used to resize and crop images.

Chapter 4

Data Set and Image Pre-Processing

This chapter presents the data set used for the thesis and filtering to obtain the relevant T2-weighted series. Finally, the image pre-processing to prepare the data for training and evaluation is described.

4.1 Data Set

The data set of prostate MRI used in this thesis is a part of PROSTATEx Challenges (PROSTATEx) [49] which was collected for research on computer-aided detection of prostate cancer in MRI [50]. The MRI was collected by performing a clinical examination at the Radboud University Medical Centre (Radboudumc), Netherlands, in the Prostate MR Reference center. It was collected a total of 165 consecutive studies with prostate cancer (187 lesions) and 183 cases with prostate cancer for a total of 347 patients. Each MR study was read and reported by or under the supervision of an expert radiologist (prof. Dr. Barentsz). The radiologist indicated areas of suspicion, and if an area was considered likely to have cancer, a MR-guided biopsy was performed to verify. Biopsy specimens were graded by a pathologist, which makes up the results for the ground truth. The dataset was collected and curated for research in computer-aided diagnosis of prostate MR under the supervision of Dr. Huisman, Radboudumc. All the studies include T2-weighted (T2W), proton density-weighted (PD-W), dynamic contrast-enhanced (DCE), and diffusion-weighted (DW) imaging. Images were acquired on two different types of Siemens 3T MR Scanners, the MAGNETOM Trio and Skyra.

The T2-weighted images which are relevant for this study were acquired using a turbo spin echo sequence and had a resolution of around 0.5 mm in plane and a slice thickness of 3.6mm.

The data is downloaded from the SPIE-AAPM-NCI PROSTATEx Challenge [51], focusing on the classification of significantly significant prostate cancer. The data is collected from 246 participants and consists of 18,321 series, giving a total of 309 251 images. The details of the data are summarized in table 4.1

Collection Statistics	
Modalities	MR
Number of Participants	346
Number of Studies	349
Number of Series	18,321
Number of Images	309,251
Images Size (GB)	15.1

Table 4.1: Detailed description of PROSTATEx Challenge Data Set

4.2 Data Filtering

4.2.1 Downloading Data

The data is downloaded through NBIA data retriever. All the images can be downloaded for the 346 patients, but there is also an option to download a training cohort of 204 subjects and a test cohort of 140 subjects which is set up for the challenge. The images are provided in DICOM encoding, which provides a lot of metadata like patient ID, sex, age, series description, and much more. The images are downloaded in a folder structure with a folder for each patient ID containing folders for each series. Afterward, the relevant series are loaded and analyzed for further processing in Jupyter Notebook using the pydicom library to work with DICOM files in Python. Figure 4.1 shows a random slice from axial T2-weighted series.

4.2.2 T2-Weighed Ground Truth

Lesion information for the data can be downloaded as .csv files for the train and test cohort. The relevant files are ProstateX-Findings and ProstateX-Images for the training

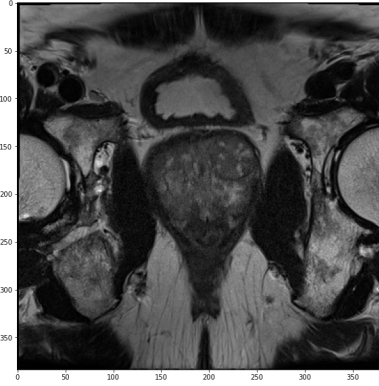


Figure 4.1: T2-weighted MRI slice of Prostate

cohort and similar files for the test cohort. The findings file contains information about findings id, the position of the lesion, which prostate zone the lesion is in, and if the finding is clinically significant or not. The test file contains the same information, except whether the clinically significant or not is not given. The ProstateX-Images file contains information about images with findings. The following relevant information is

- ProxID - Patient ID
- fid - Findings id
- pos - Scanner Coordinate position of the finding
- ijk - image col,row,slice coordinate of finding
- DCMSerDescr – The original DICOM Series Description

Each row represents an image that contains the lesion, which is why there can be multiple rows with the same ProxID and fid. The same lesion can be detected in multiple slices. The ProstateX-Images-Train makes up the ground truth for the data. The file lists images with a finding, and the ground truth can be obtained by comparing them against the finding file. By filtering on series description, we can obtain the information for the axial T2-Weighted images. Table 4.2 shows a detailed description of lesion findings, anatomical zone and classification distribution of the data set and number of ground truth for T2-weighted.

We can see that the ground truth is not balanced between clinically significant and insignificant. Further, there are MRI slices that contains more than one finding. For the

Training cohort	Insignificant	Clinically Significant	Total
lesion	254	76	330
lesion (%)	76.96%	23.04%	100%
TZ	73	9	82
PZ	155	36	191
AFS	24	31	55
Ground Truth	316	96	412
Ground Truth (%)	76.69%	23.31	100%

Table 4.2: Distribution of lesion findings, anatomical zone location and classification with respect to clinically significant or not in the prostate data set

final filtering, a slice can only be passed as an input once. The following final filtering is performed:

- If there is a slice with clinically significant finding and insignificant finding, then the slice will be classified as clinically significant and given as input once.
- If a slice have several insignificant findings, it will be classified as insignificant and only be given as input once.
- For patients with several series of axial T2-weighted, only the first series will be chosen.

Final ground truth distribution is given in table 4.3

Training cohort	Insignificant	Clinically Significant	Total
lesion	254	76	330
lesion (%)	76.96%	23.04%	100%
TZ	73	9	82
PZ	155	36	191
AFS	24	31	55
Ground Truth	256	78	334
Ground Truth (%)	76.64%	23.36	100%

Table 4.3: Final ground truth distribution of the data

4.3 Image Pre-Processing

4.3.1 Image Resizing

The majority of the T2-weighted images are given with a resolution of 384x384 pixels. A few images are stored with a resolution of 320x320 and 640x640. The model requires

each image to have the same width and height as the input. Due to memory capacity, the most normal resolution of 384x384 is too big for the model to handle. Each image is resized to a width and height of 128x128 pixels. The images are resized utilizing OpenCV for python with the built-in resize function. Interpolation is given as INTER_AREA, which uses pixel area relation for resampling.

4.3.2 Grey Scale Filtering and Image Normalization

DICOM 16-bit images have pixel values ranging from $[-32768, 32768]$. The large range of values in DICOM is useful as they correlate with the Hounsfield scale, which is a quantitative scale for describing radio density. A random slice of 128x128 with a corresponding histogram of pixel value distribution is shown in figure 4.2.

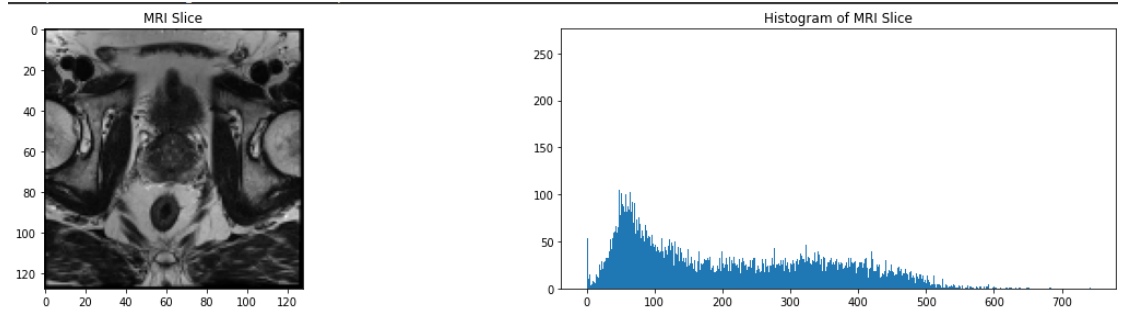


Figure 4.2: Visualize a random slice from the original dataset without any filter or normalization

Normalization is applied to each image to convert it to grayscale images with pixel values ranging from $[0, 255]$. The normalization is given by equation 4.1

$$I_N = (I - Min) \frac{NewMax - NewMin}{Max - Min} + NewMin \quad (4.1)$$

where $[Min, Max]$ is the minimum and maximum pixel value in the input image I and $[New Min, New Max]$ represents the desired new min and max pixel value in the output image I_N . The image is also filtered with outlier removal filter to remove possible image noise. The darkest and brightest 1% pixel values are replaced with their neighboring values. The same random slice and distribution of pixel value after greyscale filtering and outlier removal is shown in figure 4.3

Before an image is ready for input to the machine learning model the values need to be normalized to a value between 0 and 1. The normalization is performed by dividing each

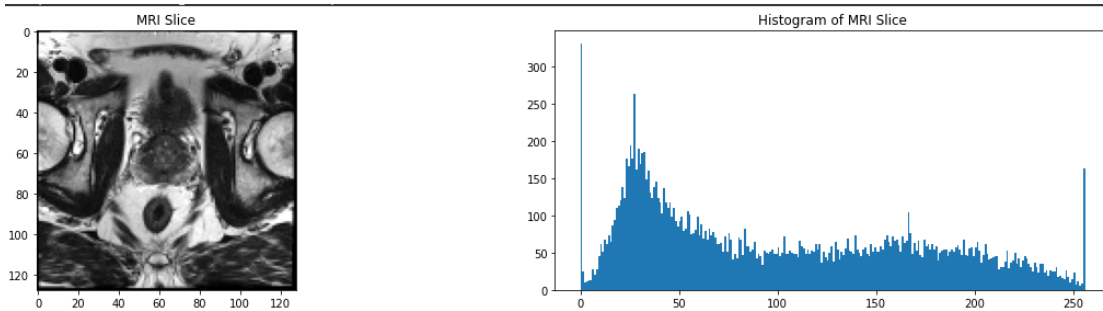


Figure 4.3: Visualize a random slice from the original data set with grayscale filtering and outlier removal filter

pixel value by the max pixel value 255. This is also the reason for first normalizing to grey-scale. If not, each image is divided by the max pixel value from the slice containing the highest pixel value, which can be much higher than the current slice.

