LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

FMAM05 Degree Project in Mathematics for Engineers, 2018

Automatic Gleason Classification of Prostate Cancer – Classification of Small Regions

Author: Kasper Tall

Advisor: Anders Heyden

Assistant advisor: Ida Arvidsson

Abstract

Title: Automatic Gleason Classification of Prostate Cancer – Classification of Small Regions

Author: Kasper Tall; **Advisor:** Anders Heyden; **Assistant advisor:** Ida Arvidsson

Purpose: To classify the severity of a case of prostate cancer, physicians use the 10-grade Gleason score. The purpose of this Master's thesis is to study how small dimensions of image crops affect the Gleason 5-classification capability of a machine learning system. In this thesis, two aspects of dimensionality have been taken into account when creating image crops, the image crop size and the degree of magnification.

Methodology: 70 x 70 and 128 x 128 pixel images, both with a 40X magnification, were cropped from larger tissue images annotated at Skåne University Hospital (SUS), creating one data set for each image crop size. The networks trained on these data sets were as follows: a CNN-architecture, a CNN-architecture with an Inception-v4-module at the end, a ResNet-architecture, and a CNN-architecture with an Inception-ResNet-v1-module at the end.

Results: The ResNet-architectures performed the best on the created data sets, achieving mean 5-fold cross-validation accuracies of 91.9% and 96.5 % for the 70 x 70 and 128 x 128 pixel images respectively. However, these architectures experienced temporary drops in accuracy. Furthermore, the modified CNN-networks could not be determined to definitely outperform the base CNN-networks.

Conclusion: The results indicated that image crops of sizes larger than 70 x 70 when using a magnification of 40X were preferable for PCa-classification purposes. However, the classification effects of using different architecture designs were inconclusive.

Keywords: Prostate cancer, Gleason grading, CNN, Inception, ResNet, Inception-ResNet

Populärvetenskaplig sammanfattning

I medicinsk bildanalys används bilder föreställande bland annat cellprover för att träna program att känna igen vissa mönster, så kallad maskininlärning. Programmen, som kan variera i sin design, kan därefter användas för att automatiskt avgöra om prover visar på förekomst av allvarliga sjukdomar, så kallad klassificering. Programdesignen som används kallas även för arkitektur. Prostatacancer, en av de vanligaste formerna av cancer hos män, har tidigare studerats med hjälp av många olika maskininlärningstekniker. Men hur stora bilder behöver man egentligen för att träna sådana program? Denna fråga är värd att besvara för att kunna minska minnesåtgången och träningstiden för maskinlärningsprogram.

I detta examensarbete har automatisk mönsterigenkänning av prostatacancerbilder av olika storlekar med 40 gångers förstoring studerats. De bilder som har använts har bestått av infärgade snitt av biopsiprover. Prostatacancerprover extraherade genom nålprover bedöms på en femgradig skala. Som ett första steg skapades därför mindre urklipp ur större prostatacancerbilder med olika grader av cancerspridning. Dessa bilder delades in i två kategorier, Gleason 5 (den mest elakartade typen av prostatacancer), respektive icke-Gleason 5. För att träna maskininlärningsprogram med olika design på olika sorters bilder skapades även roterade och speglade varianter av urklippen. I syfte att undersöka storlekens påverkan på automatisk prostatacancergradering skapades uppsättningar av bilder med storlek 70 x 70 respektive 128 x 128 pixlar.

Maskininlärningsdesign kan anta olika form beroende på vad dess skapare vill att programmen skall fokusera på i bilderna. I detta examensarbete skapades arkitekturer baserade på ett flertal olika tekniker. Till vissa av dessa arkitekturer har även moduler baserade på andra tekniker lagts till och därigenom kombinerat olika tekniker. Metoder som har använts är bland annat detektering av bildmönster av olika omfång i samma bild samt tekniker för att förenkla mönsterigenkänningsprocessen. Totalt skapades sju olika sorters arkitekturer, baserade på fyra sorters maskininlärningsmetodologier. Varje arkitektur var specialdesignad för bilder av en viss storlek, förutom en arkitektur som kunde användas både för 70 x 70 och 128 x 128 pixels urklipp.

Testresultaten för programmen visade att arkitekturer med större tränings- och testbilder uppnådde högre klassificeringsnoggrannhet. Dock kunde det ej bevisas att någon arkitektur garanterat presterade bättre än de andra. Anledningen till detta berodde på att de modifierade arkitekturerna uppvisade stora svängningar i noggrannhet under den tid då de tränades (testperioden). Arkitekturen med högst klassificeringsnoggrannhet var även instabilt då det led av kraftiga temporära minskningar i klassificeringsnoggrannheten. Undersökningar av felklassificerade bilder för de bästa arkitekturerna visade att dessa arkitekturer hade problem att klassificera bilder med låg kontrast eller med tätt intilliggande celler.

I medicinska bilder kan det förekomma variationer i hur elakartade de cellulära mönstren är. Genom att skapa små bilder kan bilderna komma att spegla andra mönster än de som de ursprungliga bilderna är klassificerade som. Genom att träna arkitekturer på mindre bilder kan dessa arkitekturer därför komma att förknippa fel mönster med en viss sorts klassificering. Resultaten av detta examensarbete indikerar att användning av bilder större än 70 x 70 kan vara bättre lämpade för prostatacancerklassificering. Vad det gäller hur arkitekturdesign bäst anpassas till små bilder var resultaten dock oklara. Hur små bilder som kan användas för att

träna arkitekturer för att uppnå goda klassificeringsresultat samt vilken arkitekturdesign som ger bäst klassificeringsresultat återstår för framtida forskare att avgöra.

Acknowledgments

I would like to express my thanks to my supervisor Anders Heyden, and my assistant supervisor Ida Arvidsson for their advice, feedback, and help in procuring the research data. Without their help, the end results wouldn't have been the same. I'm also grateful for Carl-Gustaf Werner's help with the, at times obstinate, Ubuntu environment. Many thanks also go to Axel Nyström for his insightful feedback on the thesis. Finally, I wish to thank my parents for being supportive of me during my university studies.

Kasper Tall

Malmö, August 2018

Abbreviations

ANN: Artificial neural network

CNN: Convolutional neural network

DOGS: Digital Pathology for Optimized Gleason Score in Prostate Cancer

H&E: Hematoxylin and Eosin

MLP: Multi-layer perceptron

PCa: Prostate cancer

ReLU: Rectified Linear Unit

ResNet: Residual neural network

SGD: Stochastic gradient descent

SUS: Skåne University Hospital

SVM: Support vector machine

Table of Contents

1 Introduction

1.1 Background

Prostate cancer (PCa) is one of the forms of cancer that affects the most number of men in the world [1, 2, 3]. To grade how serious the occurrence of PCa is, the originally 10-grade Gleason scale is used. The modern version Gleason scale only includes scores between 6 and 10. The grading in needle biopsies is determined as the sum of the most common and the highest scored PCa-patterns. These patterns were in the past scored on a scale from 1 to 5, where 1 indicated the most benign cancer-pattern, and 5 indicates the most malignant cancer-pattern [4, 5]. Studies have, however, showed that the assessment of the Gleason score for the same PCa-sample varies between assessors [6]. For the purpose of developing a tool for automatic assessment of Gleason scores for PCa, Sweden's innovation agency, Vinnova, is sponsoring research in computerized image analysis of Gleason grades in prostate biopsies. This research is done in the project "Digital Pathology for Optimized Gleason Score in Prostate Cancer" (DOGS). The DOGS-project is coordinated by the Institute for translational medicine at Lund University [7].

In image analysis studies of PCa, research has been done in areas such as registration, segmentation and classification, where Support vector machines (SVM), Random Forest and convolutional neural networks (CNN) have been used as classification systems [8, 9, 10, 11, 12, 13, 14]. For SVMs, transfer learning has also been used in architecture construction [10].

Within the DOGS-project, earlier master's theses at Lund University have been centered on how Gleason score classification can be done with CNN-systems, with, and without the use of earlier segmentation in the classification process [15, 16, 17]. In classification studies using CNNs it is important to determine the dimensions of the images that a system is to be trained and tested on. Litjens et al. noted that small training images resulted in worse classification capability than if larger images were used [12]. Gummeson et al. has brought up classification of small regions as a potential area for future research [13].

1.2 Purpose

The purpose of this thesis is to investigate how Gleason classification of PCa with the use of artificial neural networks (ANN) is affected by the size of histopathological images used for training and testing. In particular, we are to study how the classification of PCa with Gleason 5 patterns is affected by the use of histopathological images of small dimensions.

1.3 Delimitations

In this master's thesis, the histopathological images will come from only one source, Skåne University Hospital (SUS). Tissue images from different sources may differ with regard to appearance, a matter which will not be investigated in this thesis. The dimensionality aspect of small histopathological images will be studied by creating image crops with a large magnification factor (40X magnification), and by delimiting the size of said image crops. Any studies into the effects of contrast or color manipulation will not be undertaken. Furthermore, results of architecture tests will not be compared to those of any earlier PCa classification studies. The reasons for this is that the data sets used in this thesis have not been used in any previous study.

1.4 Thesis sections

This thesis will be divided in the following sections: ***Medical theory***, ***Machine learning theory***, ***Methodology***, ***Results***, and finally ***Discussion***.

In the section ***Medical theory***, the anatomy of the prostate and how PCa is classified will be briefly introduced. In the section ***Machine learning theory***, the theory behind machine learning in general and the architectures developed in this thesis in particular will be presented. In the section ***Methodology***, the development environments used in this thesis will be explained, as well as how the data sets and architectures were constructed. The details of the constructed architectures will also be explained in this section. In the section ***Results***, the results obtained when training and testing the constructed architectures on the data sets will be presented. Finally, in the section ***Discussion***, the data sets and training-test results for the constructed architectures will be discussed, and suggestions for future research will be presented.

2 Medical theory

2.1 The prostate

The prostate gland, which is the largest of the human male's five accessory sex glands, is itself composed of between 30 and 50 tubuloalveolar glands [18, 19, 20, 21, 22, 23]. The prostate consists to two-thirds of smaller glands and to one-third of fibromuscular stroma [19, 22]. The prostate is, however, not very large, with its size comparable to that of a walnut and with a weight of approximately 20 g [21, 23, 24, 25]. The prostate is intersected by the urethra into which it secretes an alkaline fluid containing, amongst other things, proteolytic enzymes, and lipids. This secretion constitutes approximately 20 % of the male semen [18, 19, 20, 26, 27, 28, 29, 21, 24, 23]. It has been suggested that the alkaline fluid of the prostate might increase the motility of sperm by counter-acting acids present in seminal fluid [27].

2.2 Prostate cancer

In order to detect cellular structures in biopsy samples, the samples can be treated with different stains [19, 21, 23]. One stain used with PCa-samples is Hematoxylin and Eosin (H&E) [8]. In tissue samples stained with H&E, nuclei are colored blue, cytoplasm is colored red or pink, and muscles and collagen fibers are colored pink [19].

The 10-grade Gleason scale is used to determine the severity of PCa. Contemporary Gleason grading is, however, performed on a scale from 6-10. In needle biopsies, the sum of the most common and the highest PCa-patterns are added together to determine the Gleason score. Individual patterns were originally graded on a scale from 1 to 5, with the pattern score increasing the more malign the cancer pattern is (see Figure 1) [4, 5]. Large PCa tumors might spread from the prostate to the seminal vesicles or bladder. Lymph node, skeleton and vascular metastases might also occur [24, 25, 26].

Gleason patterns graded 1 or 2, have round shapes, with variations for grade 2 patterns, and are located close to each other. For patterns graded 1, the edge of the cluster of cells is definite, while patterns graded 2, have loose cell cluster edges. These two patterns are uncommon. Gleason 3 patterns, the most common Gleason pattern, are shape-wise irregular, with occurrences of small glands. This pattern may also include cribriform epithelium. As for the distribution, Gleason 3 patterns have irregular spaces between pattern occurrences and surround normal tissue structures. The pattern edges range from poorly defined to non-existent [30].

Gleason 4 patterns are recognizable by sub-patterns, such as occurrences of large clear cells, tumor cells in stroma, and ragged parts of fused glandular epithelium. Cancer patterns with this score can be found infiltrating normal tissue structures. As for Gleason 5 patterns, they appear as cribriform or solid masses or as anaplastic glands with vacuoles. Gleason 5 patterns can be found intertwined with stroma [30].

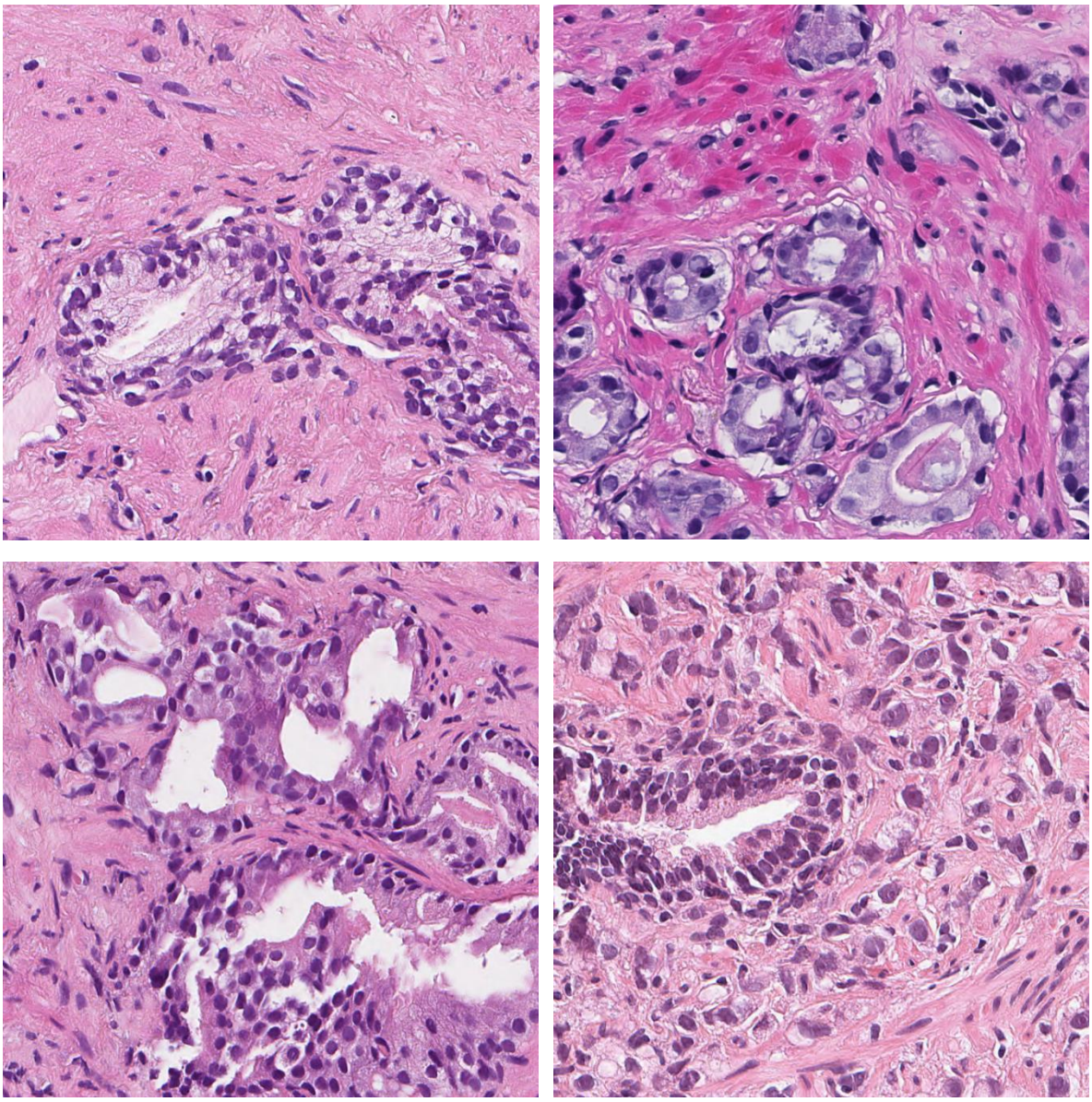


Figure 1. Gleason tissue image examples. (Upper left) Benign Gleason tissue image example, (Upper right) Gleason 3 tissue image example, (Lower left) Gleason 4 tissue image example, (Lower right) and Gleason 5 tissue image example.

3 Machine learning theory

3.1 Artificial neural networks

Some early attempts at machine learning were inspired by how the brain learns. The cellular units used for biological computation in the brain are called neurons. To denote the area of machine learning, the term artificial neural networks (ANN) has been used in the past [31, 32, 33]. In machine learning, the term neuron is taken to mean an artificial computational unit. The mathematical neural model is as follows:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \quad (1)$$

$$y_k = \varphi(u_k + b_k) \quad (2)$$

In definition (1), j, k, x_j, m, and w_{kj} denote a specific connection to neuron, a specific neuron, the input signal from a certain neuron connection to a certain neuron, the total number of input signals, and the strength of a certain neuron connection, i.e. the weight, respectively. u_k denotes the sum of the products of all input signals and the respective weights applied to them. In definition (2), φ, and b_k denote how the amplitude of a certain neuron's output is to be limited, i.e. the activation function, and how the net input to the activation function is to be increased or decreased, i.e. the bias. Finally, y_k denotes the output signal from a certain neuron [32, 33, 34, 35].

Networks consisting of several neurons, so-called multi-layer perceptrons (MLP), consist of an input layer, hidden layers, and an output layer. In these neural networks, each neuron in one layer is often connected to all other neurons in the next layer. Intermediate layers in a neural network are referred to as hidden layers (see Figure 2). The neural network output layer, the final layer, is used for representing the scores for the different classes an object can be classified as. For this reason, activation functions are sometimes not performed on the output layer [33, 36].

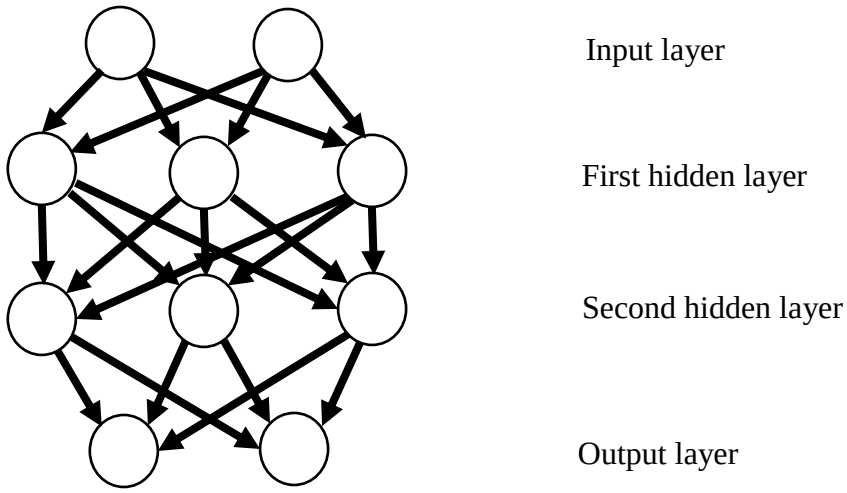


Figure 2. Multi-Layer Perceptron with two hidden layers.

3.2 Architecture construction

3.2.1 Miscellaneous architecture components

To determine how a neural network is meant to work, different settings, so-called hyper parameters, are used [31, 37]. One important hyper parameter is the learning rate, which determines the size of the algorithm's update-steps [31, 35]. Furthermore, when training a neural network architecture, it is important to decide on how large sets input data should be processed in, i.e. batch size [35]. To measure the number of times a piece of training data has been shown to a neural network architecture, the term epoch is used [38]. When connecting different layers in a neural network architecture, outputs from the layers can also be combined in various ways, e.g. through addition or concatenation [39].

3.2.2 Activation functions

Various activation functions can be used in order to perform neural network activations. The Rectified Linear Unit (ReLU) activation function returns the maximum of 0 and the input, i.e.:

$$\max(0, x) \tag{3}$$

In (3), x denotes the input [40, 35]. Another activation function is the logistic sigmoid activation function, which returns a real value between 0 and 1, defined as follows:

$$\frac{1}{(1+e^{-a})} \tag{4}$$

In (4), a denotes the sigmoid function slope parameter [32, 35, 36].