4.3.3 Organizing Data Sets

There are two data sets made up for the training and evaluation of the model. The first data set is for the autoencoders and consists of all axial T2-weighted slices for patients. Where patients have more than one series, only the first series is chosen. This makes up a total of 7060 slices for the autoencoder. The complete data set is divided into training, validation, and test sets with a distribution of 70%, 10%, and 20%, respectively. To acquire a balanced distribution of slices from the whole prostate, slices from one patient can only exist in one set.

The second data set is made up for the transfer learning and classification of insignificant and clinically significant lesions. The distribution of ground truth slices is given in section 4.2.2 and makes up 334 slices. The data is divided into training, evaluation, and test sets with the same 70%, 10%, and 20% distribution. To get a balanced distribution of clinically significant and insignificant slices, the images are first organized into two arrays, with all the clinically significant in one array and the insignificant in another. Arrays with corresponding labels 0 and 1 are equally made. After this, the clinically significant and insignificant images and labels are distributed into the train, evaluation, and test set. Each set of images and corresponding labels are randomly shuffled, with the same shuffle for images and labels in the same set before being passed as input to the model.

Chapter 5

Solution Approach and Configuration

5.1 Model Architecture

The model is based on the implementation of masked autoencoder given as code examples in Keras home page [52], which is an implementation of the masked autoencoder given in the paper [41]. The code is updated from handling color images with three channels to grayscale images with one channel. Generally, since there are very few labeled images for the ground truth data set, the model is implemented with an autoencoder for pre-training on the MRI images and only an encoder for the classification task. The overall architecture for the autoencoder is illustrated in figure 5.1. In the classification task, the weights from the encoder during pre-training are kept, and one additional dense layer of one neuron with sigmoid activation is added for binary classification.

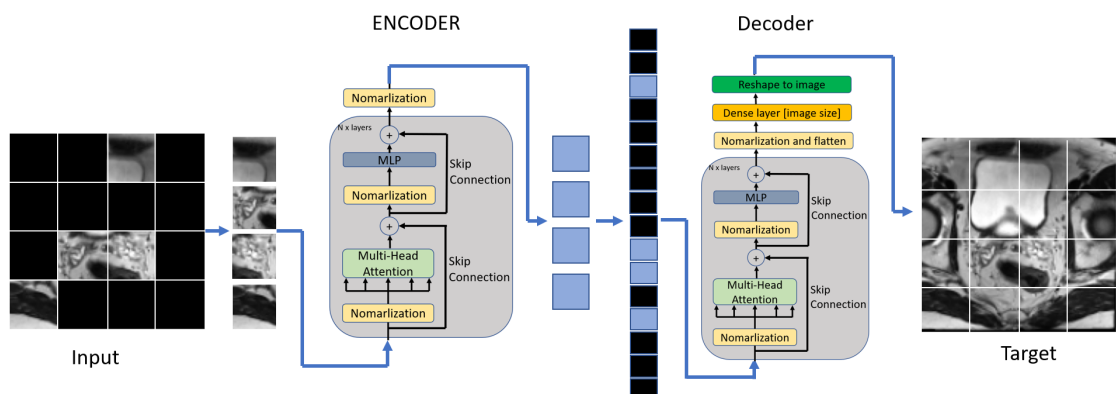


Figure 5.1: Model architecture for the autoencoder

5.1.1 Patches

The patches are extracted as regular non-overlapping by 2D convolution over the image. The stride is set to the patch size in both x and y directions to get all the pixels in the image. The patches class is implemented with functions to show the patched image and reconstruct the image from patches. The reconstructed image is used to verify the masked image during training. Illustration of the original image, patched image, and the reconstructed image for a random slice is illustrated in figure 5.2

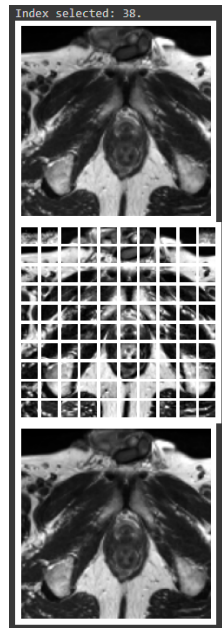


Figure 5.2: Original image at top, patched image in middle and reconstructed image at bottom

5.1.2 Patch Encoding and Masking

Downstream patch encoding

The patch encoder makes the patch embedding for all patches by a linear projection of the patch and added positional encoding, see figure 5.3. The patch encoder is split between pre-training and downstream tasks. If downstream equals true, patch projection and positional encoding will be made for every patch in the image. The patch projection and positional encoding are summed up to be the patch embedding. The whole image is input to the model when performing classification.

Autoencoder masked patch encoding

For the pre-training, the projection and positional encoding is a little different. A masked token is initialized randomly from a normal distribution and is trainable. The masked token is a vector of the same size as pixels in a patch (patch size x patch size), which represents a flattened masked patch. This is the token the decoder will try to reconstruct. Next, the patch projection and positional encoding will be made for all the patches in the image and summed up to patch embedding as shown in figure 5.3.

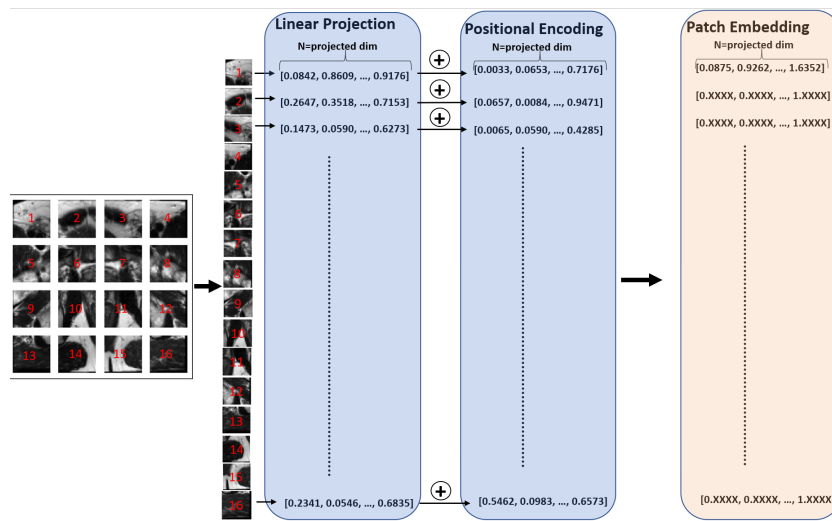


Figure 5.3: Patch embedding for downstream classification task

The difference is that now the projection and positional encoding will be split between the masked and unmasked patches. Figure 5.4 illustrates the process for masked embedding. A vector of random indices for the number of patches is made from an uniform distribution. These indices are split into a masked and unmasked vector according to the proportion of masking. The unmasked embedding is made from gathering the patch embedding for the unmasked indices. Unmasked position embedding is also gathered as this will be needed for the decoder. For the masked embedding, the first part is to gather masked tokens equal to the number of masked patches. The masked tokens are then projected to form up masked projection. The masked position encoding is gathered from the full position encoding for the masked indices. The masked embedding is made by summing the masked projection and masked position encoding.

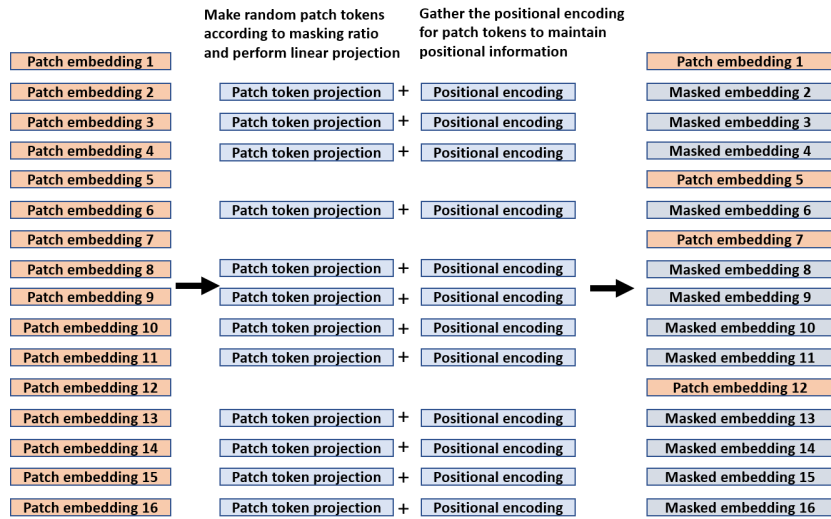


Figure 5.4: Masked patch embedding for pre-training

5.1.3 Autoencoder Architecture

Encoder

The encoder follows the architecture of the original architecture of ViT [39] and is shown in figure 5.5.