The softmax activation function can be used when the goal is to obtain the probabilities for more than two classes of objects. This activation function normalizes a real-valued score vector with regard to all real-valued score vectors, thereby producing a classification probability score between 0 and 1 for a class. The class which obtains the highest score is the most likely to describe the object being classified by the neural learning architecture in question. The softmax activation function is defined in the following way:

$$\frac{e^{W_i x+b_i}}{\sum_j e^{W_j x+b_j}} \tag{5}$$

In the above activation function definition, W_i, x, and b_i denote the weight matrix for object class i, the input vector, and the bias vector for object class i respectively. Likewise, j, W_j, and b_j denote the total number of object classes, the weight matrix for each object class, and the bias vector for each object class respectively [35, 36, 41].

3.2.3 Loss functions and optimizers

When training a neural network we want to minimize the value of the loss function, also known by other names such as the objective function or cost function. The loss function is a measure of how well network parameters, i.e. weights, lead to good results with respect to fixed labels for the training data. The concept of updating the weights such that the loss function is minimized goes by the name optimization and is performed with the help of optimizers. An example of a loss function is the cross-entropy loss function:

$$C = -\frac{1}{n}\sum_x \sum_j \left[y_j \ln a_j^L + \left(1 - y_j\right) \ln(1 - a_j^L)\right] \quad (6)$$

In the above definition, the value of the cross-entropy loss-function (C) is calculated for a network with multiple neurons in multiple layers. Input variables in the function are denoted as x. Furthermore, the notations y_j, and a_j^L denote the value we want the output layer neurons to take on and the values the output layers actually take on respectively. Finally, the notation n is used to denote the number of training data items [31, 41, 42, 43].

To determine how to update weights we can compute the gradient of the loss function. The steeper the descent, the lower the loss. The technique of evaluating the gradient and then updating the weights is called gradient descent [31, 42, 43]. Instead of performing gradient descent for all training inputs another technique called stochastic gradient descent (SGD) can be employed. This technique is performed by estimating the gradient based on a limited random selection of training inputs, a so-called mini-batch, thereby speeding up the process.

SGD can be expressed in the following way:

$$\nabla C \approx \frac{1}{m}\sum_{j=1}^{m} \nabla C_{X_j} \quad (7)$$

In the above expression, ∇C, m, j, and ∇C_{X_j} denote the gradient, the total number of randomly chosen inputs, the notation for a specific batch, and the gradient for a specific mini-batch respectively [36, 42, 43].

Other than SGD, several other optimizers are available when constructing neural networks [44, 45]. One recently constructed stochastic optimizer is Adam. Among Adam's advantages can be mentioned its capability for use with sparse gradients and the gradient rescaling invariance of its parameter update magnitudes [45]. Adam's update rule, with ε, denoting the relative rounding floating point error, set to 0, is as follows:

$$\Delta_t = \alpha \cdot \hat{m}_t / \left(\sqrt{\hat{v}_t + \varepsilon}\right) \quad (8)$$

In (8), Δ_t, α, $\hat{m}_t$, and $\hat{v}_t$ denote the step taken between two timesteps, the stepsize, the moving average of the gradient, and the moving average of the squared gradient respectively. The stepsize is bounded in the following way:

$$|\Delta_t| < \alpha \cdot (1 - \beta_1)/\sqrt{1 - \beta_2} \text{ for } (1 - \beta_1) > \sqrt{1 - \beta_2}, \text{ and} \quad (9)$$

$$|\Delta_t| < \alpha \text{ in cases when (9) does not apply.} \quad (10)$$

In the above boundary conditions, β_1, and β_2 denote the exponential decay rates for the first and second moment estimates [45].

3.2.4 Back-propagation

In order to calculate the gradient for optimizers, we can use a technique called back-propagation. Back-propagation is performed by first calculating the outputs from neurons in a network, i.e. the so-called forward-pass. The gradients of the neurons in the network are then recursively calculated from the final to the initial layer, i.e. the so-called backward-pass [32, 35, 42, 46]. The algorithm of back-propagation can be expressed in the following way:

1. Determine the activation outputs for the first layer of the network (a^1).
2. Compute the activation outputs from each layer of the network all the way to the final layer (a^L).
3. Compute the output error of the final layer (δ^L).
4. Use the output error to successively compute the output errors for all layers in the network.
5. Compute the gradients for the loss function ($\frac{\partial C}{\partial w_{jk}^l}$ and $\frac{\partial C}{\partial b_j^l}$)

For the back-propagation algorithm, we express activation outputs for a specific layer l as

$$a_j^l = \varphi\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) \quad (11)$$

In (11), w_{jk}^l, a_k^{l-1}, and b_j^l denote the weights between j:th input neuron and the k:th output neuron in the l:th layer of the network, the output activations for the k:th output neuron in the $l-1$:th layer, and the bias for the j:th input neuron in the l:th layer respectively. Furthermore, φ and a_j^l denote the activation function and the activation output for the j:th input neuron. As definition (11) requires initial values for a^1, we need to give this input in step 1. To perform step 2 of the back-propagation algorithm, we use the intermediate expressions:

$$z^l \equiv w^l a^{l-1} + b^l \text{ and} \quad (12)$$

$$a^l = \varphi(z^l) \quad (13)$$

In (12) and (13), z^l denotes the weighted input to the layer l neurons. For step 3 of the back-propagation algorithm we use another expression for support:

$$\delta^L = \nabla_a C \odot \varphi'(z^l) \quad (14)$$

In (14), $\varphi'(z^l)$ and $\nabla_a C$ denote the derivative of the activation of the weighted input to the layer l neurons, i.e. the change of the activation function for z^l, and a vector consisting of the partial derivatives of $\frac{\partial C}{\partial a_j^L}$, i.e. how the loss function changes with respect to the output activations for the j:th input neuron in the output layer, respectively. Finally, δ^L, the output error, is calculated through the element-wise product, i.e. the Hadamard product, of $\nabla_a C$ and $\varphi'(z^l)$. When we backpropagate the output error in step 4: we use the following expression:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \varphi'(z^l) \quad (15)$$

The term $(w^{l+1})^T$ in (15) denotes the transposed weight matrix for layer $l+1$. The effect of (15) is that we use the output error from the next layer in the network in order to calculate the

output error of the preceding layer. In the final step of the back-propagation algorithm we use the following expressions in order to determine the gradients for the loss functions:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{16}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \tag{17}$$

In (16), b_j^l denotes the bias of the j:th input neuron in the l:th layer of the network. Thus, $\frac{\partial C}{\partial b_j^l}$ denotes the change of the loss function with respect to bias in the network. In (17), $\frac{\partial C}{\partial w_{jk}^l}$ denotes the change of the loss function with respect to weights in the network [42]. It is important to note that, when performing back-propagation, the error surface contains both global and local minima. If back-propagation gets stuck in local minima, weight changes will cause a worsening of the loss function [32].

3.2.5 Dropout and batch normalization

A neural network can have problems with learning patterns in data. If a network learns random noise in training data, the network is said to be overfitting [35, 47, 48]. We can modify a neural network such that generalization error of the network, but not the training error, is reduced. Thereby, we introduce what is called regularization to the network. Dropout is a regularization technique used for decreasing overfitting. Dropout is performed by, for a temporary time, selecting a certain percentage of random neurons in a network that are to be active, with the rest of the neurons set to 0. Thus, the term dropout comes from the dropping of non-selected neurons [31, 35, 47, 48].

Batch normalization is another technique which can be used for regularization. It also allows for use of higher learning rates, thereby speeding up the learning process, as well as more latitude when initializing a network [48, 49]. The batch normalizing transform algorithm is defined in the following way:

1. $\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$
2. $\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$
3. $\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$
4. $y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$

In the first step, m, i, and x_i denote the total number of input values, the notation for a specific input value, and a specific input value over a mini-batch (B) respectively. μ_B denotes the mean for a mini-batch. In the second step, σ_B^2 denotes the variance for a mini-batch. In the third step of the algorithm, ϵ denotes a constant used to increase the mini-batch variance's numerical stability. In (3), $\hat{x}_i$ denotes a specific normalized input value. In the final step, γ and β denote parameters used for scaling and shifting the normalized input value. Finally, $BN_{\gamma,\beta}$, and y_i denote the Batch Normalizing Transform, and the linear transformation of a normalized input value respectively [49].

3.3 Convolutional neural networks

Convolutional neural networks (CNN) are a neural networks, which, in contrast to ANNs, which operate over input signals, take images as input. Convolution over a two-dimensional image can be expressed in the following way:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \quad (18)$$

In the above expression, $S(i,j)$, I, and K denote the result of the convolution, the two-dimensional input image, and the weights respectively. Furthermore, m and n denote the extents of the x- and y-input values of an image [31, 34]. With respect to CNNs, weight sets are often referred to as a filter or kernel [34].

CNNs make use of an idea called sparse connectivity. This idea is based on the recognition that interesting parts of an image, so-called features, can be detected with kernels much smaller than the images themselves. The memory needed to store parameters is thus smaller than if larger kernels would have been employed. The hyper parameter determining the size of input region each neuron is to be connected to is called the receptive field [31, 32, 34, 36].

CNN-parameter sharing also allows images to be translationally equivariant, meaning that moving the input image results in the output image being moved equally much [31]. Another idea used in CNNs is parameter sharing, through which only one weight set for the whole image, instead of one weight set per image element, needs to be learned by a CNN. This decreases the memory needed for architecture model storage [31, 32, 34, 35, 36].

3.3.1 Padding and spatial CNN-calculation

When performing operations on image volumes, a technique called zero-padding can be utilized. When using padding, zeros are added around the border of an image volume, which in turn allows us to control the size of spatial outputs [34]. The terms ‘valid’ and ‘same’ are sometimes used in connection with padding. The term ‘valid’ means that no padding should be applied. On the other hand, the term ‘same’ means that padding should be applied in such a way that the length of the output is the same as the length of the input [50, 51].

Convolutional layer output-parameters are calculated in the following manner for an input image volume with width W_1, height H_1, and depth (a third image dimension, e.g. the number of color channels of an image) D_1:

$$W_2 = (W_1 - F + 2P)/S + 1 \quad (19)$$

$$H_2 = (H_1 - F + 2P)/S + 1 \quad (20)$$

$$D_2 = K \quad (21)$$

In the above definitions, the variables F, and P, denote the height and width of a square window used to capture a limited amount of numerical values, known as the spatial extent, and the number of zeroes added next to each adjacent entry of an image matrix respectively. In turn, S, and K, denote the lengths of the steps used to move the square capture window left-to-right, top-to-bottom, i.e. stride, and the number of filters used on an image volume respectively. Finally, W_2, H_2, and D_2 denote the width, height, and depth respectively of the output image volume [34].

3.3.2 Pooling

Pooling is a technique commonly used in CNNs. Pooling is used to substitute the numerical contents of a certain area with a different value, thereby decreasing the computations needed for the next step of image matrix-processing. As pooling summarizes the numerical content of a certain area, pooling translated versions of the specific area produces similar summarized results. Thus, pooling, to a certain degree, makes image matrices translationally invariant [31, 34, 36]. Pooling output-parameters are calculated as follows:

$$W_2 = (W_1 - F)/S + 1 \quad (22)$$

$$H_2 = (H_1 - F)/S + 1 \quad (23)$$

$$D_2 = D_1 \quad (24)$$

In the above definitions, W_1, H_1, and D_1 denote the width, height, and depth respectively of an input image volume. The variables F and S, in turn, denote the spatial extent, and the stride respectively. Finally, W_2, H_2, and D_2 denote the width, height, and depth respectively of the output image volume [34].

Different varieties of pooling exist. For the variant called ‘Max pooling’, the highest numerical value of the values contained in a rectangular area is used to represent the chosen numerical neighborhood as a whole. Likewise, for the pooling variant ‘Average pooling’, the average of the values in a certain rectangular area is used to represent the area’s overall numerical value (see Figures 3-4) [31].

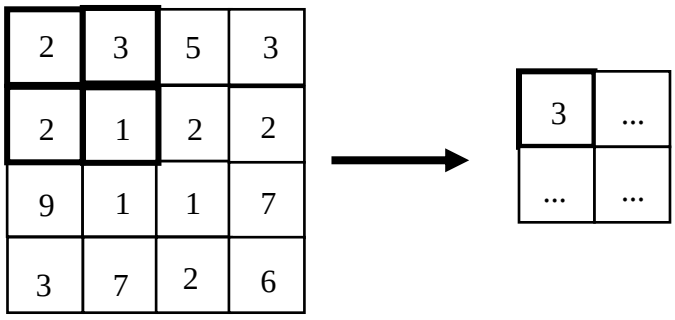


Figure 3. Max pooling example with spatial extent 2 x 2 and stride 2 x 2.

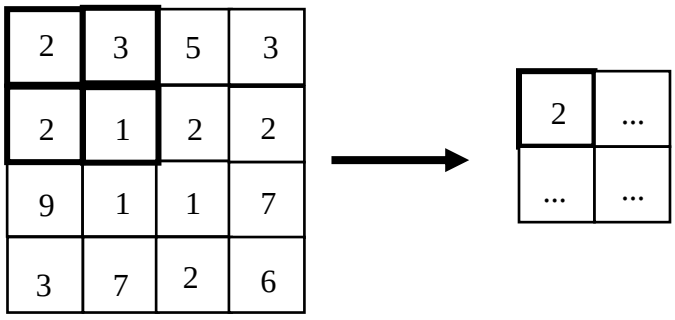
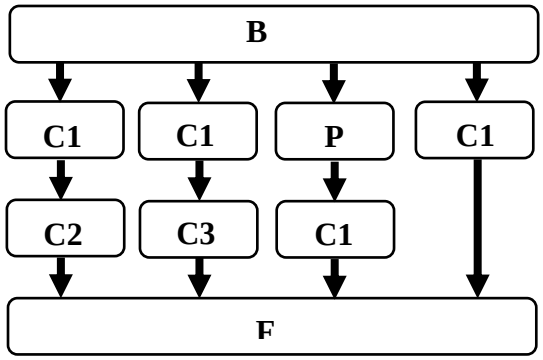


Figure 4. Average pooling example with spatial extent 2 x 2 and stride 2 x 2.

3.4 The Inception- and ResNet-architectures

The Inception architectures are based on modules with convolutional operations. For the original Inception architecture, multiple interesting image clusters were captured by using convolutions of different sizes. However, 1 x 1 convolutional operations were performed before 3 x 3 or 5 x 5 convolutions in order to reduce the dimensions of input signals. A separate pooling track was also included due to earlier successful use of pooling in CNNs (see Figure 5) [52, 53].



B: Base layer.

C1: Convolutional layer. Kernel size: 1 x 1.

C2: Convolutional layer. Kernel size: 3 x 3.

C3: Convolutional layer. Kernel size: 5 x 5.

P: Pooling layer.

F: Filter concatenation layer.

Figure 5. Original Inception-module.

Researchers made the Inception architecture computationally cheaper by factorizing spatial convolutions into smaller convolutions. As an example, a 3 x 3 convolution can be replaced by first using a 3 x 1 convolution and following this up with a 1 x 3 convolution (see Figure 6). This architectural improvement has been included in evolved Inception-based architectures, such as Inception-v4 (see Appendix 1) [51, 53].

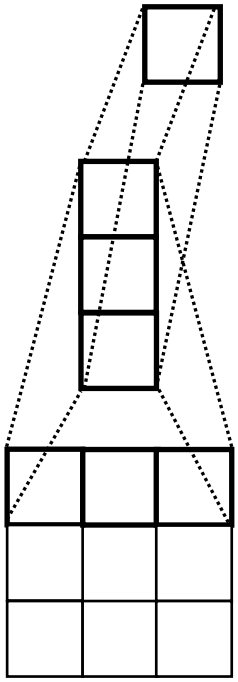
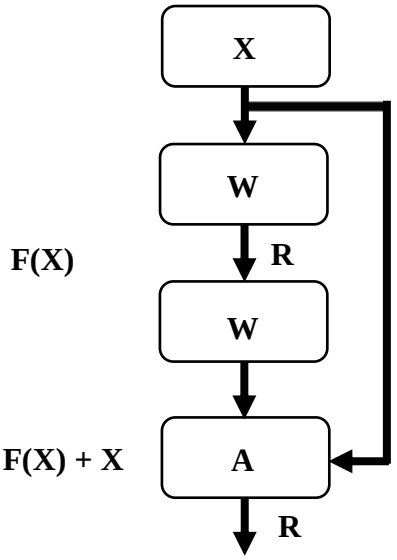


Figure 6. Spatial 3 x 3 convolutional factorization example.

In order to combat the problem of accuracy decreasing in deep networks, the ResNet-architecture was developed. The ResNet-architecture tackles the aforementioned problem by fitting stacked nonlinear layers to a residual mapping. Mathematically, this is expressed in the following way:

$$F(x) := H(x) - x \tag{25}$$

In the above expression, x, $H(x)$, and $F(x)$ denote the identity, the desired underlying mapping, and the residual function respectively. Thus, the original mapping is expressed as $F(x) + x$ (see Figure 7) [54]. The techniques of the Inception- and ResNet-architectures have been combined in the Inception-ResNet-architectures, such as Inception-Resnet-v1. In order to stabilize training, a scaling factor is included in order to scale residuals in Inception-ResNet-modules (see Figure 8 and Appendix 2) [51].



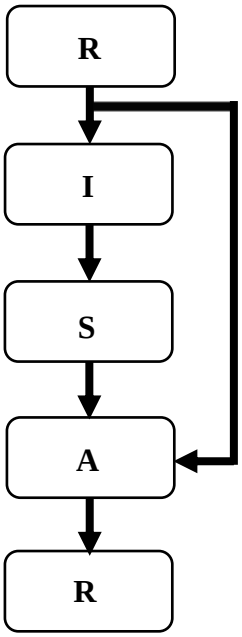
X: Identity.

W: Weight layer.

R: ReLU-activation.

A: Addition layer.

Figure 7. ResNet-architecture residual learning building block.



R: ReLU-activation layer.

I: Inception module.

S: Activation scaling.

A: Addition layer.

Figure 8. Inception-ResNet-module scaling scheme.

3.5 Results analysis

The k-fold cross-validation technique can be utilized for training and testing on a small data set. When using this method the data set is divided into k subsets. Each data point is included in only one of these subsets. For each round of training, one subset is withheld for testing with the other subsets used for training. The number of training-validation runs performed is the same as the number of subsets, i.e. k. To obtain a measurement for the validation accuracy on the data set in question, the average over the validation accuracies across all k-runs can be used [31].

When testing for an illness, where the test results are either positive or negative, the specificity and sensitivity measurements can be used to determine the correctness for each test result in relation to the outcome. The sensitivity measurement is defined as the number of positive test results with positive outcomes in relation to the total number of positive and negative test results with positive outcomes. Likewise, the specificity measurement is defined as the number of negative test results with negative outcomes in relation to the total number of positive and negative test results with negative outcomes [55].

4 Methodology

4.1 Development environment

In this thesis, two separate development environments have been used, a Windows 10 environment and an Ubuntu environment. The purpose of this set-up has primarily been to save time by performing parallel training-test rounds. The Windows environment has been used to construct the architectures described in this thesis, as well as perform training-test rounds. The Ubuntu environment, in turn, has been used primarily to run training-test rounds. Construction, training, and testing of one architecture (ResNet) could only be fully done on the Ubuntu environment as this environment included more powerful GPU and RAM than the Windows environment, which was necessary for storage of the parameters of this architecture.

The Windows environment utilized was Windows 10 Home version 1709, OS Build 16299.371. The hard-ware specifications were as follows: Processor: AMD Ryzen 5 1400 Quad-Core Processor @ 3.20 GHz, 4 Core(s), 8 Logical Processor(s); Graphics card: Nvidia GeForce GTX 1050.

The programming was done in PyCharm 2017.3.3. (Community Edition). The language used for development was Python 3.6. For machine learning purposes, Keras 2.1.4 with Tensorflow-GPU 1.6.0 as back-end and CUDA Toolkit 9.0 were used [56, 57]. The following Python packages were used: imageio 2.2.0 numpy 1.14.1, matplotlib 2.2.0, scikit-learn 0.19.1, and scipy 1.0.0. The MATLAB version used on the Windows environment was MATLAB 9.2.0.556344 (R2017a).

The Ubuntu environment utilized was Ubuntu 16.04.3 LTS. The hard-ware specifications were as follows: Processor: Intel Xeon(R) CPU E5-2640 v3 @ 2.60 GHz x 16, Graphics card: Nvidia GeForce GTX Titan Z. The programming was done in PyCharm 2018.1 (Community Edition). The language used for development was Python 3.5. For machine learning purposes, Keras 2.1.5 with Tensorflow-GPU 1.7.0 as back-end and CUDA Toolkit 9.0.176 were used. The following Python packages were used: imageio 2.3.0 numpy 1.14.2, matplotlib 2.2.2, scikit-learn 0.19.1, and scipy 1.0.1.

4.2 Data sets

The images used to construct the data sets utilized in this thesis were derived from annotated H&E stained PCa tissue samples from Skåne University Hospital (SUS). The images in this data set numbered 2 675 for benign images, 922 for Gleason 3 images, 1 006 for Gleason 4, and 624 for Gleason 5 images in total. These images partly overlap with those used in Dataset A in [58]. Several data sets with image crops of various sizes and dimensions were created for testing purposes. Among these data sets, the four largest data sets were used for longer training-test runs to determine the specific hyper parameters of interesting architectures and to obtain final results for tests on the same architectures. These four largest data sets will be referred to as 9840_Random_70_Test, 9840_Random_128_Test, Fixed_9840_Random_70_Test, and Fixed_9840_Random_128_Test.

All image data sets were composed of PCa image crops with 40X magnification. Each data set consisted of 9840 Gleason 5 image crops, and 9840 non-Gleason 5 image crops. The 9 840 non-Gleason 5 image crops were sub-divided in the following way: 3 280 Gleason 4,

3 280 Gleason 3 image crops, and 3 280 benign image crops. For 9840_Random_70_Test and Fixed_9840_Random_70_Test, image crops with dimensions of 70 x 70 pixels were used. For 9840_Random_128_Test and Fixed_9840_Random_128_Test, image crops with dimensions of 128 x 128 pixels were used.

9840_Random_70_Test, and 9840_Random_128_Test were the first two larger data sets created. Fixed_9840_Random_70_Test, and Fixed_9840_Random_128_Test, were created when it was realized that 9840_Random_70_Test, and 9840_Random_128_Test were erroneously constructed, the errors and correction of which will be further explained later in this section.

4.2.1 Original data sets

Initially, image crops of different dimensions were surveyed. The first image data sets were constructed with image crops of dimensions 70 x 70 pixels as this restriction was deemed suitable for being able to obtain image crops with whole cells within the chosen cropping-frame, if the frame was to be centered on said cells (see Figure 9). Images to crop were randomly selected. The image crops were then created by manually cropping the larger tissue images in MS Paint. Although the areas to be cropped were randomly selected, care was taken to avoid overlap between image crops as well as cropping of tissue background. Automatic creation of the small image crops was deemed as too complicated to implement due to the risk of accidentally cropping only white background or areas surrounding cells, thereby distorting the data sets. Large image crops were initially created such that both the widths and heights of the crops exceeded 70 pixels. This approach was later changed to a second approach which allowed for easier cropping and stricter control of image crop overlap. The second approach was to crop larger image crops such that they did not exceed 104 pixels in width (i.e. one pixel less than the width of one and a half image), but still had a height of 70 pixels or more.

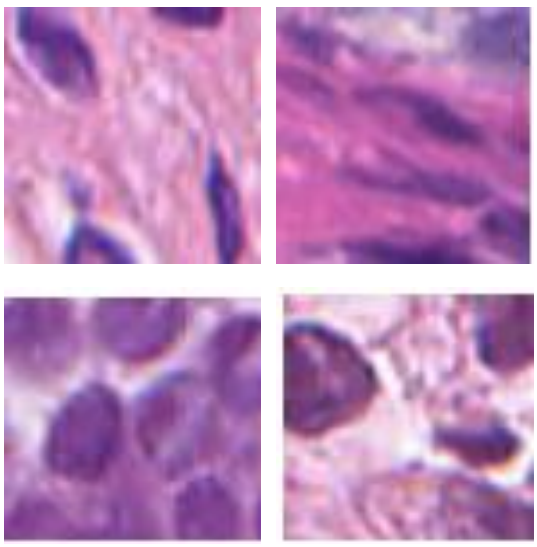


Figure 9. 70 x 70, 40 x magnification image crop examples. (Upper left) benign Gleason image crop example, (Upper right) Gleason 3 image crop example, (Lower left) Gleason 4 image crop example, (Lower right) and Gleason 5 image crop example.

To increase the number of image crops from a larger tissue crop, a sliding-window approach was used. The scripts needed for the image expansion procedure were developed and executed using MATLAB. For each part of the larger tissue crop in the sliding window's focus, the image was rotated 90, 180, and 270 degrees. Furthermore, the non-rotated image, and the 90, 180, and 270 degree rotations were also flipped along the horizontal axis. Thus, from each image crop, seven rotated and/or mirrored variants were created. When an image crop had been rotated and/or mirrored, the sliding window was moved horizontally with a movement corresponding to half the window's size, i.e. 35 pixels for a 70 x 70 pixel image crop. When the larger tissue crop would not cover the entirety of the sliding window, the sliding window would move down the height of an image in the larger tissue crop, i.e. 70 pixels for a 70 x 70 pixel image crop. The image crop expansion procedure would continue until the sliding window would no longer be able to cover the entirety of a part of the larger tissue crop.

In order to determine how much results could be improved if image crops of larger dimensions were used, image crops of dimensions 128 x 128 pixels were created (see Figure 10). These image crops were partly taken from the same large tissue images as the 70 x 70 image crops. The image cropping procedure was performed in the same way as with the 70 x 70 image crops. However, the large image crops all had widths less than 192 pixels, and heights of 128 pixels or more. The sliding window approach used for the 70 x 70 image crops was reused with the 128 x 128 image crops, with the sliding window movement changed to 64. Similarly, when moving down the height of an image in the larger tissue crop, steps of 128 pixels were used. Due to the random selection of image areas to crop and the use of not exactly the same tissue images, the results for the tests on the 128 x 128 and 70 x 70 image crops respectively were not completely comparable.

To summarize, the initial large crops used to subsequently create 70 x 70 pixel image crops were created with widths greater than 70 pixels. For large crops with a width of 105 pixels or more, the sliding window thus created image crops with 50 % overlap with the image crops preceding and following it on the same row respectively. In contrast, the large crops with widths shorter than 105 pixels did not to allow for the sliding window to move half a window. This forced the sliding window to change row after the creation of rotated and/or mirrored image crop versions. As the sliding window moved 70 pixel vertically after a row change, the smaller image crops to be created on the following row did not overlap with those created on the previous row. Thus, the larger image crops created with the second cropping approach resulted in the creation of non-overlapping 70 x 70 pixel image crops. As a whole, the 70 x 70 pixel image crops were a mix of partially overlapping and non-overlapping image crops. The 128 x 128 image crops, on the contrary, were only created using the second cropping approach, which meant that none of the 128 x 128 image crops overlapped.

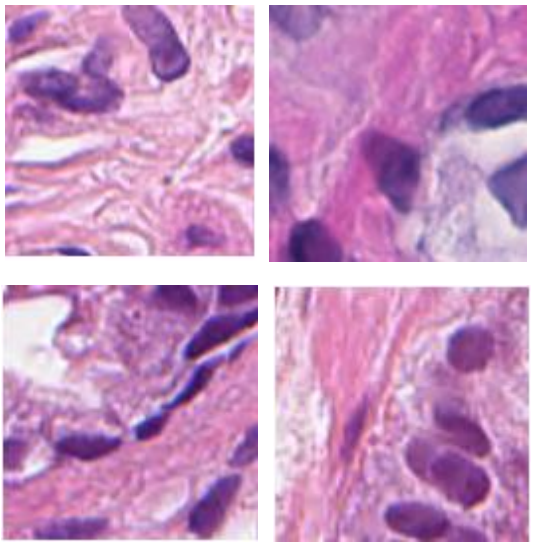


Figure 10. 128 x 128, 40 x magnification image crop examples. (Upper left) benign Gleason image crop example, (Upper right) Gleason 3 image crop example, (Lower left) Gleason 4 image crop example, (Lower right) and Gleason 5 image crop example.

For the 70 x 70 images, an initial pool of 10 560 Gleason 5 image crops, 3 288 Gleason 4 image crops, 3 472 Gleason 3 image crops, and 4 584 benign image crops was created. For the 128 x 128 images, an initial pool of 11 160 Gleason 5 image crops, 4 984 Gleason 4 image crops, 4 104 Gleason 3 image crops, and 3 520 benign image crops was created. The 128 x 128 image pool was later extended to 25 520 Gleason 5 image crops, 22 568 Gleason 4 image crops, 12 312 Gleason 3 image crops, and 26 320 benign image crops. The initial and extended 128 x 128 image pools differed with respect to how benign image crops were created. For the initial pool, benign image crops were created with the intent that they should always contain whole or parts of cells. The extended pool was, in contrast, created with the intent that image crops should be varied, with some image crops containing cells or parts of cells, and others only the surroundings of cells, n.b. not the white background. The reason for this difference in construction of benign image crops was to increase the architectures' capability to interpret the surroundings of cells as free of cancer. In contrast, the initial 128 x 128 pool, thus, gives a somewhat distorted view of benign tissue images. The extended 128 x 128 image pool was never used in training or tests due to time limitations.

Small 70 x 70 and 128 x 128 image crops data sets used for initial architecture testing purposes were constructed by randomly sampling from the respective initial image pools. For each such data set, one folder for Gleason 5 and one folder for non-Gleason 5 images were created, with equally many image crops in each folder. Furthermore, the content of each non-Gleason 5 image folder consisted of Gleason 4, Gleason 3, and benign image crops in equal proportion. Larger data sets were created by partly including image crops sampled for smaller data sets. Through successive data set construction procedures, the data sets 9840_Random_70_Test, and 9840_Random_128_Test were finally created.

4.2.2 Corrected data sets

At the end of the thesis process, an error in the data set generation was discovered. As the data sets 9840_Random_70_Test, and 9840_Random_128_Test included rotations and mirrored images, the final accuracy results might unintentionally have been inflated.

During training and testing, a 5-fold cross-validation approach has been used. During each fold, four fifths of the image crops were used for training the constructed architectures, and the remaining fifth was used for testing. When training and testing on image crops using 9840_Random_70_Test, and 9840_Random_128_Test, the training and test batches might, thus, have, to some degree, contained rotations and/or mirrored images for the same images. Thus, the architectures might have classified test images correctly based on memory of how training-variations of these images were classified.

To achieve correct results as well as determine the effects of the erroneous data set generation, the data sets Fixed_9840_Random_70_Test, and Fixed_9840_Random_128_Test were constructed. These data sets were created by first creating five folders for Gleason 5 images, and five folders for non-Gleason 5 images (both numbered from 0 to 4). For each Gleason 5 folder, 246 Gleason 5 image crops were randomly selected from the initial 70 x 70 and 128 x 128 image pools respectively. Likewise, 82 Gleason 4, Gleason 3, and Benign image crops respectively were randomly selected to each non-Gleason 5 folder. The image crops in these two data set were unique in the respect that no rotated nor mirrored versions of any individual image crops were included among the image crops.

When the image folders had been created, the image crops in each folder were then rotated and mirrored, producing the same variations of an image as in 9840_Random_70_Test, and 9840_Random_128_Test. Thus, each Gleason 5 and non-Gleason 5 image folder contained 1968 image crops. Due to the random selection of constituent image crops, and the selection of only image crops which had not been rotated or mirrored, 9840_Random_70_Test and 9840_Random_128_Test did not contain the exact same image crops as Fixed_9840_Random_70_Test and Fixed_9840_Random_128_Test respectively. Thus, the different rotated and mirrored versions of the same image crop were placed in the same folders. However, not all small image crops created from large image crops taken from the same tissue samples were allocated to the same folder. Also, not all small image crops created from the same large image crop were placed together in the same folder.

At the very end of the thesis process, the discovery that overlap in some of the 70 x 70 pixel image crops might have affected the accuracy results of the constructed architectures was made. As not all 70 x 70 pixel image crops from the same large image crop were placed in the same folder, some image crops in training and test data had similar cellular patterns. The consequence of this was that the results for the architectures using 70 x 70 pixel image crops were inflated. Due to the late stage at which this error was realized, correction of the data sets was not possible. The 128 x 128 pixel image crops did, however, not overlap. Thus, image crop overlap caused no distortions for the results for the architectures using 128 x 128 pixel image crops. However, image crops created from different but neighboring large image crops might have borne some similarities to each other. By allocating such spatially proximate

images to the same folders, the accuracy results might have been inflated even for the 128 x 128 pixel image crops.

4.3 Construction of architectures

To find promising architectures, small data sets were utilized for speeding up initial tests. As interesting architectures were discovered, training-test rounds with 9840_Random_70_Test and 9840_Random_128_Test were performed. The architectures and hyper parameters constructed through tests with the original data sets were then reused for the corrected data sets.

When constructing the architectures, optimization of hyper parameters was also a key concern. The Inception-v4, Inception-ResNet, and ResNet architectures, on which the constructed architectures were modelled, take as input images of dimensions different from the ones used in this thesis, and also utilize specific learning rates and filter counts. In accordance with the goal of this thesis, to study how small-dimensional medical images can be automatically classified, the heuristic approach of successively performing short architecture tests, with changes in only a single hyper parameter at a time, was chosen. Hyper parameters, such as suitable epoch count, batch size, learning rate, and filter counts for each layer, were, thus, experimentally and approximately decided by comparing the results of training-test runs to each other. Furthermore, this approach was also used to determine suitable number of layers for the architectures constructed in this thesis.

4.3.1 Architectures for the original data sets

The architectures constructed for the 9840_Random_70_Test, and 9840_Random_128_Test sets were structured in the following manner: Images in a data set were randomized and a label set, denoting whether an image belonged to the Gleason 5-class or nor, was created. Preceding the architecture build, image crops, normalized between 0-255 in the red, green, and blue color channels, were randomly assigned to a training or test fold in a 5-fold cross-validation manner in a for-loop using scikit-learn. For each iteration, 15 744 and 3 936 image crops were, thus, assigned as training and test images respectively. For the Gleason 5- and non-Gleason 5 image crops respectively, images were then copied to a training or test folder depending on the current fold selections. For every k-fold run, two training and two test folders were, thus, created and filled with image crops. After building the architecture, Keras's flow-from-directory-function, using shuffling of images, was used to perform batch-wise training and testing on images in each of the four folders. Finally, statistics regarding sensitivity and specificity were captured for both training and test images.

4.3.2 Architectures for corrected data sets

As mentioned in the Data sets-sub-heading, the data sets 9840_Random_70_Test, and 9840_Random_128_Test were constructed in a faulty manner. In order to compare the erroneously constructed data sets and the correctly constructed data set, a different approach was used in architecture construction for the corrected data sets. These new architectures were used for final accuracy measurements. The best hyper parameter settings and layer filter counts for the faulty data set architectures were reused for the new architectures.

For Fixed_9840_Random_70_Test, and Fixed_9840_Random_128_Test respectively, the basis of the updated architecture construction was the five Gleason 5 and five

non-Gleason 5 image folders. The contents of each such folder was fixed during all training-test rounds. As with the original data sets, each iteration contained 15 744 training images and 3 936 image test images.

Instead of using scikit-learn to perform 5-fold cross-validation, a regular for-loop and a set-counter incremented for each loop, starting at 0, was used. For each loop, images in the fixed Gleason 5 and non-Gleason 5 image folders were copied to a training or test folder. If the number of a folder coincided with the value of the set-counter, image crops would be moved to a test folder. If the number of a folder differed from the value of the set-counter, the images therein would be copied to a training folder. Depending on the name of each folder, image crops would be copied to a folder for Gleason 5 or non-Gleason 5 images. As with the architectures for the original data sets, two training and two test folders were, thus, created and filled with image crops, once again normalized between 0-255 in the red, green, and blue color channels, for each k-fold run. Furthermore, Keras's flow-from-directory-function and statistics capturing were also used in the same way for both architectures, with the exception that the architectures for the corrected data sets also stored the classification of every image crop.

Due to the different approaches used in creating the original and the corrected data sets, architectures constructed for use with the corrected data sets were trained and tested on more dissimilar image crops than architectures constructed for use with the old data set.

4.4 Architecture overview

The final constructed architectures were based on two main architectures, a CNN-architecture and a ResNet-architecture (See Appendices 3-5) [54, 59]. The CNN-architecture, using two parallel tracks, was tested by itself and as a stem with two different end-modules, adapted from other networks. The first of these end-modules was inspired by the Inception-C-module of the Inception-v4 network (See Appendices 6-7) [51, 59]. The second end-module used was inspired by the Inception-ResNet-C-module of the Inception-ResNet-v1-network (See Appendices 8-9) [51, 59]. The reason for only adding Inception-v4 and Inception-ResNet-v1 end-modules to a base CNN-architecture was that the original Inception-v4 and Inception-ResNet-v1 architectures were too large to reproduce on the available development environments.

The number of filters in each layer of both the ResNet-architecture, and the end-modules used in conjunction with a CNN, were modified in order to work together with either the 70 x 70 or the 128 x 128 image crop data sets. All hyper parameters, except the number of epochs and learning rate, which varied between architecture types, were the same for the final architectures (see Table 1).

The statistical calculations were performed with respect to the Gleason 5 image crops. Thus, the sensitivity measurement was used to determine the percentage of Gleason 5 image crops which was correctly classified. Likewise, the specificity measurement was used to determine the percentage of non-Gleason 5 images which was correctly classified.

Table 1. Hyper parameter settings for architectures with corrected data sets.

Batch size	72
Steps per epoch	218
Optimizer	Adam
ε	0
Decay rates β_1	0.9 (Default Keras value)
Decay rates β_2	0.999 (Default Keras value)
Learning rates for CNN-based architectures	0.0000875
Epoch counts for 70 x 70 CNN-based architectures	6 000
Epoch count for 70 x 70 single-tracked CNN-architecture	1 000
Epoch counts for 128 x 128 CNN-based architectures	2 000
Learning rate for ResNet-architectures	0.000034
Epoch counts for ResNet-architectures	800

5 Results

5.1 Results for the CNN-architectures

5.1.1 Results for the CNN-architecture for the 70 x 70 data set

The 6 000 epoch training-test round for the 70 x 70 data set CNN-architecture took approximately 105 hours to complete all 5 cross-validation folds. With respect to sensitivity, training and validation results surpassed 90.4 % and 87.2 % respectively for all iterations. The validation sensitivity for fold 4 was higher than those for the other four folds, approximately 91.7 % versus sub-90 % for the other four folds. As for specificity, training results were above 91.6 % (lowest for fold 3) for all folds, with folds 1, 4, and 5 having specificities above 95.5 %. Likewise, the validation specificity for fold 3 was lower than the specificities for the other four folds (approximately 83.4 % versus above 89.4 % for the other four folds).

The training and validation accuracies for the 70 x 70 CNN-architecture reached above 91.1 % and 85.3 % respectively for all folds, with a mean validation accuracy of approximately 89 % (see Appendix 10 for accuracy plots). The mean number of correct Gleason 5 image crops was approximately 1 743 out of 1 968 test images, with approximately 225 incorrect classifications. As for the non-Gleason 5 image crops, the mean number of correctly classified images was approximately 1 761, with approximately 207 misclassifications. For the 70 x 70 CNN-architecture, approximately 9 % more Gleason 5 images than non-Gleason 5 images were, thus, misclassified.

5.1.2 Results for the CNN-architecture for the 128 x 128 data set

2 000 epochs were used to train the 128 x 128 CNN-architecture. In total, the training-test rounds took approximately 69 hours to complete. For the 128 x 128 CNN-architecture, sensitivity training- and validation results exceeded 94 % and 87.7 % respectively for all folds. The highest validation sensitivity (fold 2) was 95%. As for specificity, training and validation results surpassed 90.4 % and 83.3 % respectively. The highest validation specificity (fold 5) was 93.1 %.