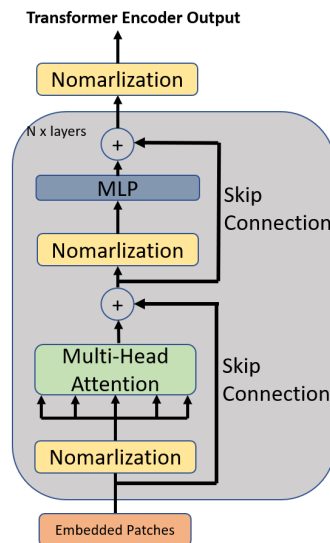


Figure 5.5: Encoder architecture

The encoder takes as input the embedded patches. The same two sub-layers from the original ViT are present in the gray box and are stacked in the desired number of layers. There is a residual connection for the multi-head attention and one residual connection for the MLP. The MLP has two layers with hidden dimension [encoder projected dimension

x 2, encoder projected dim] with Gelu activation. The only difference is the additional normalization of the output from the stacked number of layers. Where the original ViT has an additional MLP head for classification, the encoder for the autoencoder has a final normalization before the output of the encoder is fed to the decoder.

Decoder

The decoder has a similar architecture for the stacked layers as the encoder. There is one additional dense layer with the numbers of units equal to the image size and channels. For grayscale images, the number of channels is one. The dense layer has sigmoid activation to return a value between 0 and 1 to represent the normalized pixel values which are input to the autoencoder. The output of the dense layer is reshaped to make up the image. From the reconstructed image, we can again sample patches to compare patches from the original image and the reconstructed image.

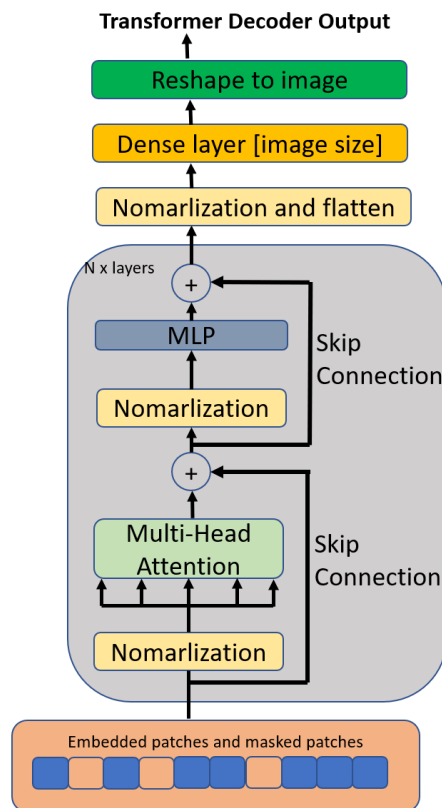


Figure 5.6: Decoder architecture

Loss function

The loss is only calculated for masked indices. Patches from the original image and reconstructed image are gathered from the masked indices. The mean square error function does not include patches which, where unmasked in the input.

5.1.4 Downstream Model Architecture

For the downstream model there is an additional module of linear probing. The encoder is taken from the masked autoencoder and the weights from the pre-training is kept.

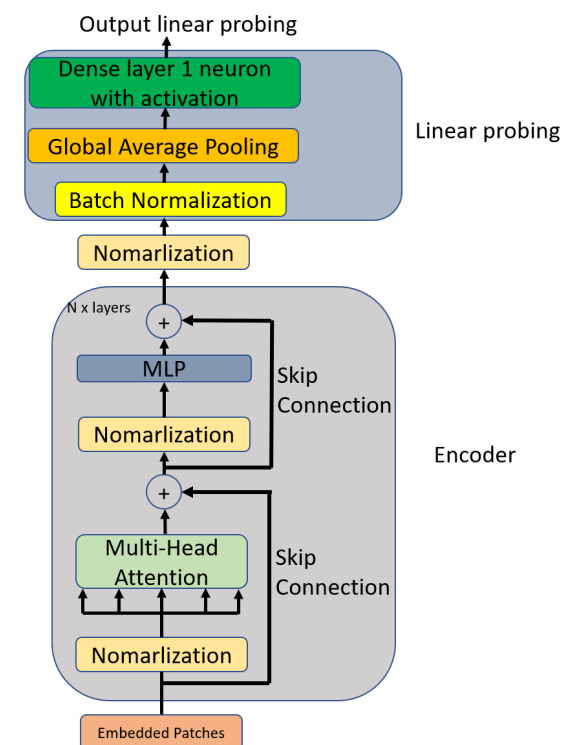


Figure 5.7: Decoder with additional linear probing

Instead of the additional MLP head in ViT used for classification, we keep the normalization from the autoencoder as it is common practice to normalize the input to a linear classifier [53]. The normalized output is further processed by batch normalization according to [54]. Pre-trained features can degenerate to all activations collapsing to zero. The network will not be able to tune the lower-level features. Batch normalization will force the network activation to vary across examples [54]. A global average pooling is performed before the final layer to make a feature map for each class. The final layer is implemented as a dense layer with one neuron, which is the classification layer. The

neuron has sigmoid activation to produce an output between 0 and 1 which corresponds to the labels 0 and 1.

5.2 Configuration

5.2.1 Width and depth of the network

Autoencoder

The width of a deep neural network is often given as the number of neurons in the widest layer, and the depth is given as the number of layers. For the autoencoder, the stacked layers of multi-head attention and MLP have the same architecture for encoder and decoder. The width of the multi-head attention is the encoded projection dimension of the flattened patches. The MLP has two layers. The width of the first hidden layer of the MLP is two times the projected dimension, and the output layer has the width of the projected dimension. The decoder has an additional layer with the number of neurons equal to the number of pixels in the image times the number of channels. In the case of grayscale MRI, there is only one channel, resulting in the number of neurons being the number of pixels in the image.

The depth of the network is the number of layers. For the hyperparameters, the number of layers decides the number of layers for the stacked multi-head attention and MLP shown as the two sub-layers in figure 5.5 and 5.6. The decoder has one additional layer to produce values equal to the number of pixels.

The autoencoder follows the main ideas of a smaller decoder compared to the encoder [41]. The goal of the autoencoder is to enforce learning of useful features from the MRI image and not reproduce a good representation of the image. The projected dimension for the decoder is half of the encoder, meaning that the width of the decoder is half of the encoder. This greatly reduces the computational and memory requirements for pre-training. The number of layers of the decoder is also equal to or less than half the layers of the encoder. Table 5.1 shows the number of parameters for the autoencoder in the different layer types for an image of 128×128 pixels, patch size of 16, encoder projected dimension of 128, decoder projected dimension of 64 and number of layers in the encoder and decoder equal to 6 and 2. We see that the number of parameters is

dominated by the decoder, even though the width and number of layers are half of the encoder. The decoder takes in all the trainable masked tokens compared to the encoder, and the last layer with neurons equal to the number of pixels in the image adds on a lot of parameters in the decoder.

Layer (type)	Parameters
Patches	0
Patch encoder	41,344
MAE encoder	1,981,696
MAE decoder	67,300,032
Total parameters	69,323,072

Table 5.1: Number of parameters for autoencoder with image size = 128x128, patch size = 16x16, encoder projected dimension = 128, decoder projected dimension = 64, number of heads = 4, number of layers for encoder = 6, and number of layers for decoder = 2

Downstream model

The downstream model extracts the trained encoder from the autoencoder and adds the batch normalization, global average pooling, and the additional layer of one neuron for classification. The total width of the downstream model will be the same as the autoencoder, but there is one additional layer for classification. By keeping the weights from the encoder and only allowing the last additional layer to be trainable, we get the following number of parameters and trainable parameters presented in table 5.2.

Layer (type)	Parameters
Patches	0
Patch encoder	41,344
MAE encoder	1,981,696
Batch normalization	512
Global average pooling 1D	0
Dense layer classification	129
Total parameters	2,0123,681
Trainable parameters	129
Non-trainable parameters	2,023,552

Table 5.2: Number of parameters for downstream model with image size = 128x128, patch size = 16x16, encoder projected dimension = 128, number of heads = 4, number of layers for encoder = 6

5.3 Augmentation

Image augmentation is a well-established method to artificially expand the training data set. The data set is not expanded to more samples of images, but alternative variations of the images are fed as input to the model for each epoch¹ or batch². Augmentation is implemented through the Keras layers API [46]. Masked autoencoders for images are not dependent on augmentation and work well with little to no augmentation [41]. The following image augmentation is implemented for the model during training.

- Random crop: The images are resized a little compared to input size of 128×128 pixels. After the resizing, random crop is implemented where the location for cropping is randomly chosen to crop the images down to target size. All the images in the same batch are cropped to the same location.
- Random flip: Randomly flips images horizontally during training. This is equal to reverse all rows of pixels.

During testing the images are only resized to the same size as random cropping.

5.4 Hyperparameters

Image Size

Image size is the size of the image fed as input to the model and is given as one side of the image. The images have the same height and width as input to the model. During training, the input image is resized to a larger width and height and then cropped down to image size. During testing, the input image is simply resized to image size.

Patch size

Patch size is the size of the patches extracted from the image. The patch size gives the height and width of the patch, meaning that every patch is a square. The patch size and

¹One epoch represents that the full data set is passed through the network once.

²Batch size is the number of training samples simultaneously passed through a neural network. Each batch size is an iteration in one epoch.

image size correlate so that the image can be divided into the exact number of patches with no overlap or gap between the patches.

Masking ratio

The masking ratio decides how many percent of the image is masked. From the masked ratio images will be masked randomly from a uniform distribution according to the masking ratio.

Learning rate

Learning rate is the tuning parameter for the optimizer algorithm. The learning rate is initially set to a small value. The model is implemented with an adaptive cosine decay learning rate [55] and warmup steps [56]. The warmup slowly increases the learning rate at the beginning of training to avoid rapid changes in the network and early optimization issues. The cosine decay learning rate gradually decreases the learning rate after reaching the top after warmup. The goal of the adaptive learning rate is for the optimizer algorithm to find the global minimum and not converge towards a local minimum. Adaptive learning rate for one simulation is shown in figure 5.8.