The training and validation accuracies for the 128 x 128 CNN-architecture exceeded 93.9 % and 89.1 % respectively for all folds. The mean validation accuracy was approximately 91.4 % (see Appendix 11 for accuracy plots). Compared with the 70 x 70 CNN-architecture, the 128 x 128 CNN-architecture had an approximately 2.4 percentage points higher mean validation accuracy. The mean number of correct Gleason 5 image crops were approximately 1 816 out of 1 968 test images. Thus, approximately 152 Gleason 5 image crops were misclassified. As for the non-Gleason 5 image crops, approximately 1 780 images were correctly classified, with 188 non-Gleason 5 image crops misclassified. Thus, approximately 24 % more non-Gleason 5 images than non-Gleason 5 images were misclassified.

5.2 Results for the CNN-architectures with Inception-v4-modules

5.2.1 Results for the CNN-architecture with an Inception-v4-module for the 70 x 70 data set

The 6 000 epoch training-test round for the 70 x 70 Inception-v4 modified CNN-architecture took approximately 109 hours to complete all 5 cross-validation folds. With respect to sensitivity, training and validation results for the five folds ranged from above 89.9 % to above 95.7 %, and from approximately 84.5 % to approximately 93.1 % respectively. The validation sensitivity for fold 4 was approximately three percentage points higher than the second-highest validation sensitivity (93.1 % vs. 90.1%). As for specificity, training results were above 93 % for all folds, with the highest training specificity being approximately 96.9 %. Validation specificity ranged from approximately 89.7 % to approximately 92.8 %.

The training and validation accuracies for the architecture test reached above 92.6 % and 87.7 % respectively for all folds. The mean validation accuracy for all five folds was approximately 89.9 % (see Appendix 12 for accuracy plots). The mean number of correct Gleason 5 image crops was approximately 1 741 out of 1 968 test images, with approximately 227 incorrect classifications. As for the non-Gleason 5 image crops, approximately 1 796 images were correctly classified, with approximately 172 misclassifications. Thus, the architecture misclassified approximately 32 % more Gleason 5 than non-Gleason 5 images.

5.2.2 Results for the CNN-architecture with an Inception-v4-module for the 128 x 128 data set

2 000 epochs were used to train the 128 x 128 Inception-v4 modified CNN. The training-test rounds took approximately 90 hours complete. With regard to sensitivity, training and validation results exceeded 93.6 % and 88.8 % respectively. The highest validation sensitivity was approximately 96.8 % (fold 2). With regard to specificity, training and validation results were above 87.5 % and 81.5 % respectively. The highest validation specificity was approximately 92.9 % (fold 1).

As for the training and validation accuracy results, these exceeded 92.7 % and 89.1 % respectively. The mean validation accuracy was approximately 92 % (see Appendix 13 for accuracy plots). The mean number of correctly classified Gleason 5 image crops was 1 843 out of 1 968 test images, with approximately 125 misclassifications. As for the non-Gleason 5 images, approximately 1 777 images were correctly classified. Thus, approximately 191 non-Gleason 5 images were incorrectly classified. Comparing the misclassification rates for the 128 x 128 Inception-v4 modified CNN-architecture, approximately 53 % more non-Gleason 5 than Gleason 5 images were incorrectly classified.

5.3 Results for the ResNet-architectures

5.3.1 Results for the ResNet-architecture for the 70 x 70 data set

The 800 epoch training-test round for the 70 x 70 data set ResNet-architecture took approximately 33 hours to complete all 5 cross-validation folds. With respect to sensitivity, training and validation results surpassed 99.8 % and 89.5 % for all iterations. The validation sensitivity for fold 2 was lower than those for the other four folds, approximately 89.5 % versus more than 91.8 % for the other four iterations. As for specificity, training results were slightly worse than for training sensitivity, but still above 99.8 % for all iterations. Validation specificity results were lower than the corresponding sensitivity results, exceeding 85.4 % for all folds, but with only two folds exceeding 90 % (95.4% for fold 2 and 93.6 % for fold 5).

The training and validation accuracies for the architecture test reached above 99.8 % and 90.7 % respectively for all folds, with the mean cross-validation accuracy being approximately 91.9 % (see Appendix 14 for accuracy plots). The mean number of correct Gleason 5 image crops were approximately 1 835 out of 1 968 test images, with approximately 133 incorrect classifications. As for the non-Gleason 5 image crops, approximately 1 782 images were correctly classified, with approximately 186 misclassifications. The architecture appeared to have had a harder time correctly classifying non-Gleason 5 images, with almost 40 % more misclassifications for non-Gleason 5 images than for Gleason 5 images.

5.3.2 Results for the ResNet-architecture for the 128 x 128 data set

The 800 epoch training-test round for the 128 x 128 data set ResNet-architecture took approximately 77 hours to complete all 5 cross-validation folds. Training sensitivity reached 100 % for all folds. With regard to validation sensitivity, the results for all folds exceeded 95.2 %, with the results for folds 1-3 exceeding 96.5 %. Validation sensitivity was the highest for fold 5 with sensitivity exceeding 97.5 %. With respect to specificity, the training results again reached 100 % for all folds. Validation specificity results were on the same level as the validation sensitivity results. Validation specificity for folds 2 and 4 were in the 95 %-range, (95 % and 95.8 % respectively). Folds 3 and 5 achieved slightly higher results (96.5 % and 96.6 % respectively), and fold 1 had the highest validation specificity result (98 %).

The training accuracies reached 100 % for all iterations. As for the validation accuracies, the results reached above 95.9 % for all iterations, with the mean cross-validation accuracy being approximately 96.5 % (see Appendix 15 for accuracy plots). This was approximately 4.6 percentage points higher than the mean validation accuracy for the 70 x 70 ResNet-architecture. The mean number of correct Gleason 5 image crops were approximately 1 901 out of 1 968 test images, with approximately 67 misclassifications. The architecture was almost equally good at classifying non-Gleason 5 image crops, with a mean number of approximately 1 897 correct classifications, and 71 misclassifications.

5.4 Results for the CNN-architectures with Inception-ResNet-v1-modules

5.4.1 Results for the CNN-architecture with an Inception-ResNet-v1-module for the 70 x 70 data set

The epoch count for the 70 x 70 Inception-ResNet-v1 modified CNN was 6 000. The total time it took to perform the 6 000 epoch training-test rounds for all 5 cross-validation folds of the 70 x 70 Inception-ResNet-v1 modified CNN-architecture was approximately 104 hours.

With respect to sensitivity, training and validation results for the five folds ranged from above 97.9 % to 100 %, and from approximately 80.5 % to approximately 87.6 % respectively. The validation sensitivity for fold 3 was approximately three percentage points higher than the second-highest validation sensitivity (87.6 % vs. 84.6%). With regard to specificity, training results were above 99.9 % for all folds. Validation specificity ranged from approximately 82.7 % to approximately 87.3 %.

The training and validation accuracies reached above 98.9 % and 81.6 % respectively for all iterations. The highest validation accuracy was approximately 87 % for fold 3. The mean validation accuracy for all five folds was approximately 84.2 % (see Appendix 16 for accuracy plots). The mean number of correct Gleason 5 image crops were approximately 1 637 out of 1 968 test images, with approximately 331 incorrect classifications. As for the non-Gleason 5 image crops, approximately 1 676 images were correctly classified, with approximately 292 misclassifications. Thus, the architecture misclassified approximately 13 % more Gleason 5 than non-Gleason 5 images.

5.4.2 Results for the CNN-architecture with an Inception-ResNet-v1-module for the 128 x 128 data set

The epoch count for the 128 x 128 Inception-ResNet-v1 modified CNN was 2 000. The total time for performing all 5 cross-validation folds was approximately 69 hours. Training sensitivity exceeded 95.4 % for all folds. For the validation sensitivities, the results for all folds were above 93 %, with the highest validation sensitivity reaching approximately 95.1 % (fold 2). As for training and validation specificities, the results exceed 94.5 % and 90.6 % for all folds. The highest validation specificity was achieved in fold 4 (95.1 %).

The training and validation accuracies reached above 94.9 % and 92.7 % respectively for all folds. The highest validation accuracy was achieved in fold 3 (94.1 %). The mean cross-validation accuracy for all five folds was approximately 93.3 % (see Appendix 17 for accuracy plots and Appendix 18 for a comparison of the architecture results for the original and corrected data sets). The mean number of correct Gleason 5 image crops were approximately 1 849 out of 1 968 test images, with approximately 119 images incorrectly classified. As for the non-Gleason 5 image crops, approximately 1 822 images were correctly classified, with approximately 146 images incorrectly classified. Thus, the architecture misclassified approximately 23 % more non-Gleason 5 than Gleason 5 images.

5.5 Misclassification examples

Several architectures have been constructed in this thesis. For the purpose of being concise, and to exemplify misclassifications with an accurate architecture, only the ResNet-architecture will be used to exemplify image misclassifications. Misclassified image crops examples of both size 70 x 70 and 128 x 128 have been covered in order to study misclassification differences related to differences in image crop dimensions.

5.5.1 Misclassifications for the 70 x 70 data set

To determine which image crops the 70 x 70 ResNet-architecture had the most difficulties in classifying correctly, the classification results for fold 5 were inspected. This fold had the highest validation accuracy, approximately 93.5 %, with the lowest difference between sensitivity and specificity (approximately 0.4 percentage points).

The breakdown in misclassified non-Gleason 5 image crops for fold 5 was as follows: 54 benign, 19 Gleason 3, and 52 Gleason 4. From this we can see that the Gleason 3 images were much easier to classify than the benign or Gleason 4 images. For many of the misclassified non-Gleason 5 images, several rotational or mirrored variations of the same image crop were all misclassified. However, there were also cases where only a single rotational or mirrored variant of the same image crop was incorrectly classified. In many cases, misclassified non-Gleason 5 images did not appear to differ from correctly classified non-Gleason 5 images. Some similarities between misclassified non-Gleason 5 images were, however, noted. Incorrectly classified non-Gleason 5 image crops commonly contained cells with low contrast against the surrounding tissue, or several clustered cells (see Figure 11).

With regard to the Gleason 5 images, 132 of these images were incorrectly classified. As for the misclassified non-Gleason 5 images, it was common that many rotational or mirrored variations of the same image crop were incorrectly classified. Although, there were instances of only one variation of the same image crop being incorrectly classified. Misclassified Gleason 5 images were in many cases hard to differentiate from correctly classified Gleason 5 images. However, it was observed that incorrectly classified Gleason 5 images often depicted cells very close to each other, appearing as a few large cells, or depicted a few small cells (see Figure 11).

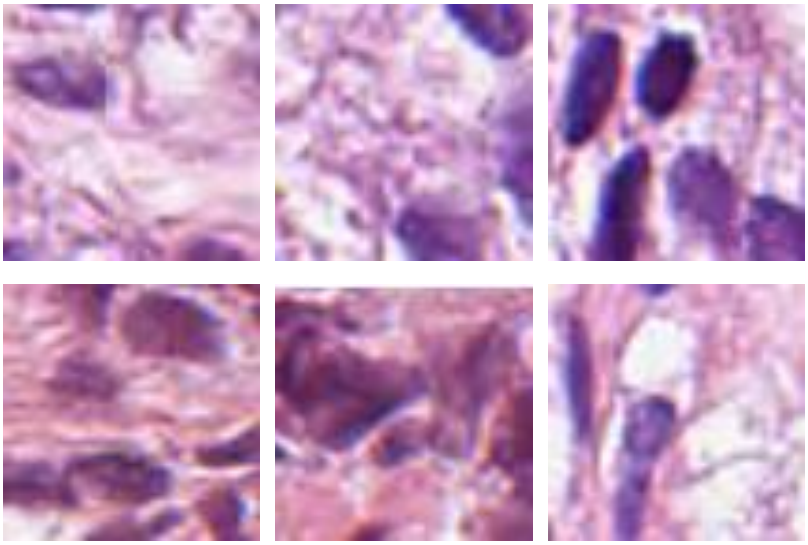


Figure 11. Misclassified image crops in the 70 x 70 data set. (Upper left) Gleason 3 image crop depicting cells with low contrast against tissue background, (Upper middle) Gleason 4 image crop depicting cells with low contrast against tissue background, (Upper right) benign image crop depicting several clustered cells, (Lower left) Gleason 3 image crop depicting several clustered cells, (Lower middle) Gleason 5 image crop depicting cells very close to each other, (Lower right) and Gleason 5 image crop depicting a few small cells.

5.5.2 Misclassifications for the 128 x 128 data set

To determine which image crops the 128 x 128 ResNet-architecture had the most difficulties in classifying correctly, the classification results for fold 5 were inspected. This fold had the second highest validation accuracy, approximately 97.1 %, with a difference of approximately 1 percentage point between sensitivity and specificity. The sensitivity-specificity difference for fold 5 was approximately 0.4 percentage points lower than the sensitivity-specificity difference for the fold with the highest validation accuracy.

The breakdown in misclassified non-Gleason 5 image crops for fold 5 was as follows: 22 benign, 27 Gleason 3, and 18 Gleason 4. The non-Gleason 5 images were all approximately equally simple for the architecture to classify. If we compare this breakdown with that of the 70 x 70 ResNet-architecture, we can see that the 128 x 128 architecture appeared to be better at classifying non-Gleason 5 images as a whole. The 128 x 128 ResNet-architecture made approximately half as many non-Gleason 5 misclassifications as the 70 x 70 architecture (67 vs. 125), and had a more even distribution of misclassifications between the three types of non-Gleason 5 image crops.

For some image crops, only a single rotational or mirrored variant of a certain non-Gleason 5 image crop was misclassified. However, for most of the misclassified non-Gleason 5 images several rotational or mirrored variations of the same image crop were all misclassified. As with the 70 x 70 architecture, misclassified non-Gleason 5 images did in many cases not appear different from correctly classified non-Gleason 5 images. Some similarities between misclassified non-Gleason 5 images were noted. Incorrectly classified non-Gleason 5 image crops commonly contained several clustered cells or a small number of

cells located close to the borders of the image crops (see Figure 12). The misclassified benign image crops mostly consisted of variations of image crops with cells close to the borders of the image crops. Some misclassified Gleason 3 image crops also depicted cells close to the borders of the image crops. All misclassified Gleason 4 images, and the remainder of the benign and Gleason 3 image crops shared the similarity of depicting several clustered cells. The reason why the 128 x 128 ResNet-architecture had trouble classifying these image crops might have been that the patterns of lone cells or several chaotically spread-out cells are more common for malign PCa than benign PCa.

With regard to the Gleason 5 images for the 128 x 128 ResNet-architecture, 48 of these were incorrectly classified. Compared with the Gleason 5 misclassifications for the 70 x 70 architecture, this was almost equal to three times fewer misclassifications (48 vs. 132). For incorrect classifications, it was common that rotational or mirrored variations of the same image crop were also incorrectly classified. However, for some image crops only one variant was misclassified. Once again, misclassified Gleason 5 were in many cases hard to differentiate from correctly classified Gleason 5 images. Two similarities between misclassified Gleason 5 images were observed: first, that incorrectly classified Gleason 5 images often contained cells with low contrast, and, second, depicted cells very close to each other, appearing as a few large cells (see Figure 12).

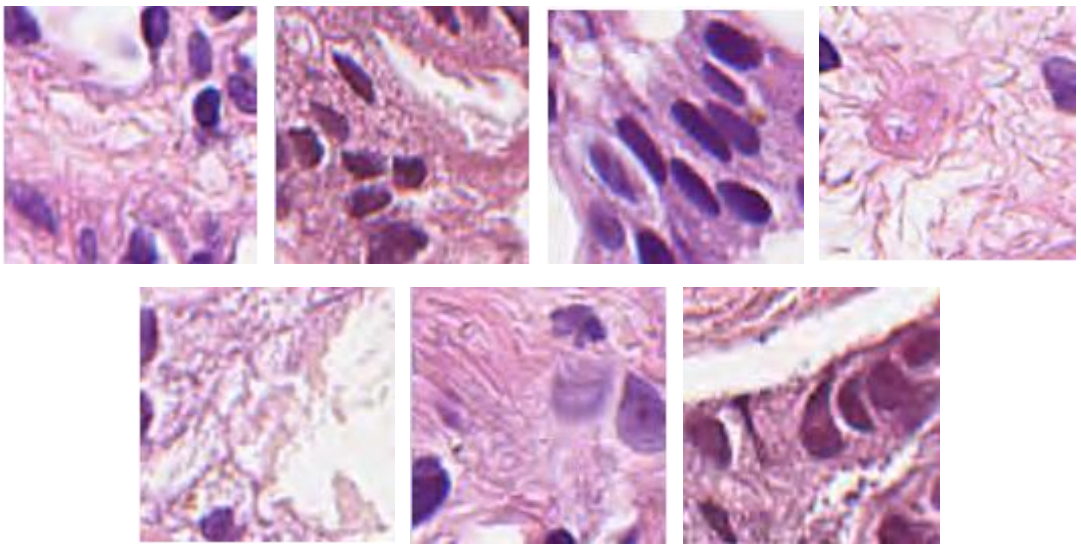


Figure 12. Misclassified image crops in the 128 x 128 data set. (Upper left) benign image crop depicting several clustered cells, (Upper middle-left) Gleason 3 image crop depicting several clustered cells, (Upper middle-right) Gleason 4 image crop depicting several clustered cells, (Upper right) benign image crop depicting cells close to the borders of the image, (Lower left) Gleason 3 image crop depicting cells close to the borders of the image, (Lower middle) Gleason 5 image crop depicting cells with low contrast, (Lower right) and Gleason 5 image crop depicting cells very close to each other.

6 Discussion

6.1 Analysis

6.1.1 Data set analysis

In this thesis image crops have been created by, first, manually creating larger image crops in MS Paint and, second, automatically expanding these image crops using a sliding-window script. As the image crops were of small dimensions (70 x 70, and 128 x 128 respectively, both with 40 x magnification), not creating images crops depicting only the tissue surrounding cells nor image crops depicting only the background, became a priority matter. If the studies performed in this thesis are to be recreated with new image crops, the image cropping procedure could become a problem.

Using a program which automatically subdivides a large tissue image into smaller image crops, image crops depicting only tissue surrounding cells might be created for Gleason 3-5 tissue samples. If these images are to be used, the architecture would have a harder time differentiating between benign or malign tissue. However, creating image crops with automatic scripts could save time and man power spent on manually creating initial large image crops. If the intent of cropping images is to automatically generate image crops of small dimensions, background color conditions should be included in the cropping algorithm. For instance, making it a requirement that the background color is not white for more than 50 % of an image crop would stop the cropping program from creating image crops containing only the background. Similarly, conditions on color, and color changes could be used in order to hinder an automatic cropping program from creating image crops containing only tissue surrounding cells.

The problem of automatic image cropping is, however, intrinsically connected to the dimensions (e.g. crop size and magnification) of the image crops and which tissue samples images are cropped from. If the dimensions of image crops are large (large crop size and small magnification), the classification objects would shift from the cellular level to the multicellular pattern-level. As this means a shift in which morphological patterns that are interesting to classify, the need for manual inspection, to avoid cropping only surrounding tissue, would disappear. As Gleason scoring is based on the histopathological appearance of samples, this would enable easier medical classification of image crops.

If image crops with small dimensions are to be classified, the individual image crops give a limited view of how cells are clustered together and how far apart such clusters are, with the boon being that the shapes of individual cells is clearer than for image crops with lower magnification. Conversely, if image crops with large dimensions are to be classified, cellular patterns, including how much surrounding tissue there is between cells, and if cells lie close together, would be much easier to detect. Likewise, the concentration of cells vs. surrounding tissue in tissue samples affects the need to manually check if image crops actually contain cells.

Problems with the dimensions of the image crops were also suggested by the misclassification analyses. Looking at the misclassification results for the 70 x 70 and 128 x 128 ResNet-architectures, we can see some common themes. The contrast of cells against tissue background, if there existed clusters of cells in an image, and if the cells were close enough to

each other to appear as a few large cells were similarities noted for both architectures. The problem of contrast signifies that differences in tissue staining practices might pose a problem if data sets consisting of images from multiple sources are used in medical image analysis. Even though color and contrast changes have not been used in this thesis, the misclassification analyses suggest that they might be of use in improving the accuracy of neural networks.

That the number of cells and the distances between cells appeared to affect the classification capability of the ResNet-architectures suggested that the image crop sizes 70 x 70 and 128 x 128 were too small. Creating images crops with larger sizes would lead to the possibility of more cells being captured in each image crop. Thus, a few differences in the overall cellular pattern of an image crop, such as clusters of cells, would not affect the overall classification of an image crop.

Another problem with creating image crops of small dimensions is whether the image crops are labelled with the correct Gleason grade or not. In a large tissue sample, some parts of the tissue sample might have a different degree of malignancy than the overall classification of the tissue sample. If an architecture is trained on small image crops it might therefore learn to associate the wrong Gleason grade with a certain pattern. To solve this problem, professionals trained in classifying PCa could be involved in the creation of smaller image crops, either as responsible for creating image crops or in a verifying capacity. Another way to attenuate the risk of training an architecture on image crops with incorrect labels is to use image crops of a large size relative to the size of the tissue image. If a tissue image as a whole can be classified with a certain Gleason grade, a large image crop might depict several cellular patterns, where the majority of cellular patterns might have the same Gleason grade as the overall tissue score. Thus, malignancy variations across a certain tissue sample would not affect the correctness of annotations of image crops from said tissue sample.

A problem related to spatial malignancy differences is overlap between image crops and similarity due to spatial proximity in general. As a portion of the 70 x 70 pixel image crops had a 50 % overlap with neighboring image crops from the same row on the larger crops, the architectures might have learned patterns in the training data partially existing in test data. Thus, similarity due to spatial proximity between overlapping images might have led to inflated accuracy results for the 70 x 70 pixel image crops. Image crops taken from regions in close proximity to each other might also have been similar to each other both with respect to cellular patterns and staining. This also means that small image crops created from a large image crop might have been similar not only to other image crops created from the same large image crop, but also to spatially proximate image crops created from other large image crops.

As stated in the Methodology section, all rotated and mirrored versions of the same small image crops were placed in the same folder. However, all small image crops created from large image crops belonging to the same tissue sample were not allocated to the same folder. Furthermore, not all small image crops created from the same large image crop were placed in the same folder. Thus, the architectures might have been able to discern patterns for small image crops from spatially proximate large image crops or the same large image crop residing in the training folders. Even though the 128 x 128 pixel image crops did not overlap, training and test data might thus still have had similarities due to how image crops were allocated to the different folders.

In the case that multiple large tissue samples are available, small image crops created from tissue samples can be allocated to the same folders. If the available tissue samples are of different sizes, and the majority of small image crops are created from a small set of tissue samples, image crops generated from large tissue samples might need to be allocated to different folders. For such situations, small image crops spatially far apart from each other, i.e. created from large image crops not proximate to each other, could be placed in different folders. This could make the image crops in different folders more dissimilar, and not inflate the accuracy results due to spatio-cellular similarities between image crops in different folders.

The data sets constructed in this thesis were also quite small, with 19 680 image crops created for each data set, where all of the images were taken from one source (SUS). The problem areas raised earlier under the "Data set analysis" heading might also have been exacerbated by the data set sizes. If a neural network is trained on a large amount of image crops from various sources, the system is exposed to a variety of different histopathological patterns. Certain cellular patterns common to the tissue images used in this thesis might have been easier for the constructed architectures to learn, thereby, making the constructed networks appear to be better at classifying PCa than if trained on a larger data set.

6.1.2 Architecture analysis

Analyzing the architectures constructed in this thesis, we can see that drop-out was not needed in order for the architectures to produce good results. The reason why drop-out was not needed might have been the level of hyper parameter optimization used in this thesis. As only images from SUS were used, and as the data sets in this thesis were quite small, heavy use of hyper parameter optimization might have made normalization aids, such as drop-out, unnecessary. However, if architectures are trained on data sets with larger disparities between images, e.g. arising from inter-hospital tissue sampling or staining differences, drop-out, or other normalization methods, might be needed in order for the architectures to be able to perform Gleason scoring with high accuracy.

From the architecture designs, we can see that the 70 x 70 and 128 x 128 base CNN- and Inception-v4-modified CNN-architectures had no differences with regard to their respective filter counts per layer. As the 70 x 70 Inception-ResNet-v1-modified CNN-architecture suffered from design flaws, filter count comparisons with its 128 x 128 counterpart could not be done in a fair manner. In contrast, the ResNet-architectures differed with respect to filter counts, with the 70 x 70 version having slightly more filters per layer than the 128 x 128 version. Thus, it did not appear as though scaling up an architecture for larger images would always require scaling of the filter counts per layer. Rather, the matter of filter scaling appeared to depend on the specific architecture utilized. For instance, a reason why the ResNet-architectures allowed for the filter counts per layer to be higher for smaller images could have been that more aspects of smaller images could have been inspected without creating a model too complex to successfully classify the image crops.

Looking at the parameters needed to model the different architectures, we can see that the base CNN-architectures required much fewer parameters than the other architectures. Interestingly, the Inception-v4 architectures required the most parameters by a wide margin, with the ResNet-architectures at a distant second-place. In contrast, the Inception-ResNet-v1-architectures required less than 20 times more parameters than the base CNN-architectures.

The need for few parameters to model the base CNN-architecture can be explained by its simple architecture, consisting of only two tracks with a small number of filters on each layer. Even though the Inception-ResNet-v1-architectures had several tracks, the filter counts in each layer were below 60 filters. The Inception-v4 architectures' needs for large models might have been due to the multiple tracks of large filter layers in the Inception-v4 end-modules. Likewise, the reason for the large parameter counts in the ResNet-architectures might have been the use of multiple modules consisting of layers with fairly large filter-stacks.

All the designed architectures, except the 70 x 70 Inception-ResNet-v1-CNN-architecture, achieved validation accuracies close to or above 90 % for the corrected data set. Comparing the validation accuracies for the properly designed architectures, we can see that the modified CNN-architectures achieved higher validation accuracies than the base CNN-architectures. We can also see that the 128 x 128 Inception-ResNet-v1 modified CNN-architecture performed better than the Inception-v4 modified CNN-architecture. However, both the 70 x 70 and the 128 x 128 ResNet-architectures outperformed the other architectures in their respective data set categories. The 70 x 70 ResNet-architecture also performed nearly as good as the 128 x 128 Inception-v4 modified CNN-architecture.

The explanation for the base CNN-architectures' quite high validation accuracies might have been that the double-track design allowed for detection of many patterns and combinations of patterns. The main reason for this might have been the use of ReLU-activations in the convolution layers before the convolution outputs were added together in the addition layers. In turn, the reason why the Inception-v4 modified networks performed better than the base CNN-network might have been the addition of an Inception-module, inspecting images for different interesting patterns. Similarly, the 128 x 128 Inception-ResNet-v1 network might have produced better validation results than the corresponding 128 x 128 Inception-v4 networks due to the inclusion of an Inception-ResNet-v1-module, utilizing both the Inception pattern scanning-technique and the ResNet-technique.

As we can see from the validation accuracy plots, the ResNet-architectures quickly reached high levels of validation accuracy. The reason why the ResNet-architectures produced better validation results than the other networks might have been due to the special combination of ResNet-methodology and filter counts utilized for these architectures. The fast learning speed of the ResNet-architectures signaled that they could speedily identify interesting patterns in the images. However, the ResNet-networks also experienced massive temporary accuracy drops. These networks might, thus, have been more over-optimized than the other networks designed in this thesis. The optimization could have caused fast learning for some image batch combinations, with resultant validation accuracies increases, and for some image batch combinations resulted in validation accuracy drops. As the ResNet-architectures were instable, the most powerful network was the 128 x 128 Inception-ResNet-v1 architecture. Although the properly designed Inception-v4 and Inception-ResNet-v1 architectures gave better mean cross-validation accuracies than the base CNN-architectures, the inter-epoch validation accuracy variations for the modified CNN-architectures exceeded the differences in mean cross-validation accuracies between the CNN-architectures and the modified CNN-architectures. Thus, it could not be concluded that adding the Inception-v4 or Inception-ResNet-v1 modules to the end of CNN-architectures produced better validation accuracy results than the base architectures without any such modifications.

6.2 Conclusion

The decision of which image crop dimensionality to use has ramifications on what a neural network can classify. In this thesis it is not purported that the dimensions selected for the image crops are the best fit for all automatic Gleason-classification situations. Using larger image crops with a different degree of magnification changes what medical information content as well as which cellular patterns that are analyzed. Creating image crops with large sizes might combat problems with malignancy variations in images, allowing for capture of image crops more representative of a tissue image's overall classification. Better validation accuracy results for 128 x 128 architectures and the misclassification analyses indicated that using image crops larger than 70 x 70 pixels might yield better validation accuracies when using a magnification of 40X, supporting the important role image crop dimension selection plays.

As for the effect of the constructed architectures on the validation accuracies, the results were unclear. Even though modifying CNN-architectures with Inception-v4 or Inception-ResNet-v1 end-modules yielded higher mean cross-validation accuracies than the base CNN-architectures by themselves, the differences between the base and the modified networks were dwarfed by the inter-epoch accuracy variations of the modified networks. The fact that the best networks, those of ResNet design, suffered from massive temporary drops in validation accuracy, further muddied the waters.

This thesis has had an exploratory purpose to study PCa-classification for images of small dimensions. For this purpose, architectures were constructed with different designs and trained on image crops of different dimensions. For PCa-classification purposes, the results indicated that using image crops larger than 70 x 70 pixels when using a magnification of 40X might yield better validation accuracies. However, several questions still remain unanswered: Which dimensions are the most suitable for PCa-classification or classification of other medical disorders? Does dimensionality have to be adapted depending on the facility where tissue samples are created? Can tissue samples be classified using parallel architectures adapted for image crops with different dimensions? Which network is best suited for high-accuracy image classifications? These questions are left for future researchers to answer.

6.3 Suggestions for future research

The focal point in this thesis has been on how small dimensions, herein exemplified with 70 x 70 and 128 x 128, 40X magnification images, affect the classification of Gleason scores. This theme can be further delved into by creating smaller image crops from large image crops, the results of which can be compared with each other. Different degrees of magnification could also be taken into account. Accuracy results for different image crop sizes, e.g. 128 x 128 and 256 x 256, could then readily be compared with each other. Such an approach could provide researchers with more insight into how dimensionality and increases/decreases in image information affect classification results. This type of study could be performed with the same architectures as laid out in thesis, thus, using this thesis as a comparative basis, or other suitable architectures. Use of larger data sets when performing such tests could also more clearly show the viability of small region PCa-classification.

In this thesis, transfer learning, using architectures pre-trained on histopathological images, has not been used for classification testing. Repeating the experiments performed in this thesis

but with architectures pre-trained on histopathological images might yield better results than those obtained in this thesis. To account for variations between histopathological images created at different facilities, the used image data set should consist of images from different sources.

Apart from using another data set for transfer learning purposes, other architectures than those studied in this thesis, such as the full Inception-v4-, and Inception-ResNet-v1/v2-architectures, and/or NASNet, could also be used in order to obtain inter-architecture comparison data [51, 60].

The classification studies performed in this thesis have been made through supervised learning with CNNs. A future avenue of small-dimensional histopathological sample classification could be the use of unsupervised learning. Furthermore, CNN-comparison studies between different kinds of carcinoma in varying dimensions could be made in order to determine the lowest dimensional bounds for practical use of automatic classification for the respective kind of carcinoma.

As the architectures used in this thesis were not rotationally invariant, rotation and mirroring of base image crops was used extensively as compensation. To save time spent on creating rotated variations of image crops, capsule networks could be used to study how rotational invariance in machine learning architectures affects classification results [61, 62].

References

[1] Siegel, R. L., Miller, K. D., & Jemal, A. (2017). *Cancer statistics, 2017*. CA: A Cancer Journal for Clinicians 67(1), 7-30 (2017).

[2] Tsang, E. (2017). Hong Kong records latest high of 30,318 new cancer cases, with colorectal cancer most common. *South China Morning Post*. 24 October. http://www.scmp.com (Accessed 2018-06-17).

[3] Nasr, N. (2016). Ofarlig prostatacancer hos en man ökar risken för aggresiv variant hos hans bror. *Sydsvenskan*. 11 July. https://www.sydsvenskan.se (Accessed 2018-06-17).

[4] Epstein, J.I. (2010). An Update of the Gleason Grading System. *The Journal of Urology* 183(2):433-40.

[5] Epstein, J.I., Zelefsky, M.J., Sjoberg, D.D., Nelson, J.B., Egevad, L., Magi-Galluzzi, C., Vickers, A.J., Parwani, A.V., Reuter, V.E., Fine, S.W., Eastham, J.A., Wiklund, P., Han, M., Reddy, C.A., Ciezki, J.P., Nyberg, T. & Klein, E.A. (2016). A Contemporary Prostate Cancer Grading System: A Validated Alternative to the Gleason Score. *European urology* 69(3): 429-435.

[6] Persson, J., Wilderäng, U., Jiborn, T., Wiklund, P.N., Damber, J.-E., Hugosson, J., Steinbeck, G., Haglind, E. & Bjartell, A. (2014). Interobserver variability in the pathological assessment of radical prostatectomy specimens: Findings of the Laparoscopic Prostatectomy Robot Open (LAPPRO) study. *Scandinavian Journal of Urology* 48(2): 160-7.

[7] Vinnova. (2015). *DOGS-Digital Pathology for Optimized Gleason Score in Prostate Cancer*. https://www.vinnova.se/p/dogs--digital-pathology-for-optimized-gleason-score-in-prostate-cancer/ (Accessed 2018-06-17).

[8] Lippolis, G., Edsjö, A., Helczynski, L., Bjartell, A., & Overgaard, N. C. (2013). Automatic registration of multimodal microscopy images for integrative analysis of prostate tissue sections. *BMC Cancer* 13: 408-418.

[9] Isaksson, J., Arvidsson, I., Åström, K. & Heyden, A. (2017). Semantic segmentation of microscopic images of H&E stained prostatic tissue using CNN. *2017 International Joint Conference on Neural Networks (IJCNN), Neural Networks (IJCNN), 2017 International Joint Conference*: 1252-6, May, 2017.

[10] Källén, H., Molin, J., Heyden, A., Lundström, C. & Åström, K. (2016). Towards grading gleason score using generically trained deep convolutional neural networks. *2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI)*: 1163-1167, April, 2016.

[11] Doyle, S., Hwang, M., Shah, K., Madabhushi, A., Feldman, M., & Tomaszeweski, J. (2007). Automated Grading of Prostate Cancer Using Architectural and Textural Image Features. *2007 4Th IEEE International Symposium On Biomedical Imaging: From Nano To Macro*: 1284-7 April, 2007.

[12] Litjens, G., Sánchez, C. I., Timofeeva, N., Hermsen, M., Nagtegaal, I., Kovacs, I., Kovacs, I., Hulsbergen-van de Kaa, C., Bult, P., van Ginneken, B. & van der Laak, J. (2016). Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. *Scientific Reports* 6:26286.

[13] Gummeson, A., Arvidsson, I., Ohlsson, M., Overgaard, N. C., Krzyzanowska, A., Heyden, A., Bjartell, A. & Åström, K. (2017). Automatic Gleason grading of H&E stained microscopic prostate images using deep convolutional neural networks. *Proceedings Of SPIE*: 10140.

[14] Jimenez-del-Toro, O., Atzori, M., Otálora, S., Andersson, M., Eurén, K., Hedlund, M., Rönnquist, P. & Müller, H. (2017). Convolutional neural networks for an automatic classification of prostate tissue slides with high-grade Gleason score. *Proceedings Of SPIE*: 10140.

[15] Flood, G. (2016). *Deep Learning with a Directed Acyclic Graph Structure for Segmentation and Classification of Prostate Cancer*. Lund: Lund University, 2016.

[16] Gummeson, A. (2016). *Prostate Cancer Classification using Convolutional Neural Networks*. Lund: Lund University, 2016.

[17] Ekelund, J. (2017). *Rotational Invariant Neural Networks for Prostate Cancer Classification*. Lund: Lund University, 2017.

[18] Drake, R.L., Vogl, A.W. & Mitchell, A.W.M. (2014). *Gray's Anatomy for students*. International edition, 3rd edition. Philadelphia, PA: Elsevier/Churchill Livingstone.

[19] Eroschenko, V.P. (2017). *Atlas of Histology with Functional Correlations*. International edition, 13th edition. Philadelphia, PA: Wolters Kluwer/Lippincott Williams & Wilkins.

[20] Gartner, L.P. & Hiatt, J.L. (2007). *Color Textbook of Histology*. 3rd edition. Philadelphia, PA: Elsevier Saunders.

[21] Mescher, A.L. (2016). *Junqueira' Basic Histology: Text and Atlas*. 14th edition. New York, NY: McGraw-Hill Education.

[22] Moore, K.L., Dalley, A.F. II & Augur, A.M.R. (2017). *Clinically Oriented Anatomy*. 8th edition. Philadelphia, PA: Wolters Kluwer/Lippincott Williams & Wilkins.

[23] Ross, M.H. & Pawlina, W. (2011). *Histology: a text and atlas: with correlated cell and molecular biology*. 6th edition. Philadelphia, PA: Wolters Kluwer Health/Lippincott Williams & Wilkins.

[24] Moore, K.L., Dalley, A.F. II & Augur, A.M.R. (2014). *Essential clinical anatomy*. 5th edition. Philadelphia, PA: Wolters Kluwer Health/Lippincott Williams & Wilkins.

[25] M., & Scardino, P. T. (2002). Localized prostate cancer. *Current Problems In Surgery*, 39843-957.

[26] Gilroy, A.M. (2016). *Anatomy: An Essential Textbook*. New York, NY: Thieme.

[27] Hall, J.E. (2016). *Guyton and Hall Textbook of Medical Physiology*. 13th edition. Philadelphia, PA: Elsevier.

[28] Johanson, F. (2015). *Anatomi i klartext*. 1st edition. Lund: Studentlitteratur.

[29] Lowe, J.S & Anderson, P.G. (2015). *Steven's & Lowe's Human histology*. 4th edition. Philadelphia, PA: Elsevier/ Mosby.

[30] Strayer, D.S., Rubin, E., Saffitz, J.E. & Schiller, A.L. (2015). *Rubin's pathology: Clinicopathologic foundations of Medicine*. Philadelphia, PA: Wolters Kluwer Health/Lippincott Williams & Wilkins.

[31] Goodfellow, I., Bengio, Y. & Courville, A. (2016). Cambridge, MA: MIT Press. http://www.deeplearningbook.org (Accessed 2018-07-12).

[32] Haykin, S. (2008). *Neural Networks and Learning Machines*. 3rd edition. Pearson/Prentice Hall. https://cours.etsmtl.ca/sys843/REFS/Books/ebook_Haykin09.pdf (Accessed 2018-07-12).

[33] Stanford. (2018). *CS231 – Convolutional Neural Networks for Visual Recognition, Neural Networks 1*. http://cs231n.github.io/neural-networks-1/ (Accessed 2018-07-24).

[34] Stanford. (2018). *CS231 – Convolutional Neural Networks for Visual Recognition, Convolutional Networks*. http://cs231n.github.io/convolutional-networks/ (Accessed 2018-07-24).

[35] Torres, J. (2016). *First Contact with TensorFlow, get started with Deep Learning Programming*. https://jorditorres.org/research-teaching/tensorflow/first-contact-with-tensorflow-book/first-contact-with-tensorflow/ (Accessed 2018-07-12).

[36] LISA lab, University of Montreal. (2015). *Deep Learning Tutorial*. Release 0.1. http://deeplearning.net/tutorial/deeplearning.pdf (Accessed 2018-07-12).

[37] Stanford. (2018). *CS231 – Convolutional Neural Networks for Visual Recognition, Classification*. http://cs231n.github.io/classification/ (Accessed 2018-07-24).

[38] Stanford. (2018). *CS231 – Convolutional Neural Networks for Visual Recognition, Neural Networks 3*. http://cs231n.github.io/neural-networks-3/ (Accessed 2018-07-24).

[39] https://keras.io/layers/merge/ (Accessed 2018-07-05).

[40] https://keras.io/activations/ (Accessed 2018-07-05).