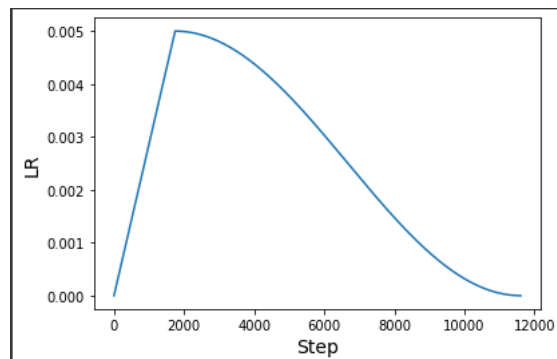


Figure 5.8: Adaptive learning rate with warmup and cosine decay

Batch Size

Batch size decides how many images are processed through the model for each iteration in the epoch. Batch size times iteration is processing all images in one epoch.

Encoder and decoder

There are several hyperparameters deciding the configuration of the network according to section 5.2.1.

- Encoder projected dimension: Sets the projected dimension for the patch encoder and the width for the layers in the encoder.
- Decoder projected dimension: Sets the width of the decoder
- Encoder number of head: Sets the number of heads for the multi-head attention in encoder
- Decoder number of head: Sets the number of heads for the multi-head attention in decoder
- Encoder layers: Number of stacked layers in encoder
- Decoder layers: Number of stacked layers in decoder

5.5 Limitations

The configuration of the model is mostly restricted by the memory capacity given by google colab. The time limitation of 12 hours running time was not an issue when the size of the model was limited. The biggest limitations were image size, projected dimension, and patch size. With a larger image, the number of patches does not increase linearly but exponentially. As the projected dimension increase, the width of the network increase in every layer. Also, the patch embedding increase with the projected dimension. The limit was 128 for the encoder projected dimension. With this size, the model was able to run a batch size of 32. Increasing the projected dimension would surpass the memory capacity, even with a batch size of 1. For a patch size of 16, the projected dimension is smaller than a patch. For a patch size of 10, the projected dimension is a little larger than the flattened patch. However, decreasing patch size would also exhaust the memory capacity. With a patch size of 10, the batch size needed to be reduced to 4.

Chapter 6

Experimental Evaluation

This chapter presents the experimental setup, results, and performance measures for masked autoencoder on prostate MRI.

6.1 Experimental Setup

The masked autoencoder and downstream model used to test the proposed approach is based on the masked autoencoder given in the paper[41] and implemented according to code examples in Keras home page [52]. The last layer of the downstream model is changed from multiclass classifier with softmax activation to a single neuron with sigmoid activation for binary classification. The code for extracting the images, filtering the data, pre-processing, and labeling is implemented from scratch.

The autoencoder is implemented as a generative self-supervised pre-training model to train an encoder to encode the input into an explicit vector and a decoder to reconstruct the input. The loss function used to optimize the autoencoder is the mean square error on the masked patches only, and the evaluation measure is the mean average error. The optimization algorithm for the pre-training is Adam with an adaptive learning rate and weight decay of $1e-4$. The base learning rate is $5e-3$, and warmup with a start learning rate of 0 and cosine decay is implemented for adaptive learning rate. The pre-training is trained with epochs between 150 and 750 with a batch size of 32 for a patch size of 16 and a batch size of 4 with a patch size of 10.

The downstream model is implemented with the encoder and weights from the pre-training and added layers with batch normalization, global average pooling, and a layer with a single neuron for classification of significantly significant and insignificant lesions. The loss function is binary crossentropy for classification, and the evaluation measure is accuracy and AUC during training. The optimizer algorithm for the downstream task is SGD with an adaptive learning rate and 0.9 momentum. The base learning rate is increased to 0.1, inspired by the original paper [41]. Training is performed with epochs from 30 to 250.

The proposed method is generative self-supervised pre-training on the unlabeled MRI slices to learn useful features in prostate MRI. With the small amount of labeled data, the goal is for the encoder to learn general useful features from a larger data set. Transfer learning is then applied with the downstream model. The downstream model keeps the learned weights from the encoder and trains the last layer to classify clinically significant lesions and insignificant lesions.

6.2 Experimental results

6.2.1 Result of Pre-Training

As the goal of the pre-training is not to reconstruct the image, but for the encoder to learn useful features, the evaluation of the autoencoder through the evaluation metric does not give a clear understanding of which features it has learned. The development of the loss function can indicate that the model is learning during training as it is punishing dissimilarities between the original patches, which are masked, and the reconstructed masked patches. More similarities will reduce the loss function. A random reconstructed patch is also printed at a certain epoch interval so we can see the development during training. As the model is small and the decoder has half the width and fewer layers than the encoder, we do not expect good reconstructed images.

The development of the loss function and reconstructed images are illustrated for a few hyperparameters. The fixed parameters that are not changed for the described results are listed in table 6.1.

Parameter	Value
Encoder projected dimension	128
Decoder projected dimension	64
Encoder number of heads	4
Decoder number of heads	4
Optimizer	Adam
Base learning rate	$5e-3$
Warmup learning rate	0

Table 6.1: Constant hyperparameters for all simulations

Autoencoder Model 1

The parameters for the first simulation is shown in table 6.2

Parameter	Value
Epochs	750
Masking ration	0.75
Number of layers encoder	6
Number of layers decoder	2
Image size	128
Patch size	16
Batch size	32

Table 6.2: Hyperparameters for simulation 1

The corresponding development of the loss function is shown in figure 6.1.

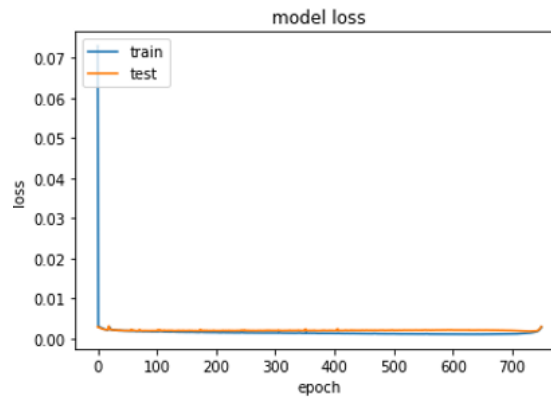


Figure 6.1: Loss during training for model 1

We see that the loss is decreasing fast and then moving almost horizontally after 100 epochs. The loss on the training set decreases negligibly after 200 epochs, while the loss on the evaluation set (labeled test in figure) increases slightly. There is not necessarily much gain from running many epochs. Figure 6.2 shows printout of the reconstructed image for epoch 1 and 746. The image in the left column is the original image, the image

in the middle column shows the masked input, and the image in the right column is the reconstructed.

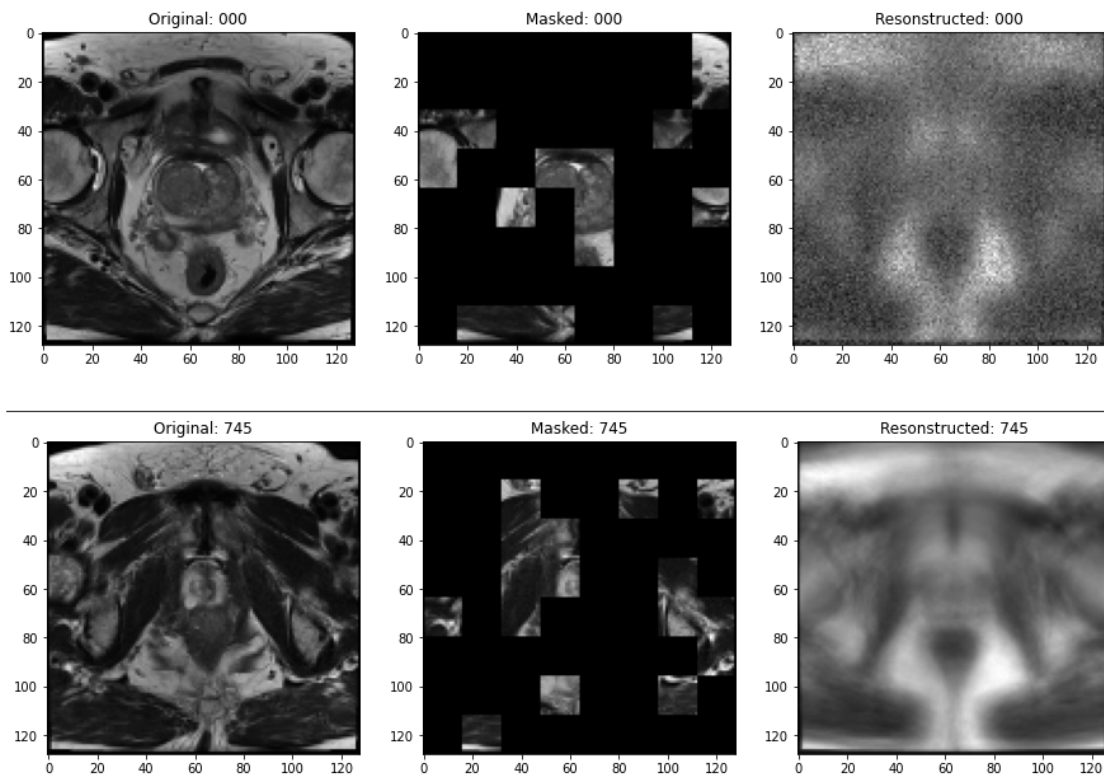


Figure 6.2: Model 1 - Reconstructed image for epoch 1 at top row and epoch 746 at bottom

The figure clearly shows that the autoencoder has learned during training. The top row representing the first epoch has a lot of noise and is not a good reconstruction of the slice. The light area covering the pixel range $([80,40], [120,90])$ seems more like the slice in the bottom row. At epoch 746, the reconstructed image has a decent representation of the dark and light areas in the original slice. None of the details in the MRI is preserved in the reconstructed image. However, the goal is not to reproduce the details but for the details to be preserved in the latent space of the encoder.