[41] Stanford. (2018). *CS231 – Convolutional Neural Networks for Visual Recognition, Linear Classify*. http://cs231n.github.io/linear-classify/ (Accessed 2018-07-24).

[42] Nielsen, M.A. (2015). Neural Networks and Deep Learning. Determination Press. http://neuralnetworksanddeeplearning.com/index.html (Accessed 2018-07-20).

[43] Stanford. (2018). *CS231 – Convolutional Neural Networks for Visual Recognition, Optimization 1*. http://cs231n.github.io/optimization-1/ (Accessed 2018-07-24).

[44] https://keras.io/optimizers/ (Accessed 2018-07-05).

[45] Kingma, D. & Ba, J. (2017). Adam: A Method for Stochastic Optimization. https://arxiv.org/abs/1412.6980v8 (Accessed 2018-07-21).

[46] Stanford. (2018). *CS231 – Convolutional Neural Networks for Visual Recognition, Optimization 2*. http://cs231n.github.io/optimization-2/ (Accessed 2018-07-24).

[47] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15(1): 1929-1958.

[48] Stanford. (2018). *CS231 – Convolutional Neural Networks for Visual Recognition, Neural Networks 2*. http://cs231n.github.io/neural-networks-2/ (Accessed 2018-07-24).

[49] Ioffe, S. & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://arxiv.org/abs/1502.03167 (Accessed 2018-07-08).

[50] https://keras.io/layers/convolutional (Accessed 2018-07-04).

[51] Szegedy, C., Ioffe, S., Vanhoucke, V. & Alemi, A. (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. https://arxiv.org/abs/1602.07261 (Accessed 2018-07-09).

[52] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. & Rabinovich, A. Going Deeper with Convolutions. https://arxiv.org/abs/1409.4842 (Accessed 2018-07-09).

[53] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision. https://arxiv.org/abs/1512.00567 (Accessed 2018-07-09).

[54] He, K., Zhang, X., Ren, S. & Sun, S. (2015). Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.03385 (Accessed 2018-07-09).

[55] Altman, D.G. & Bland, J.M. (1994). Statistics Notes: Diagnostic tests 1: sensitivity and specificity. *BMJ* 1994; 308:1552.

[56] Chollet, F. et al. (2015). Keras. https://keras.io (Accessed 2018-07-14).

[57] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. Software available from tensorflow.org. https://www.tensorflow.org (Accessed 2018-07-14).

[58] Arvidsson, I., Overgaard, N.C., Marginean, F.-E., Krzyzanowska, A., Bjartell, A., Åström, K. & Heyden, A. Generalization of prostate cancer classification for multiple sites using deep learning. *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, April 4-7, 2018, Washington, D.C., USA.

[59] Tensorflow GitHub. https://github.com/tensorflow/models/tree/master/research/slim/nets (Accessed 2018-07-01).

[60] Zoph, B., Vasudevan, V., Shlens, J. & Le, Q.V. (2016). Learning Transferable Architectures for Scalable Image Recognition. https://arxiv.org/abs/1707.07012 (Accessed 2018-06-18).

[61] Sabour, S., Frosst, N. & Hinton, G.E. Dynamic Routing Between Capsules. *31st Conference on Neural Information Processing Systems (NIPS 2017)*, December 4-9, Long Beach, CA, USA.

[62] Hinton, G.E., Sabour, S. & Frosst, N. Matrix capsules with EM routing. *Sixth International Conference on Learning Representations (ICLR 2018)*, April 30 – May 3, Vancouver, BC, Canada.

Appendix 1. Original Inception-v4 Architecture

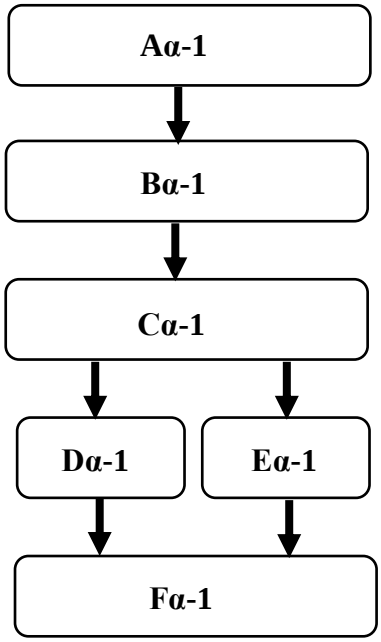


Figure 13. Part 1-1 of the Inception-v4-stem.

Aα-1: Convolutional layer. Filters: 32, kernel size: 3 x 3, stride: 2 x 2, padding: valid.

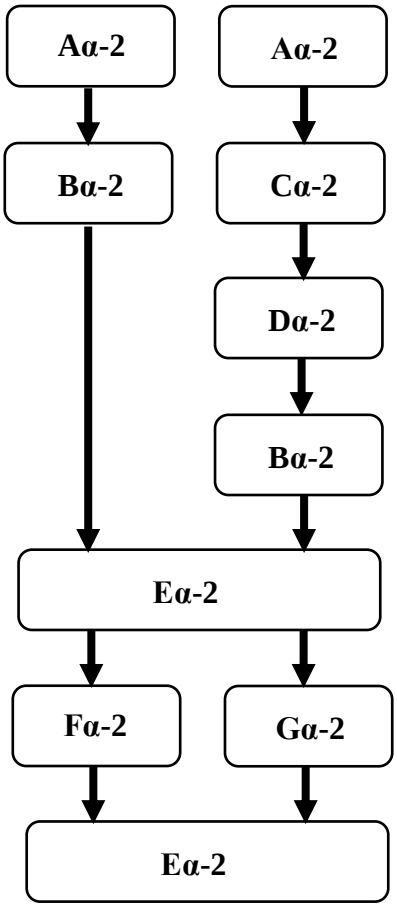
Bα-1: Convolutional layer. Filters: 32, kernel size: 3 x 3, padding: valid.

Cα-1: Convolutional layer. Filters: 64, kernel size: 3 x 3, padding: same.

Dα-1: Max pooling layer. Kernel size: 3 x 3, stride: 2 x 2, padding: valid.

Eα-1: Convolutional layer. Filters: 96, kernel size: 3 x 3, stride: 2 x 2, padding: valid.

Fα-1: Filter concatenation layer.



Aα-2: Convolutional layer. Filters: 64, kernel size: 1 x 1, padding: same.

Bα-2: Convolutional layer. Filters: 96, kernel size: 3 x 3, padding: valid.

Cα-2: Convolutional layer. Filters: 64, kernel size: 7 x 1, padding: same.

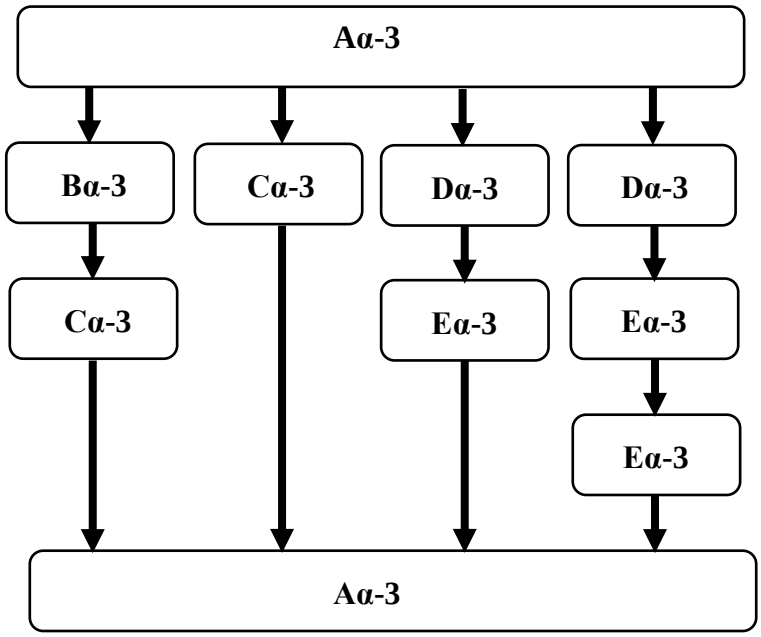
Dα-2: Convolutional layer. Filters: 64, kernel size: 1 x 7, padding: same.

Eα-2: Filter concatenation layer.

Fα-2: Convolutional layer. Filters: 192, kernel size: 3 x 3, padding: valid.

Gα-2: Max pooling layer. Stride: 2 x 2, padding: valid.

Figure 14. Part 1-2 of the Inception-v4-stem.



Aα-3: Filter concatenation layer.

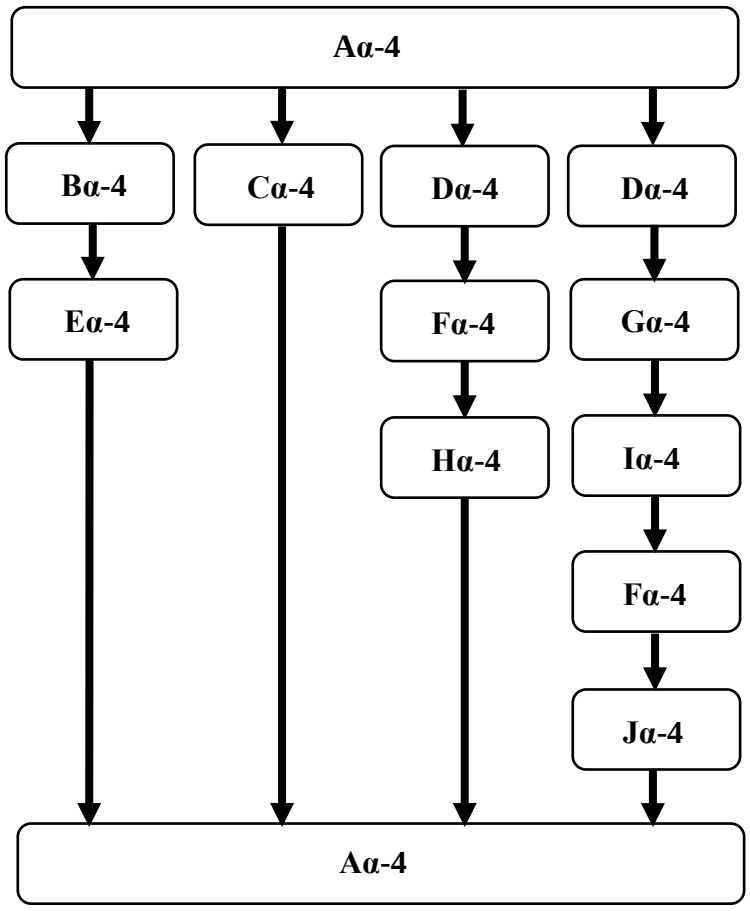
Bα-3: Average pooling layer.

Cα-3: Convolutional layer. Filters: 96, kernel size: 1 x 1, padding: same.

Dα-3: Convolutional layer. Filters: 64, kernel size: 1 x 1, padding: same.

Eα-3: Convolutional layer. Filters: 96, kernel size: 3 x 3, padding: same.

Figure 15. Inception-v4-A-module.



Aα-4: Filter concatenation layer.

Bα-4: Average pooling layer.

Cα-4: Convolutional layer. Filters: 384, kernel size: 1 x 1, padding: same.

Dα-4: Convolutional layer. Filters: 192, kernel size: 1 x 1, padding: same.

Eα-4: Convolutional layer. Filters: 128, kernel size: 1 x 1, padding: same.

Fα-4: Convolutional layer. Filters: 224, kernel size: 1 x 7, padding: same.

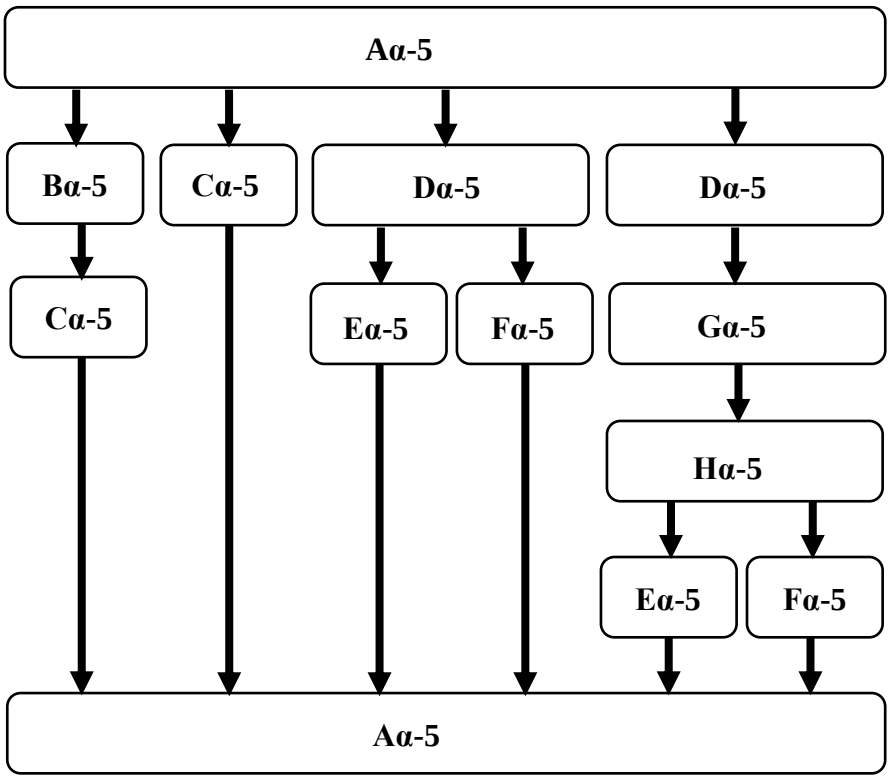
Gα-4: Convolutional layer. Filters: 192, kernel size: 1 x 7, padding: same.

Hα-4: Convolutional layer. Filters: 256, kernel size: 1 x 7, padding: same.

Iα-4: Convolutional layer. Filters: 224, kernel size: 7 x 1, padding: same.

Jα-4: Convolutional layer. Filters: 256, kernel size: 7 x 1, padding: same.

Figure 16. Inception-v4-B-module.



Aα-5: Filter concatenation layer.

Bα-5: Average pooling layer.

Cα-5: Convolutional layer. Filters: 256, kernel size: 1 x 1, padding: same.

Dα-5: Convolutional layer. Filters: 384, kernel size: 1 x 1, padding: same.

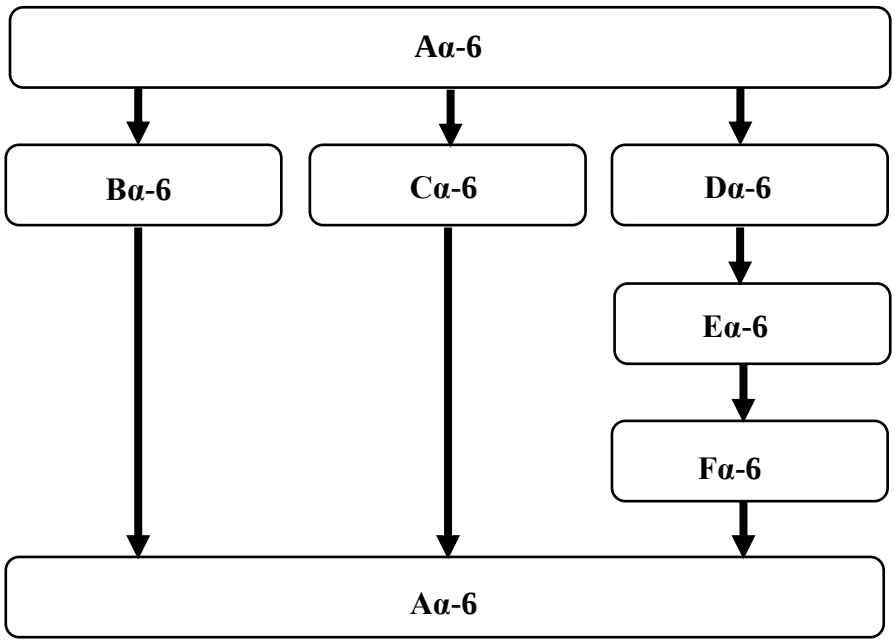
Eα-5: Convolutional layer. Filters: 256, kernel size: 1 x 3, padding: same.

Fα-5: Convolutional layer. Filters: 256, kernel size: 3 x 1, padding: same.

Gα-5: Convolutional layer. Filters: 448, kernel size: 1 x 3, padding: same.

Hα-5: Convolutional layer. Filters: 512, kernel size: 3 x 1, padding: same.

Figure 17. Inception-v4-C-module.



Aα-6: Filter concatenation layer.

Bα-6: Max pooling layer. Kernel size: 3 x 3, stride: 2 x 2, padding: valid.

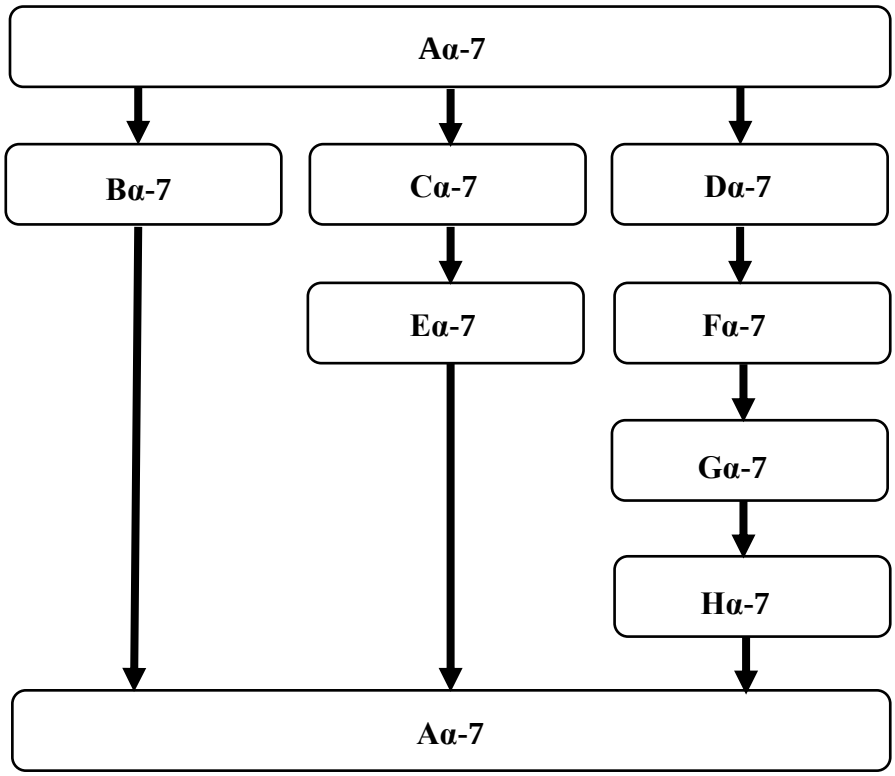
Cα-6: Convolutional layer. Filters: 384, kernel size: 3 x 3, stride: 2 x 2, padding: valid.

Dα-6: Convolutional layer. Filters: 192, kernel size: 1 x 1, padding: same.

Eα-6: Convolutional layer. Filters: 224, kernel size: 3 x 3, padding: same.

Fα-6: Convolutional layer. Filters: 256, kernel size: 3 x 1, stride: 2 x 2, padding:

Figure 18. Inception-v4-Reduction-A-module.



Aα-7: Filter concatenation layer.

Bα-7: Max pooling layer. Kernel size: 3 x 3, stride: 2 x 2, padding: valid.

Cα-7: Convolutional layer. Filters: 192, kernel size: 1 x 1, padding: same.

Dα-7: Convolutional layer. Filters: 256, kernel size: 1 x 1, padding: same.

Eα-7: Convolutional layer. Filters: 192, kernel size: 3 x 3, stride: 2 x 2, padding: valid.

Fα-7: Convolutional layer. Filters: 256, kernel size: 1 x 7, padding: same.

Gα-7: Convolutional layer. Filters: 320, kernel size: 7 x 1, padding: same.

Hα-7: Convolutional layer. Filters: 320, kernel size: 3 x 3, stride: 2 x 2, padding: valid.

Figure 19. Inception-v4-Reduction-B-module.