Autoencoder Model 2

The parameters for the second model are shown in table 6.3. The image size is increased to 160 and the batch size is reduced to 8 to compensate for memory restrictions.

Parameter	Value
Epochs	250
Masking ration	0.75
Number of layers encoder	6
Number of layers decoder	2
Image size	160
Patch size	16
Batch size	8

Table 6.3: Hyperparameters for simulation 1

The development of the loss function during training is shown in figure

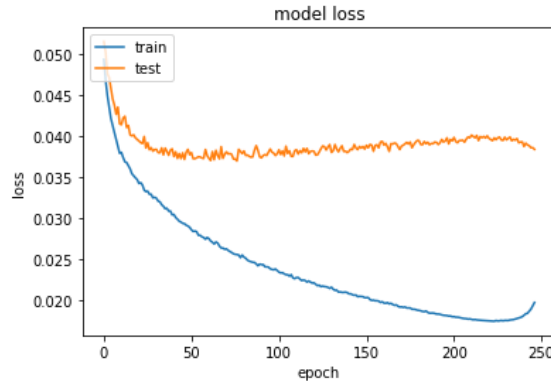


Figure 6.3: Loss during training for model 2

It takes around 40 epochs before the loss of the validation set starts to increase. The loss on the training set continues to decrease until around epoch 230, before it increases. An increase in loss may suggest overfitting of the data with too many epochs. Figure 6.4 and 6.5 shows reconstructed images for model 2. In figure 6.4 the top row is the first epoch and second rows is epoch 176. We see that the reconstruction is not similar for the first epoch, while for epoch 175 the model can reconstruct the contours of the light and dark area decently. We can distinguish the rectum in the lower middle part, and prostate gland in the middle of the slice. The same applies for figure 6.5 where the rectum is clearly more correct for epoch 111 compared to epoch 11.

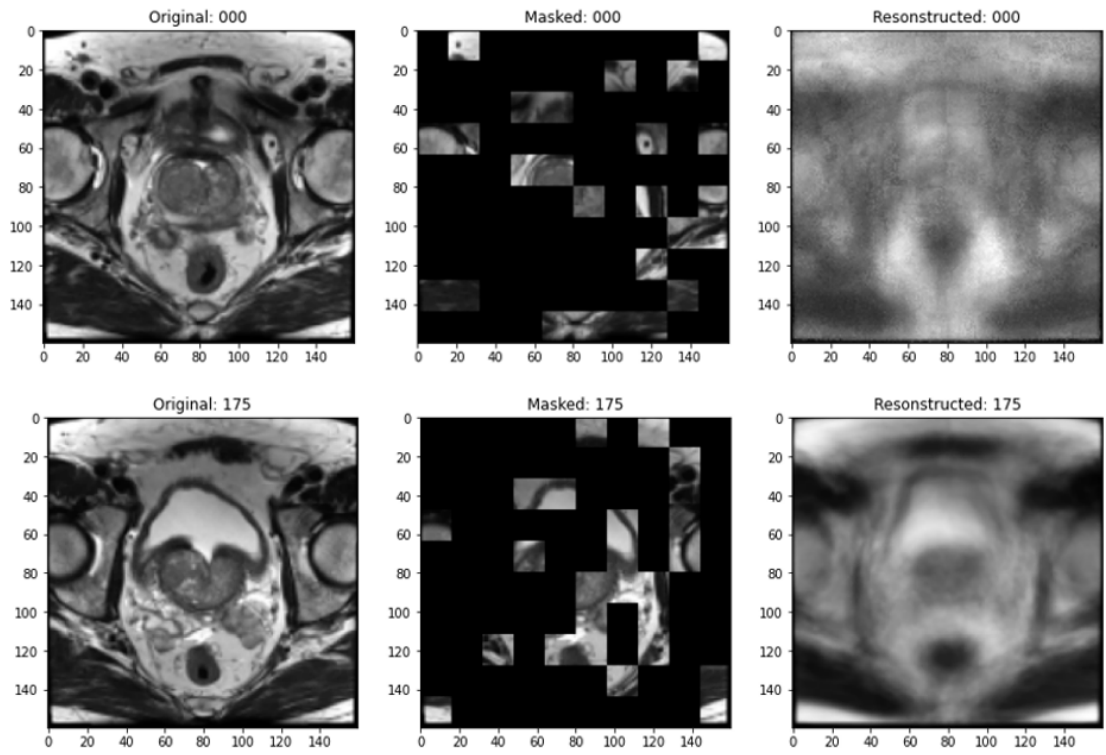


Figure 6.4: Reconstructed image for epoch 1 at top row and epoch 176 at bottom

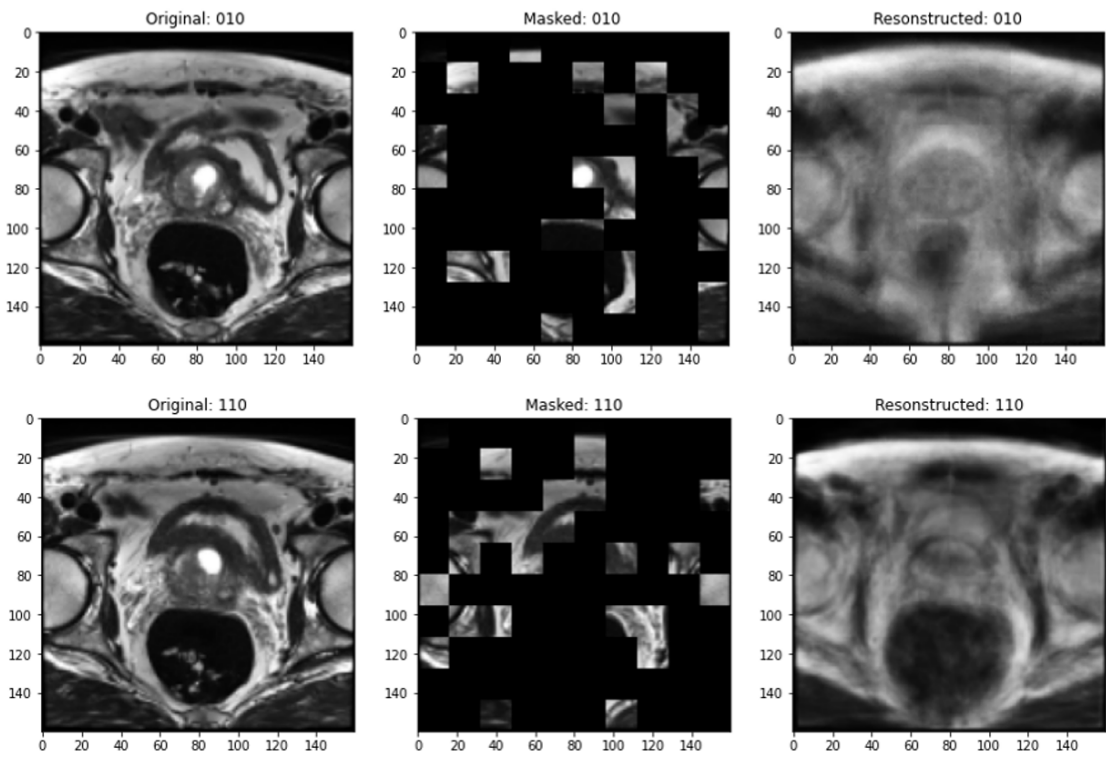


Figure 6.5: Model 2 - Reconstructed image for epoch 10 at top row and epoch 111 at bottom

6.2.2 Results from Downstream Model

The downstream model is not able to distinguish between clinically significant and insignificant lesions. Several configurations and different runs changing the hyperparameters and the number of trainable layers in the downstream model were completed without producing satisfactory results. Some of the runs are presented to show the evaluation of the model.

Downstream Model 1

Using the decoder from autoencoder model 1 in section 6.2.1. For the downstream model the parameters are given in table 6.4.

Parameter	Value
Epochs	50
Optimizer	SGD with momentum = 0.9
Base learning rate	0.1
Warmup learning rate	0
Number of layers trainable	1

Table 6.4: Downstream model 1 - Hyperparameters

The development of the loss for model 1 is given in figure 6.6

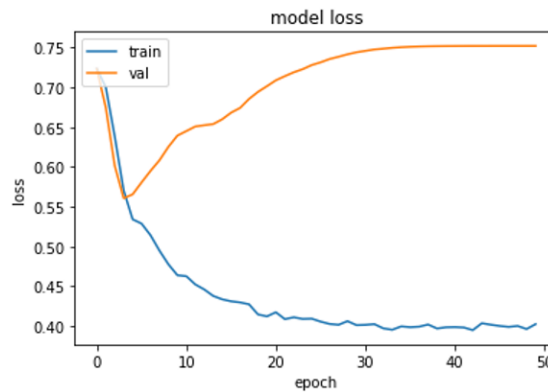


Figure 6.6: Loss during training of downstream model 1

We see that the loss for the training set is decreasing almost throughout the training. The loss for the validation set is, however, increasing after 5 epochs and stabilizing around 0.75. The corresponding accuracy is shown in figure 6.7.

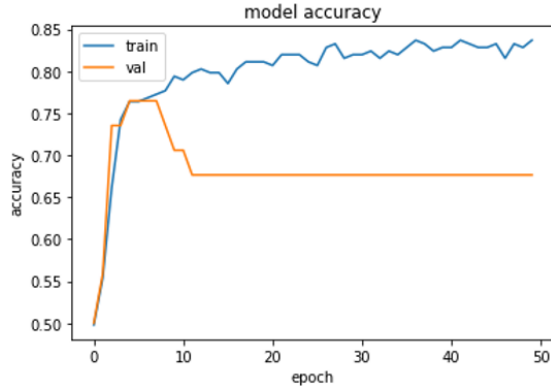


Figure 6.7: Accuracy during training of downstream model 1

The accuracy is steadily increasing for the training set towards 0.85. The accuracy for the validation set is first increasing the first 5 epochs and then decreasing before stabilizing at approximately 0.68. The AUC is showing similar trends in figure 6.8

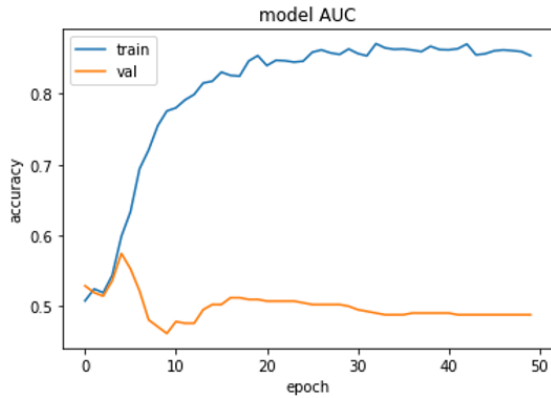


Figure 6.8: AUC during training of downstream model 1

The AUC is increasing for the training set, which should suggest that the model is able to distinguish between classes. However, the AUC for the validation set is over 0.5 for a few epochs before decreasing to below 0.5. The validation set suggests that the model can not distinguish between the two classes and that the model is more likely to predict around 50/50 or predict more wrong than correct.

The resulting performance score on the test set is presented in table 6.7

Accuracy	AUC	Precision	Recall	F1
0.58	0.46	0.125	0.125	0.125

Table 6.5: Downstream model 1 - performance score

The model predicts several significant lesions, but out of the predicted significant cases, only two are predicted correctly as true positive. The confusion matrix is shown in table 6.6

	Predicted Positive	Predicted Negative
Actual Positive	TP=2	FN=14
Actual Negative	FP=14	TN=37

Table 6.6: Confusion matrix for downstream model 1

To check the model, the number of trainable layers was first increased to 3. By this, we mean the last 3 trainable layers. The downstream model was also initialized from the start without preserving the weights from the autoencoder. This corresponds to a ViT, which is implemented with an encoder, except the last MLP head is replaced with single neuron with sigmoid activation. The results are summarized in table

Trainable layers	Accuracy	AUC	Precision	Recall	F1
3 trainable layers	0.76	0.26	0	0	0
Initialized with random weights	0.71	0.41	0.285	0.125	0.173

Table 6.7: Downstream model 1 - performance score with 3 trainable layers and initialized from start

The result of increasing the last three layers to trainable was that non of the predictions were true positive. When initialized with random weights from the start, there are a few less false positive. Reduced from 14 to 5, which gives a little higher precision and recall. The higher accuracy comes from fewer predictions of false positive. Since there are very few clinically significant samples in the test set, the accuracy will increase if almost all predictions are insignificant.

Downstream Model 2

The downstream model is based on an image size of 160 pixels and a patch size of 10 pixels. The number of layers is increased to 10 for the encoder and 5 for the decoder. The parameters for the autoencoder and downstream model are given in table 6.8.

Parameter autoencoder	Value
Epochs	250
Masking ration	0.75
Number of layers encoder	10
Number of layers decoder	5
Image size	160
Patch size	10
Batch size	4
Parameter downstream	Value
Epochs	250
Masking ration	0.75
Optimizer	SGD with momentum = 0.9
Base learning rate	0.1
Warmup learning rate	0
Number of layers trainable	1, 3 and intialized by random weights

Table 6.8: Downstream model 2 - Hyperparameters

The same trends as model one can be seen in the results. The development of the AUC is shown in figure 6.9. The AUC on the training set is increasing while the AUC on the validation set is decreasing.

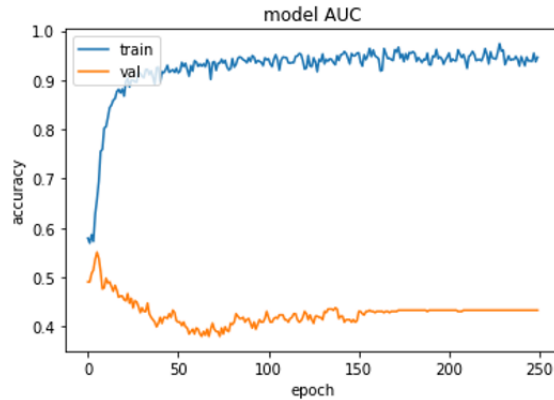


Figure 6.9: AUC during training of downstream model 1

The AUC on the validation set suggests that the model is not able to distinguish between the two classes. This is confirmed by the performance measures for the model shown in table 6.9

Trainable layers	Accuracy	AUC	Precision	Recall	F1
1 trainable layer	0.76	0.5	0.125	0.125	0.125
3 trainable layers	0.76	0.5	0.125	0.125	0.125
Initialized with random weights	0.71	0.4	0.125	0.125	0.125

Table 6.9: Downstream model 2 - performance score with 3 trainable layers and initialized from start

With all configurations of trainable layers, the model results in two true positives. Same confusion matrix as table 6.6 for downstream model 1.

6.2.3 Cropping the Image Around Prostate Gland

Due to unsatisfying results for the full slice, the images are cropped to remove unnecessary information in the slice. Taking the full slice, the prostate gland is a very small part of the image. The cropped image is shown in figure 6.10.

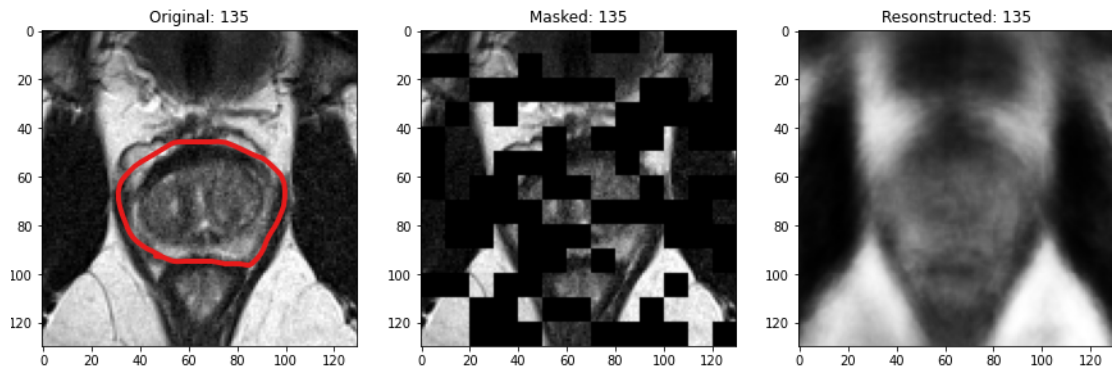


Figure 6.10: Reproduced image when cropping around prostate gland. Prostate gland marked in red.

The prostate gland is marked in red. The rectum is below in the image, and the bladder is the dark area above. The results are still very similar to the previous. With the same parameters as downstream model 1, the number of true positives was increased to 3. However, the number of false positives was also increasing.

Chapter 7

Discussion

This chapter presents a discussion of the proposed approach and model, the achieved results, limitations and alternative approaches.

7.1 Approach and Model

As publicly available labeled images are very limited for prostate cancer, a self-supervised approach was attempted to improve prostate cancer diagnosis. Public labeled images are generally limited for medical diagnosis with assisted machine learning. In most countries, there are strict regulations for revealing patient medical information, and it is often required with written consent from each patient. The idea is based on promising results in NLP and machine vision, where models are pre-trained on large amounts of unlabeled data and then apply transfer learning on the limited available labeled data. The masked autoencoder outperformed previous best results when pre-training on ImageNet and apply transfer learning on iNat and Places dataset [41]. Self-supervised vision transformer based on denoising has also proven to outperform models based on ResNet [36]. The results are based on simple self-supervised learning methods.

The same simple approach of masked autoencoders was implemented to try and improve prostate cancer diagnostics. The autoencoder performs pre-training on a larger dataset of unlabeled images for the encoder to learn the latent representation of the image. Transfer learning is then applied to the labeled images by keeping the trained encoder and adding simple linear transformation through batch normalization, global average pooling, and a

final layer with one neuron for classification. Compared to implementation of ViT and masked autoencoder model applied in this thesis is very small due to limitations on GPU memory. The original ViT-Huge was implemented with 32 layers, a projected size of 1280 and 16 heads adding up to 632 million parameters [39]. The masked autoencoder is based on the same encoder as ViT and is implemented with projected dimensions ranging from 768 for ViT-Base up to 1280 for ViT-Huge [41]. The model applied in this thesis has a projected dimension of 128 and a maximum of 12 layers for the encoder.