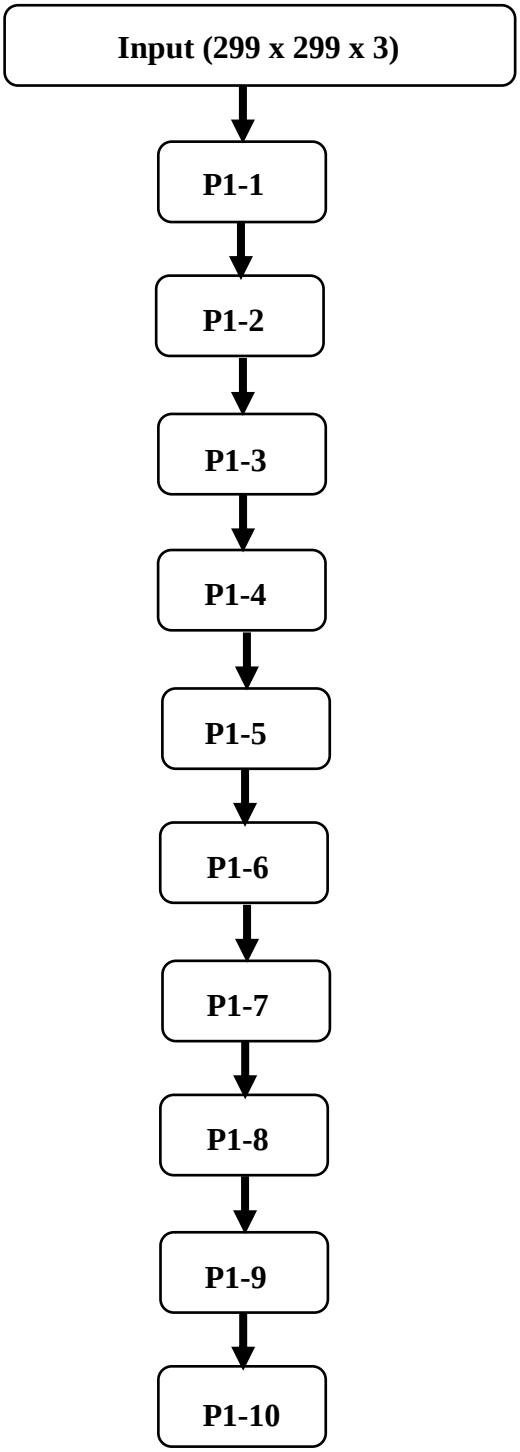


Figure 20. Full Inception-v4-architecture.

P1-1: Part 1 of the Inception-v4-stem.

P1-2: Part 2 of the Inception-v4-stem.

P1-3: 4 x Inception-v4-A-module.

P1-4: Inception-v4-Reduction-A-module.

P1-5: 7 x Inception-v4-B-module.

P1-6: Inception-v4-Reduction-B-module.

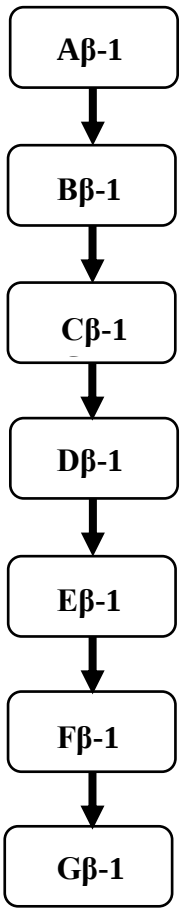
P1-7: 3 x Inception-v4-C-module.

P1-8: Average pooling layer.

P1-9: Dropout layer (keep 0.8).

P1-10: Softmax layer.

Appendix 2. Original Inception-ResNet-v1 Architecture



Aβ-1: Convolutional layer. Filters: 32, kernel size: 3 x 3, stride: 2 x 2, padding: valid.

Bβ-1: Convolutional layer. Filters: 32, kernel size: 3 x 3, padding: valid.

Cβ-1: Convolutional layer. Filters: 64, kernel size: 3 x 3, padding: same.

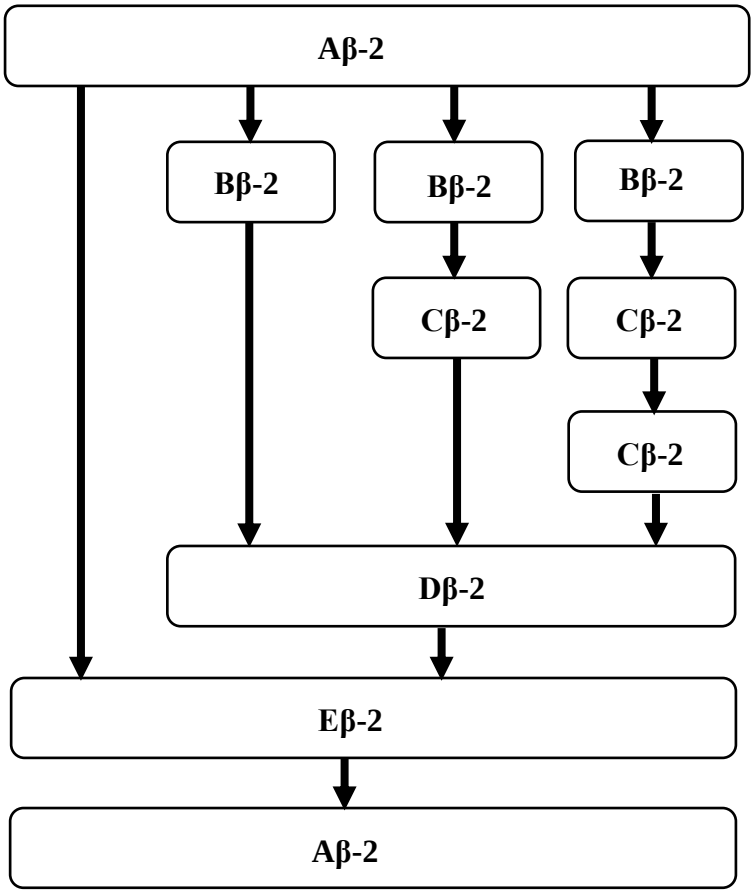
Dβ-1: Max pooling layer. Kernel size: 3 x 3, stride: 2 x 2, padding: valid.

Eβ-1: Convolutional layer. Filters: 80, kernel size: 1 x 1, padding: same.

Fβ-1: Convolutional layer. Filters: 192, kernel size: 3 x 3, padding: valid.

Gβ-1: Convolutional layer. Filters: 256, kernel size: 3 x 3, stride: 2 x 2, padding: valid.

Figure 21. Inception-ResNet-v1-stem.



Aβ-2: ReLU-activation layer.

Bβ-2: Convolutional layer. Filters: 32, kernel size: 1 x 1, padding: same.

Cβ-2: Convolutional layer. Filters: 32, kernel size: 3 x 3, padding: same.

Dβ-2: Convolutional layer. Filters: 256, kernel size: 1 x 1, padding: same, linear.

Eβ-2: Addition layer.

Figure 22. Inception-ResNet-v1-A-module.

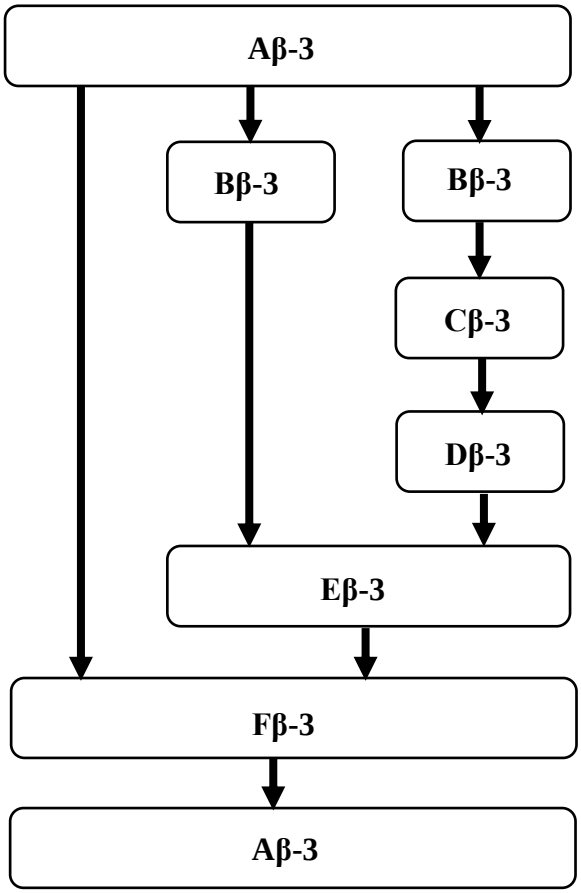


Figure 23. Inception-ResNet-v1-B-module.

Aβ-3: ReLU-activation layer.

Bβ-3: Convolutional layer. Filters: 128, kernel size: 1 x 1, padding: same.

Cβ-3: 2D convolutional layer. Filters: 128, kernel size: 1 x 7, padding: same.

Dβ-3: 2D convolutional layer. Filters: 128, kernel size: 7 x 1, padding: same.

Eβ-3: Convolutional layer. Filters: 896, kernel size: 1 x 1, padding: same, linear.

Fβ-3: Addition layer.

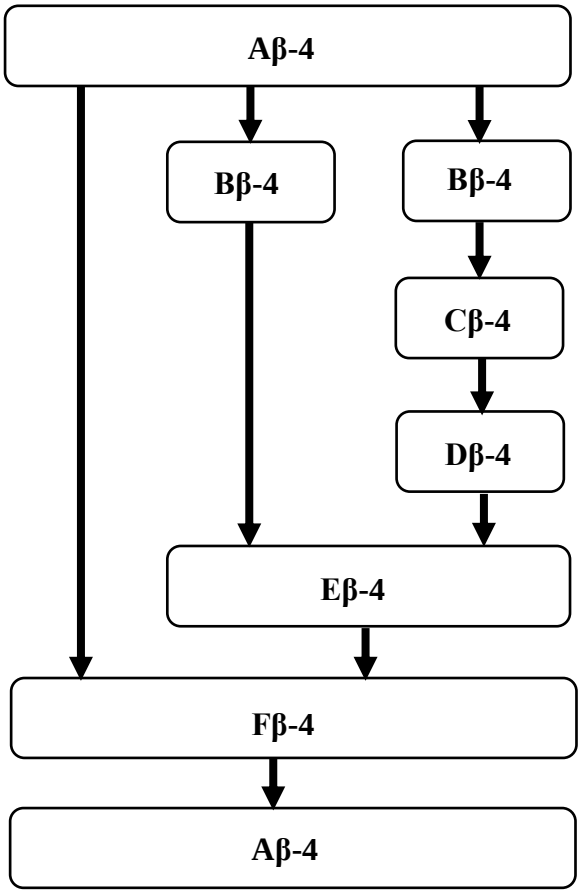


Figure 24. Inception-ResNet-v1-C-module.

Aβ-4: ReLU-activation layer.

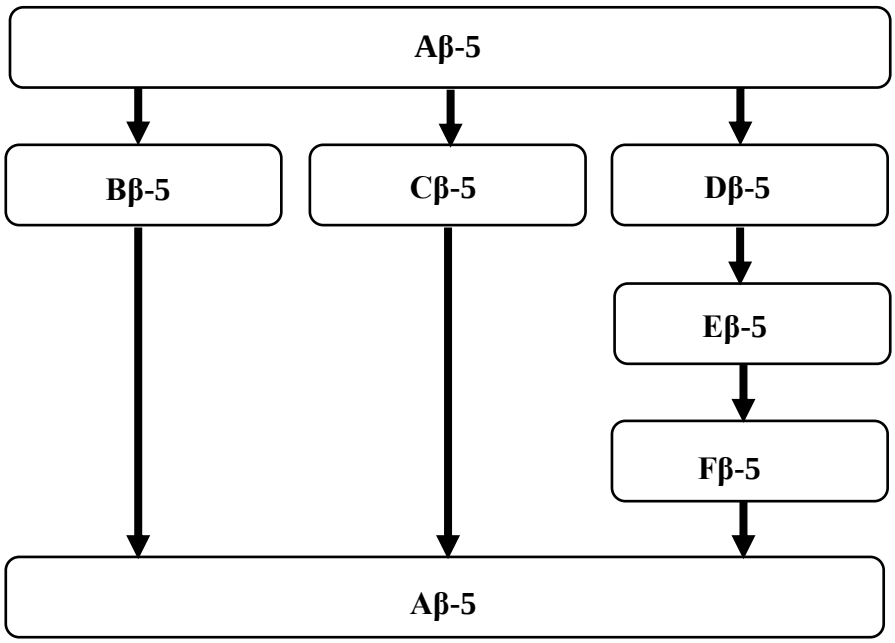
Bβ-4: Convolutional layer. Filters: 192, kernel size: 1 x 1, padding: same.

Cβ-4: Convolutional layer. Filters: 192, kernel size: 1 x 3, padding: same.

Dβ-4: 2D convolutional layer. Filters: 192, kernel size: 3 x 1, padding: same.

Eβ-4: Convolutional layer. Filters: 1792, kernel size: 1 x 1, padding: same, linear.

Fβ-4: Addition layer.



Aβ-5: Filter concatenation layer.

Bβ-5: Max pooling layer. Kernel size: 3 x 3, stride: 2 x 2, padding: valid.

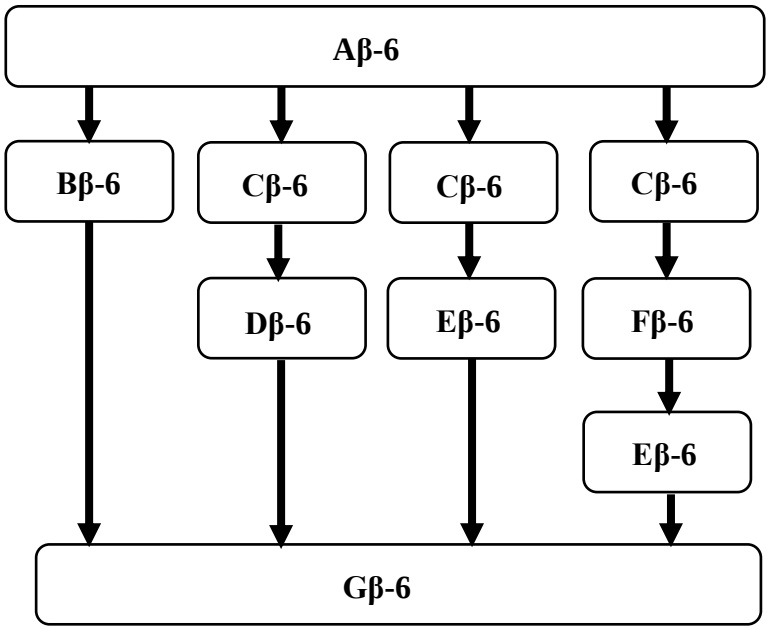
Cβ-5: Convolutional layer. Filters: 384, kernel size: 3 x 3, stride: 2 x 2, padding: valid.

Dβ-5: Convolutional layer. Filters: 192, kernel size: 1 x 1, padding: same.

Eβ-5: Convolutional layer. Filters: 192, kernel size: 3 x 3, padding: same.

Fβ-5: Convolutional layer. Filters: 256, kernel size: 3 x 1, stride: 2 x 2, padding:

Figure 25. Inception-ResNet-v1-Reduction-A-module.



Aβ-6: Previous layer.

Bβ-6: Max pooling layer. Kernel size: 3 x 3, stride: 2 x 2, padding: valid.

Cβ-6: Convolutional layer. Filters: 256, kernel size: 1 x 1, padding: same.

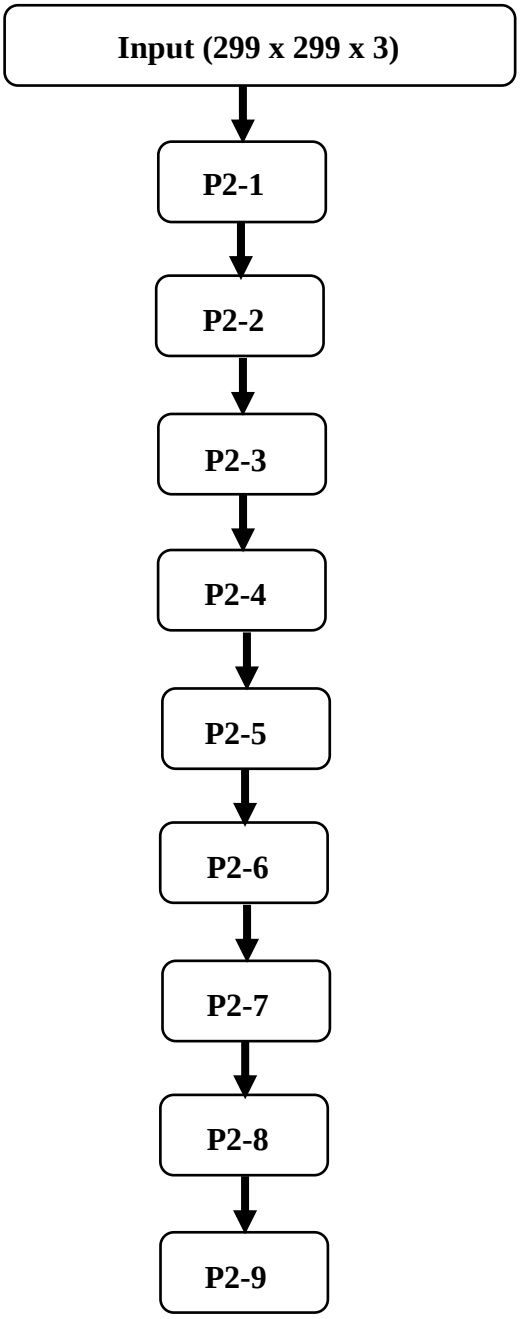
Dβ-6: Convolutional layer. Filters: 384, kernel size: 3 x 3, stride: 2 x 2, padding: valid.

Eβ-6: Convolutional layer. Filters: 256, kernel size: 3 x 3, stride: 2 x 2, padding: valid.

Fβ-6: Convolutional layer. Filters: 256, kernel size: 3 x 3, padding: same.

Gβ-6: Filter concatenation layer.

Figure 26. Inception-ResNet-v1-Reduction-B-module.



P2-1: Inception-ResNet-v1-stem.

P2-2: 5 x Inception-ResNet-v1-A-module.

P2-3: Inception-ResNet-v1-Reduction-A-module.

P2-4: 10 x Inception-ResNet-v1-B-module.

P2-5: Inception-ResNet-v1-Reduction-B-module.

P2-6: 5 x Inception-ResNet-v1-C-module.

P2-7: Average pooling layer.

P2-8: Dropout layer (keep 0.8).

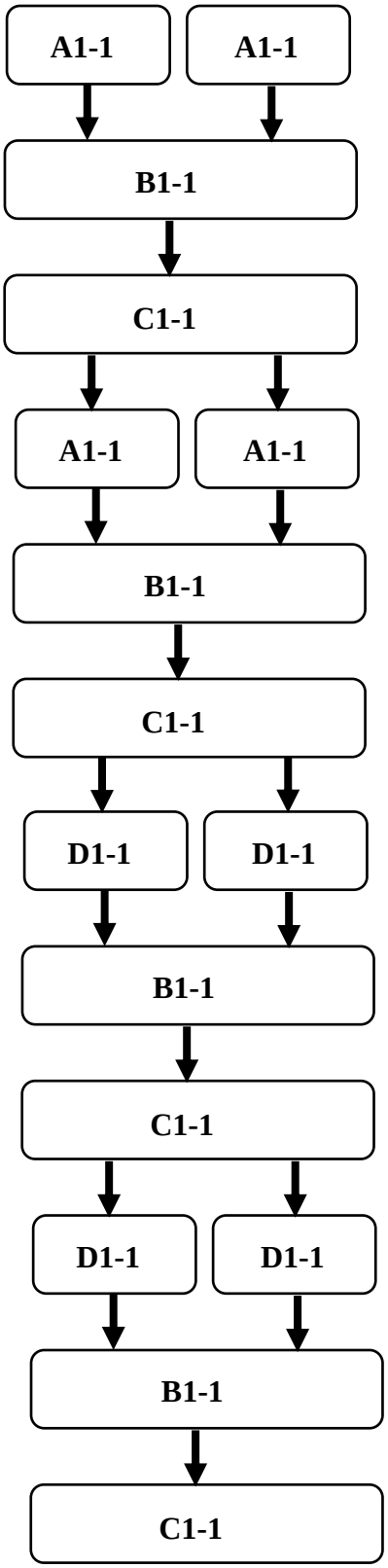
P2-9: Softmax layer.