7.2 Image Pre-Processing and Augmentation

Little image pre-processing was performed before inputting the images to the model. The standard normalization of converting DICOM images to grayscale with pixel values between 0 and 255 and then normalization to values between 0 and 1 by dividing each image by 255 was performed. An additional outlier removal filter is applied to replace the darkest and brightest 1% pixel values. This was considered reasonable image pre-processing as the histogram shows a good spread on the pixel values. Histogram equalization is an option to spread pixel values over the range between 0 and 255. The biggest impact of the pre-processing might be the resizing of the images. Resizing from 384x384 to 128x128 will affect the details in the image. When an image is downscaled by a factor of three, it will affect the details in the image.

Basic augmentation was implemented to artificially expand the dataset a little during training. Augmentation is usually a vital part of increasing the performance of neural networks. Based on the masked autoencoder where augmentation gave little to no increase in performance [41], only random cropping and random horizontal flip were implemented. Using augmentation or not had no impact on the results for the downstream model.

7.3 Results and Classification

The results are not satisfactory, and the model is not able to distinguish between clinically significant lesions and insignificant lesions. The autoencoder seems to learn the features distinguishing between tissue that gives the light and dark areas in the MRI.

The reconstruction improves with training, and more features are represented in the reconstructed image.

The model is not suitable for classification. By tuning the model, it will predict more clinically significant cases, but almost all of them are false predictions of clinically significant. Even when cropping the image to remove unnecessary information and focusing on the prostate gland, the model is far from being able to classify the lesions. The size of the model is a huge disadvantage. The width and depth of the model are several times smaller than the ViT and masked autoencoder, which are starting to outperform other deep neural networks for image classification. It is also hard to pinpoint how large a data set is required to train the model. Ian Goodwill's book *Deep Learning* suggests that there should be at least 5000 ground truth examples for achieving acceptable performance [17]. This is almost seventeen times more than is available for prostate MRI. The self-supervised approach is implemented to overcome the issue, but ViT and masked autoencoders were pre-trained on tens of thousands to millions of images and fine-tuned or applied transfer learning on several thousands or more images.

Several variations of hyperparameters extending the results presented in section 6.2 were explored without reaching better performance. Longer and shorter runs with number of epochs, fixed learning rate which was higher and lower than the base rates for adaptive learning rate, decoder with the same size as encoder and other. The masking ratio was also lowered considerably to 50% to see if it would alter the results. None showed results where the model was able to predict clinically significant lesions. Another approach could have been object detection and segmentation to see if the model could have produced better results.

Classification of lesion in the prostate is not directly comparable with the evaluation performed on ImageNet, and other data set available to measure the evaluation on deep learning models for computer vision. The prostate is small, and covering a very tiny part of the image. There is usually much more information and features defining different animals and vehicles in images. Here we are searching for small abnormalities in a small area of the image. This would suggest the requirement of more data and larger model.

7.4 Limitations

7.4.1 Model size

One of the biggest limitations was the available memory on google colab. Colab is a very nice and free tool for machine learning. You can upload a jupyter notebook directly into google lab and code and print out results runtime while having access to gpu or tpu. Since it is free, it have restrictions on runtime and memory. It is possible to upgrade to a paid subscription for better hardware, more memory, and increased run time. This option is unfortunately not available in Norway. Due to the limited memory, there were several limitations to the model. The width of the layers was narrow, and the projected dimension of the patches was small. The image size also had to be downsized by a factor of three to run the model.

7.4.2 Data Set Size

The data set contains ground truth for 204 individual patients, with 300 lesions distributed on 334 two-dimensional slices. There should probably be ground truth in the order of 5000 labeled slices to achieve acceptable results [17]. The data set contains 256 slices with insignificant lesions and only 78 slices that are clinically significant. This is most likely far from a reasonable size data set. Evaluating the model on validation and test set produces different results. Both the sets are very small, with the validation set only containing 10% and the test set containing 20% of an already small data set.

The data set is also very unbalanced and has an unequal distribution between classifications. This unbalanced data set is probably culminating in inadequate predictive performance, especially for the clinically significant class, which we have seen from the results. The clinically significant images only represent 23.36% of the data.

7.4.3 Other Approaches with 2.5D and 3D MAE

The problem was tried to be approached with a 2.5-dimensional and 3-dimensional approach. For the 2.5D approach, all the slices for one patient are fed as input to the model instead of one and one slice. The implementation proved challenging to implement.

The positional encoding was the difficult part. For one slice, the image is divided into patches, and masking is performed by random indices from a uniform distribution. When extracting patches from a volume and applying the same approach, the distribution of masking is not equal on each slice. An algorithm for distributing the masking equally was implemented with the help of numpy, supplying the required tools, and not via tensorflow. The distribution of masked patches was correct, but the model could not run on GPU with this fix. Another approach is 3D MAE. The goal would then be to get the depth information from the slices by tubelet embedding. Instead of looking at the temporal information which is applicable for video we want to extract the depth information of lesion through several slices. The patches were extracted by 3D convolution, but it was not time to implement the full 3D model.

Chapter 8

Conclusion and Future Direction

This thesis explores the potential to use masked autoencoders to classify lesions in T2-weighted MRIs of the prostate. The backbone of the model is ViT, and it was implemented as an autoencoder for pre-training and a downstream model for the classification of labeled images. The images are masked with a high masking ratio to limit the input to the model instead of limiting the model itself. A high masking ratio is also chosen to challenge the model into learning useful features in the latent space of the decoder, and not being able to utilize redundancy in the image to learn an easy way to reconstruct the image.

Several experiments were conducted to find the configuration, hyperparameters, image pre-processing, and augmentation to generate the best possible data and train a model that can predict prostate cancer in an MRI image. Much time was spent on trying to implement a functional, 3-dimension masked autoencoder. This was also a reason for choosing google colab as the GPU hardware. It is very easy to test the implementation on small models and data sets with the option to print out results during training and edit code during runtime. With valuable time passing, the model had to be completed as a 2-dimensional masked autoencoder. The results from the experiments show that a small, masked autoencoder with limiting data set is not a viable option for lesion classification. The proposed method could not be configured to predict clinically significant lesions.

The final classification score had an accuracy of 76%. The high accuracy comes from the small amount of clinically significant slices in the test set. Predicting insignificant on all slices would give an accuracy of over 70%. The bad performance can be measured through

the other metrics. Best AUC score of 46% means that the model can not distinguish between the classes. Precision and recall score of 0.285 and 0.125 shows how few true positive the model was able to predict. The thesis results suggest that bigger models when it comes to depth and width is required. The thesis doesn't give solid results on if the pre-training has an impact. The result was similar for a pre-trained downstream model and a model initialized with random weights. It is hard to conclude on the size of the data set, but it would require a bigger set to compare the results.

8.1 Future Directions

Transformers have become a popular topic within deep learning for NLP and machine vision, and it is still improving. The architecture is very young and was first introduced in 2017. For machine vision, it was introduced only in 2020, and there has been good progress in only a few years. The architecture has proven to compete with CNN and recurrent neural networks, and in some cases, outperformed. The future direction would be to test a full-size model on the scale of ViT-Basic or ViT-Huge to see if it would improve the results.

Another interesting approach would be to test a 3-dimensional masked autoencoder on the prostate MRI volume. Tublets can be extracted from all the slices of the patients to gain depth knowledge on top of the 2-dimensional positional encoding for a slice. There are working examples for video ViT, and lesions may be more visible for the model over several slices.

The model is only considering one mode of the prostate MRI, the axial T2-weighted images. The data set contains images of other modes like proton density-weighted, dynamic contrast-enhanced, and diffusion-weighted images. Looking into a multi-modal architecture would automatically increase the size of the data set. In the actual ProstateX challenge, you can apply all the MRI modes for both classification and segmentation. The current leader-board have an AUC score of 0.95.

List of Figures

2.1	Prostate Anatomy [7]	6
2.2	Digital rectal exam [7]	8
2.3	Transrectal ultrasound guided biopsy [7]	8
2.4	Illustration of Glason score for different cancer cell patterns [15]	10
3.1	Illustration of neural network with one input layer, one hidden layer and one output layer. Each circular node represents an artificial neuron and an arrow represents a connection from the output of one artificial neuron to the input of another [18]	12
3.2	Input, activation and output of a neuron	12
3.3	Function $f(x)$ including one global minimum and several local minimums	15
3.4	Sigmoid activation function	18
3.5	GELU and ReLU activation function	19
3.6	GELU and ReLU activation function	20
3.7	Residual learning building block. <i>The figure is reprinted in unaltered form from the paper written by Kaiming He et al. named "Deep residual learning for image recognition" [31]</i>	20
3.8	Mathematical illustration of autoencoder	22
3.9	Illustration of autoencoder [32]	22
3.10	Computational graph for denoising autoencoder	23
3.11	Illustration of the original transformer architecture. <i>The figure is reprinted in unaltered form from the original paper written by Ashish Vaswani et al. named "Attention is all you need" [37]</i>	25
3.12	Multi-head attention with h heads. <i>The figure is reprinted in unaltered form from the original paper written by Ashish Vaswani et al. named "Attention is all you need" [37]</i>	26
3.13	Positional Embedding	27
3.14	Visual transformer architecture. <i>The figure is reprinted in unaltered form from the original paper written by Alexey Dosovitskiy et al. named "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" [39]</i>	28
3.15	Uniform frame sampling from video frames	29
3.16	Tubelet embedding from video frames	29
3.17	MAE architecture [41]. Published under free license.	30
3.18	Masking of video frame [42]. Published under open license.	32
4.1	T2-weighted MRI slice of Prostate	41
4.2	Visualize a random slice from the original dataset without any filter or normalization	43

4.3	Visualize a random slice from the original data set with grayscale filtering and outlier removal filter	44
5.1	Model architecture for the autoencoder	45
5.2	Original image at top, patched image in middle and reconstructed image at bottom	46
5.3	Patch embedding for downstream classification task	47
5.4	Masked patch embedding for pre-training	48
5.5	Encoder architecture	48
5.6	Decoder architecture	49
5.7	Decoder with additional linear probing	50
5.8	Adaptive learning rate with warmup and cosine decay	54
6.1	Loss during training for model 1	59
6.2	Model 1 - Reconstructed image for epoch 1 at top row and epoch 746 at bottom	60
6.3	Loss during training for model 2	61
6.4	Reconstructed image for epoch 1 at top row and epoch 176 at bottom	62
6.5	Model 2 - Reconstructed image for epoch 10 at top row and epoch 111 at bottom	62
6.6	Loss during training of downstream model 1	63
6.7	Accuracy during training of downstream model 1	64
6.8	AUC during training of downstream model 1	64
6.9	AUC during training of downstream model 1	66
6.10	Reproduced image when cropping around prostate gland. Prostate gland marked in red.	67

List of Tables

2.1	Suggested age specific normal values of PSA [9]	7
2.2	Gleason score and ISUP-Grading [14]	10
3.1	Confusion matrix with n=2 classes	34
4.1	Detailed description of PROSTATEx Challenge Data Set	40
4.2	Distribution of lesion findings, anatomical zone location and classification with respect to clinically significant or not in the prostate data set	42
4.3	Final ground truth distribution of the data	42
5.1	Number of parameters for autoencoder with image size = 128x128, patch size = 16x16, encoder projected dimension = 128, decoder projected dimension = 64, number of heads = 4, number of layers for encoder = 6, and number of layers for decoder = 2	52
5.2	Number of parameters for downstream model with image size = 128x128, patch size = 16x16, encoder projected dimension = 128, number of heads = 4, number of layers for encoder = 6	52
6.1	Constant hyperparameters for all simulations	59
6.2	Hyperparameters for simulation 1	59
6.3	Hyperparameters for simulation 1	61
6.4	Downstream model 1 - Hyperparameters	63
6.5	Downstream model 1 - performance score	64
6.6	Confusion matrix for downstream model 1	65
6.7	Downstream model 1 - performance score with 3 trainable layers and initialized from start	65
6.8	Downstream model 2 - Hyperparameters	66
6.9	Downstream model 2 - performance score with 3 trainable layers and initialized from start	66

Bibliography

- [1] World health organization. prostate. <https://gco.iarc.fr/today/data/factsheets/cancers/27-Prostate-fact-sheet.pdf>. Accessed: 2022-04-27.
- [2] MaryBeth B Culp, Isabelle Soerjomataram, Jason A Efstathiou, Freddie Bray, and Ahmedin Jemal. Recent global patterns in prostate cancer incidence and mortality rates. *European urology*, 77(1):38–52, 2020.
- [3] Dragan Ilic, Molly M Neuberger, Mia Djulbegovic, and Philipp Dahm. Screening for prostate cancer. *Cochrane Database of Systematic Reviews*, (1), 2013.
- [4] Veeru Kasivisvanathan, Antti S Rannikko, Marcelo Borghi, Valeria Panebianco, Lance A Mynderse, Markku H Vaarala, Alberto Briganti, Lars Budäus, Giles Hellewell, Richard G Hindley, et al. Mri-targeted or standard biopsy for prostate-cancer diagnosis. *New England Journal of Medicine*, 378(19):1767–1777, 2018.
- [5] Maarten de Rooij, Esther HJ Hamoen, J Alfred Witjes, Jelle O Barentsz, and Maroeska M Rovers. Accuracy of magnetic resonance imaging for local staging of prostate cancer: a diagnostic meta-analysis. *European urology*, 70(2):233–245, 2016.
- [6] Jessica Carlsson, Gisela Helenius, Mats G Karlsson, Ove Andrén, Karin Klinga-Levan, and Björn Olsson. Differences in microrna expression during tumor development in the transition and peripheral zones of the prostate. *BMC cancer*, 13(1):1–11, 2013.
- [7] American urological association. medical student curriculum: Prostate cancer screening and management. <https://www.auanet.org/education/auauniversity/for-medical-students/medical-students-curriculum/medical-student-curriculum/prostate-cancer/psa>. Accessed: 2022-04-27.
- [8] JF Myrtle, PG Klimley, LP Ivor, et al. Advances in cancer diagnostics. *San Diego, CA: Hybritech Inc*, pages 1–4, 1986.

- [9] Mohan Adhyam and Anish Kumar Gupta. A review on the clinical utility of psa in cancer prostate. *Indian journal of surgical oncology*, 3(2):120–129, 2012.
- [10] Prostate cancer uk: Prostate biopsy. <https://prostatecanceruk.org/prostate-information/prostate-tests/prostate-biopsy>. Accessed: 2022-05-06.
- [11] Hashim U Ahmed, Alex Kirkham, Manit Arya, Rowland Illing, Alex Freeman, Clare Allen, and Mark Emberton. Is it time to consider a role for mri before prostate biopsy? *Nature reviews Clinical oncology*, 6(4):197–206, 2009.
- [12] Baris Turkbey, Andrew B Rosenkrantz, Masoom A Haider, Anwar R Padhani, Geert Villeirs, Katarzyna J Macura, Clare M Tempany, Peter L Choyke, Francois Cornud, Daniel J Margolis, et al. Prostate imaging reporting and data system version 2.1: 2019 update of prostate imaging reporting and data system version 2. *European urology*, 76(3):340–351, 2019.
- [13] Brett Delahunt, Rose J Miller, John R Srigley, Andrew J Evans, and Hemamali Samaratunga. Gleason grading: past, present and future. *Histopathology*, 60(1):75–86, 2012.
- [14] Jonathan I Epstein, Lars Egevad, Mahul B Amin, Brett Delahunt, John R Srigley, and Peter A Humphrey. The 2014 international society of urological pathology (isup) consensus conference on gleason grading of prostatic carcinoma. *The American journal of surgical pathology*, 40(2):244–252, 2016.
- [15] Irish cancer society. staging and grading prostate cancer. <https://www.cancer.ie/cancer-information-and-support/cancer-types/prostate-cancer/staging-and-grading-prostate-cancer>. Accessed: 2022-05-07.
- [16] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- [17] Ian Goodfellow. Deep learning, 2017.
- [18] Wikipedia contributors. Artificial neural network — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Artificial_neural_network&oldid=1084355594, 2022. [Online; accessed 9-May-2022].

- [19] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
- [20] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014.
- [21] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.
- [22] Geoffrey Hinton, Nitish Srivastava and Kevin Swersky. Neural networks for machine learning, lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, 2012.
- [23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [24] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *towards data science*, 6(12):310–316, 2017.
- [25] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.
- [26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.
- [27] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.
- [28] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [29] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.
- [30] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

- [31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [32] Francois Chollet. The keras blog, building autoencoders in keras. <https://blog.keras.io/building-autoencoders-in-keras.html>, 2016. Accessed: 2022-03-14.
- [33] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.
- [34] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.
- [35] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.
- [36] Sara Atito, Muhammad Awais, and Josef Kittler. Sit: Self-supervised vision transformer. *arXiv preprint arXiv:2104.03602*, 2021.
- [37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [38] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.
- [39] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- [40] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6836–6846, 2021.
- [41] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377*, 2021.

- [42] Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. Videomae: Masked autoencoders are data-efficient learners for self-supervised video pre-training. *arXiv preprint arXiv:2203.12602*, 2022.
- [43] Mohammad Hossin and Md Nasir Sulaiman. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2):1, 2015.
- [44] Jin Huang and Charles X Ling. Using auc and accuracy in evaluating learning algorithms. *IEEE Transactions on knowledge and Data Engineering*, 17(3):299–310, 2005.
- [45] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <https://www.tensorflow.org/>. Software available from tensorflow.org.
- [46] François Chollet et al. Keras. <https://keras.io>, 2015. Accessed: 2022-05-17.
- [47] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL <https://doi.org/10.1038/s41586-020-2649-2>.
- [48] G. Bradski. The OpenCV Library. *Dr. Dobb’s Journal of Software Tools*, 2000.
- [49] Geert Litjens, Oscar Debats, Jelle Barentsz, Nico Karssemeijer, and Henkjan Huisman. Prostatex challenge data, 2017.

- [50] Geert Litjens, Oscar Debats, Jelle Barentsz, Nico Karssemeijer, and Henkjan Huisman. Computer-aided detection of prostate cancer in mri. *IEEE transactions on medical imaging*, 33(5):1083–1092, 2014.
- [51] Tracy Nolan. Spie-aapm-nci prostatex challenges - the cancer imaging archive (tcia) public access - cancer imaging archive wiki. <https://wiki.cancerimagingarchive.net/pages/viewpage.action?pageId=23691656#23691656e9a7bb92dcee4b419511436cc3a364b3>, 2021.
- [52] Roy Aritra Gosthipaty and Paul Sayak. Masked image modeling with autoencoders. https://keras.io/examples/vision/masked_image_modeling/, 2021. Accessed: 2022-02-04.
- [53] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
- [54] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430, 2015.
- [55] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.
- [56] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.