Figure 27. Full Inception-ResNet-v1-architecture.

Appendix 3. CNN-architecture for the 70 x 70 and 128 x 128 data sets

The 70 x 70 and the 128 x 128 CNN-architectures used the same architecture structure with two tracks. The number of filter on the respective layers varied from 3, at the initial levels, to 4, at the final levels (see Figures 28-30). The 70 x 70 architecture had a total of 1 031 parameters, with all 1 031 of these being trainable parameters. On the other hand, the 128 x 128 architecture had a total of 1 799 parameters, with all of them being trainable.

To create a comparison architecture to the CNN-architecture, a single-tracked 70 x 70 version of the CNN-architecture was also constructed (see Figures 31-33). This architecture required 648 parameters, with all of these parameters being trainable. In the double-tracked and single-tracked CNN-architecture, ReLU-activations were performed in the convolutional layers before the addition layers. As $\max(o, x) \neq x$, the input to the addition layers were not linear. Thus, the double-tracked convolutional layers in the double-tracked CNN-architecture are not interchangeable with a single convolutional layer.



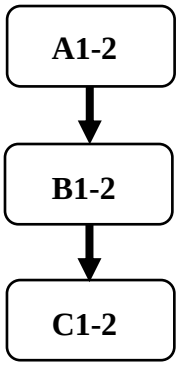
A1-1: 2D convolutional layer. Filters: 3, kernel size: 3 x 3, stride: 1 x 1, padding: same, activation function: ReLU.

B1-1: Addition layer.

C1-1: 2D max pooling layer. Pool size: 2 x 2.

D1-1: 2D convolutional layer. Filters: 4, kernel size: 3 x 3, stride: 1 x 1, padding: same, activation function: ReLU.

Figure 28. Section 1 of the CNN-architecture.

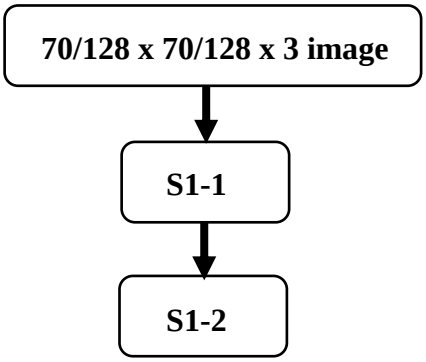


A1-2: Flatten layer.

B1-2: Dense layer. Output space dimensionality: 4, activation function: ReLU.

C1-2: Dense layer. Output space dimensionality: 1, activation function: Sigmoid.

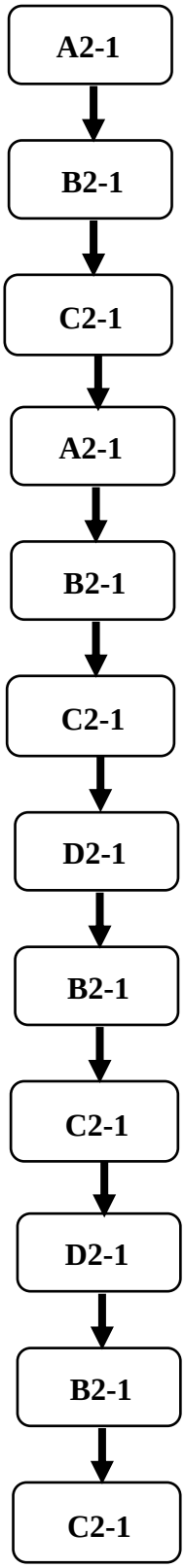
Figure 29. Section 2 of the CNN-architecture.



S1-1: Section 1 of the CNN-architecture.

S1-2: Section 2 of the CNN-architecture.

Figure 30. Full CNN-architecture.



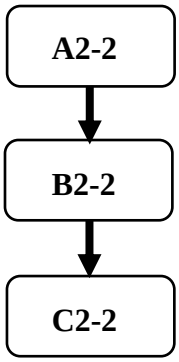
A2-1: 2D convolutional layer. Filters: 3, kernel size: 3 x 3, stride: 1 x 1, padding: same, activation function: ReLU.

B2-1: Addition layer.

C2-1: 2D max pooling layer. Pool size: 2 x 2.

D2-1: 2D convolutional layer. Filters: 4, kernel size: 3 x 3, stride: 1 x 1, padding: same, activation function: ReLU.

Figure 31. Section 1 of the 70 x 70 data set single-tracked CNN-architecture.

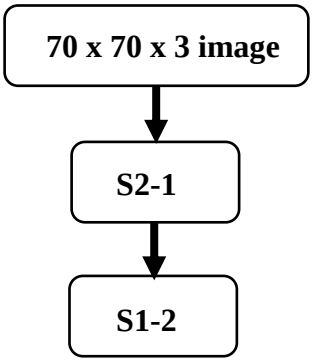


A2-2: Flatten layer.

B1-2: Dense layer. Output space dimensionality: 4, activation function: ReLU.

C1-2: Dense layer. Output space dimensionality: 1, activation function: Sigmoid.

Figure 32. Section 2 of the 70 x 70 data set single-tracked CNN-architecture.



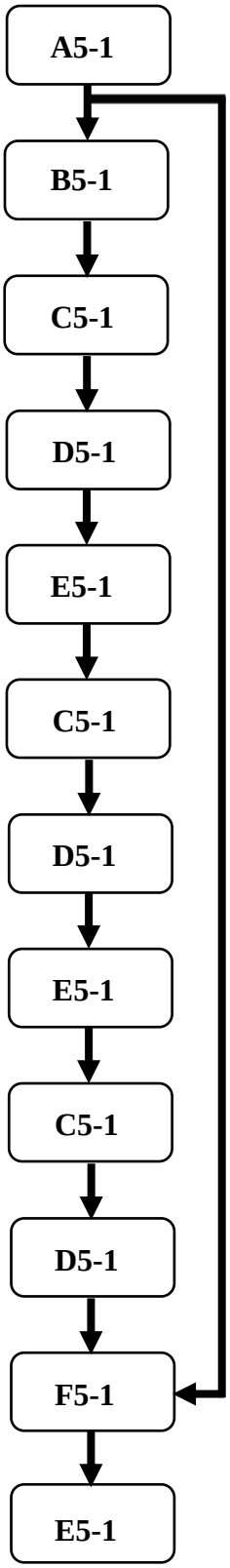
S2-1: Section 1 of the 70 x 70 data set single-tracked CNN-architecture.

S1-2: Section 2 of the 70 x 70 data set single-tracked CNN-architecture.

Figure 33. Full 70 x 70 data set single-tracked CNN-architecture.

Appendix 4. ResNet-architecture for the 70 x 70 data set

The architectures for the 70 x 70 and 128 x 128 ResNet-architectures differed in their layer filter counts. 70 x 70 ResNet-architecture filter counts on the respective layers varied from 63, at the initial levels, to 75, at the final levels (see Figures 34-39). The architecture had a total of 507 180 parameters, with 506 850 of these being trainable parameters.



A5-1: 2D convolutional layer. Filters: 63, kernel size: 3 x 3, stride: 1 x 1, padding: valid, activation function: ReLU.

B5-1: 2D max pooling layer. Pool size: 2 x 2.

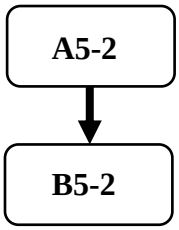
C5-1: 2D convolutional layer. Filters: 63, kernel size: 3 x 3, stride: 1 x 1, padding: same.

D5-1: Batch normalization layer.

E5-1: ReLU-activation layer.

F5-1: Addition layer.

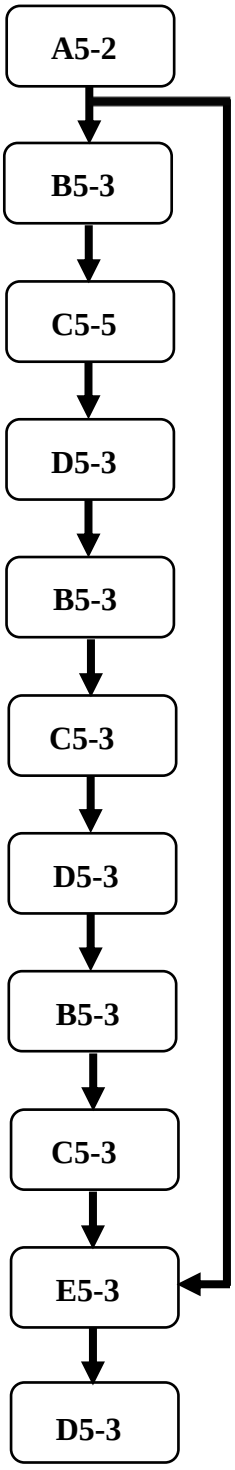
Figure 34. Section 1 of the 70 x 70 data set ResNet-architecture.



A5-2: 2D convolutional layer. Filters: 67, kernel size: 3 x 3, stride: 1 x 1, padding: valid, activation function: ReLU.

B5-2: 2D max pooling layer. Pool size: 2 x 2.

Figure 35. Section 2 of the 70 x 70 data set ResNet-architecture.



A5-3: 2D convolutional layer. Filters: 67, kernel size: 3 x 3, stride: 1 x 1, padding: valid, activation function: ReLU.

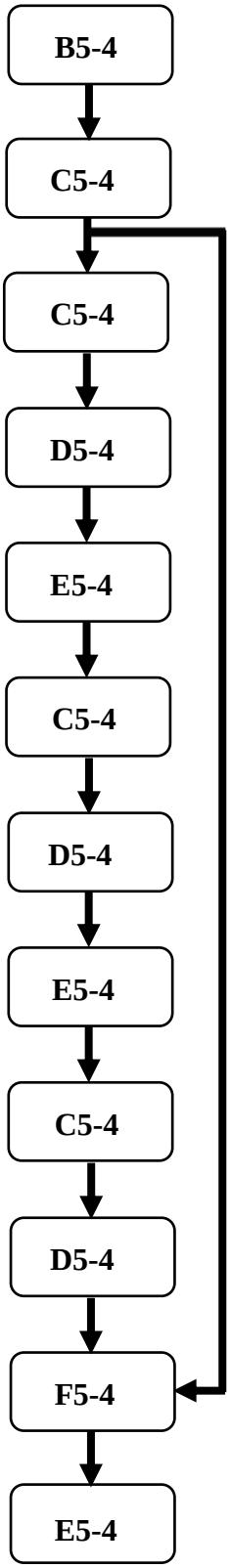
B5-3: 2D convolutional layer. Filters: 67, kernel size: 3 x 3, stride: 1 x 1, padding: same.

C5-3: Batch normalization layer.

D5-3: ReLU-activation layer.

E5-3: Addition layer.

Figure 36. Section 3 of the 70 x 70 data set ResNet-architecture.



B5-4: 2D max pooling layer. Pool size: 2 x 2.

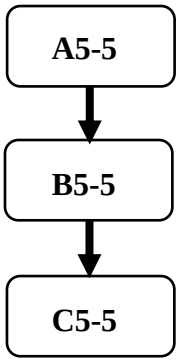
C5-4: 2D convolutional layer. Filters: 75, kernel size: 3 x 3, stride: 1 x 1, padding: same.

D5-4: Batch normalization layer.

E5-4: ReLU-activation layer.

F5-4: Addition layer.

Figure 37. Section 4 of the 70 x 70 data set ResNet-architecture.

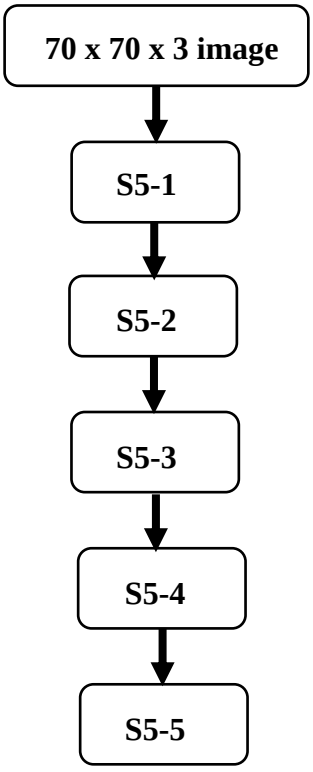


A5-5: 2D average pooling layer. Pool size: 4 x 4.

B5-5: Flatten layer.

C5-5: Dense layer. Output space dimensionality: 1, activation function: Sigmoid.

Figure 38. Section 5 of the 70 x 70 data set ResNet-architecture.



S5-1: Section 1 of the 70 x 70 data set ResNet-architecture.

S5-2: Section 2 of the 70 x 70 data set ResNet-architecture.

S5-3: Section 3 of the 70 x 70 data set ResNet-architecture.

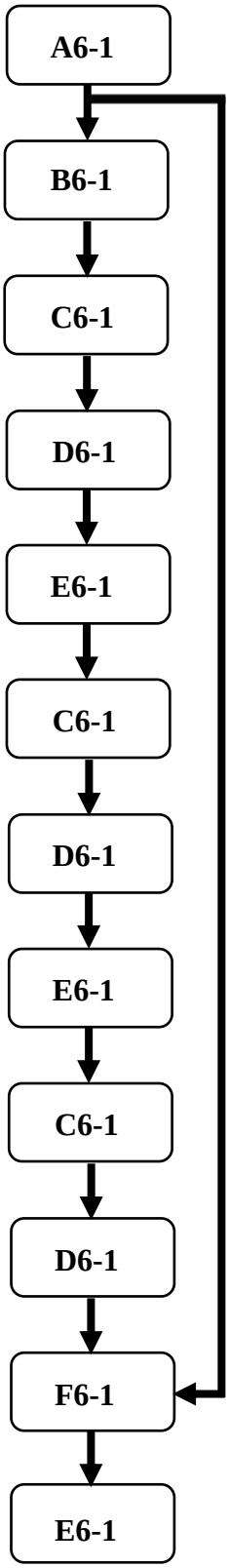
S5-4: Section 4 of the 70 x 70 data set ResNet-architecture.

S5-5: Section 5 of the 70 x 70 data set ResNet-architecture.

Figure 39. Full 70 x 70 data set ResNet-architecture.

Appendix 5. ResNet-architecture for the 128 x 128 data set

The 128 x 128 ResNet-architecture had the following structure: Filter counts on the respective layers progressively increased from 58, at the initial levels, to 70, at the final levels (see Figures 40-45). The architecture had a total of 436 865 parameters, with 630 of these being non-trainable.



A6-1: 2D convolutional layer. Filters: 58, kernel size: 3 x 3, stride: 1 x 1, padding: valid, activation function: ReLU.

B6-1: 2D max pooling layer. Pool size: 2 x 2.

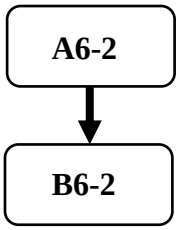
C6-1: 2D convolutional layer. Filters: 58, kernel size: 3 x 3, stride: 1 x 1, padding: same.

D6-1: Batch normalization layer.

E6-1: ReLU-activation layer.

F6-1: Addition layer.

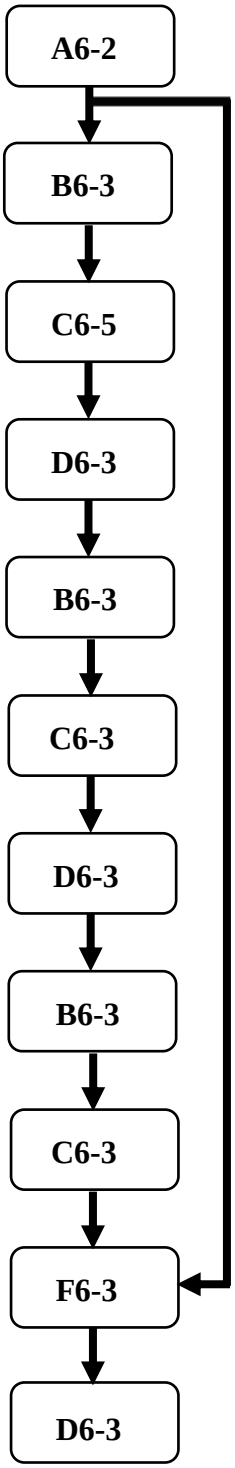
Figure 40. Section 1 of the 128 x 128 data set ResNet-architecture.



A6-2: 2D convolutional layer. Filters: 62, kernel size: 3 x 3, stride: 1 x 1, padding: valid, activation function: ReLU.

B6-2: 2D max pooling layer. Pool size: 2 x 2.

Figure 41. Section 2 of the 128 x 128 data set ResNet-architecture.



A6-3: 2D convolutional layer. Filters: 62, kernel size: 3 x 3, stride: 1 x 1, padding: valid, activation function: ReLU.

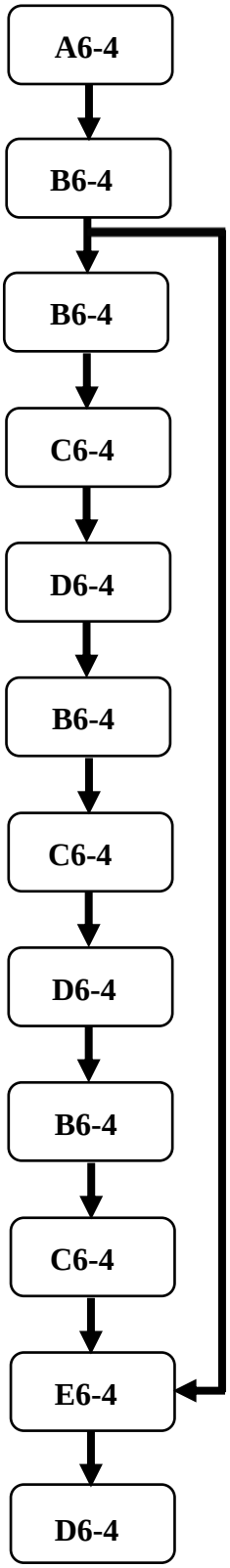
B6-3: 2D convolutional layer. Filters: 62, kernel size: 3 x 3, stride: 1 x 1, padding: same.

C6-3: Batch normalization layer.

D6-3: ReLU-activation layer.

F6-3: Addition layer.

Figure 42. Section 3 of the 128 x 128 data set ResNet-architecture.



A6-4: 2D max pooling layer. Pool size: 2 x 2.

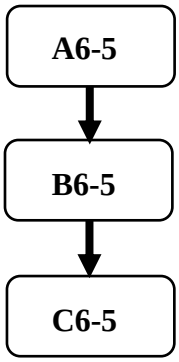
B6-4: 2D convolutional layer. Filters: 70, kernel size: 3 x 3, stride: 1 x 1, padding: same.

C6-4: Batch normalization layer.

D6-4: ReLU-activation layer.

E6-4: Addition layer.

Figure 43. Section 4 of the 128 x 128 data set ResNet-architecture.

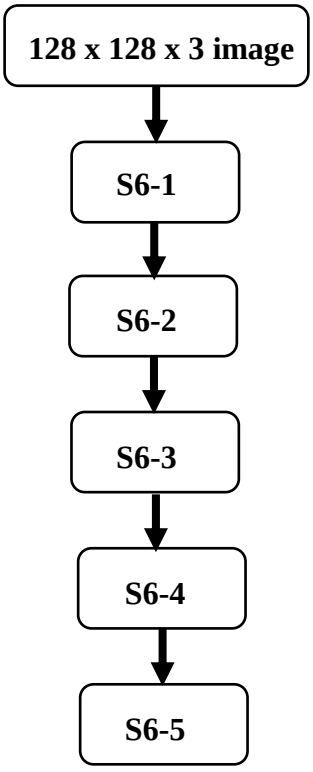


A6-5: 2D average pooling layer. Pool size: 7 x 7.

B6-5: Flatten layer.

C6-5: Dense layer. Output space dimensionality: 1, activation function: Sigmoid.

Figure 44. Section 5 of the 128 x 128 data set ResNet-architecture.



S6-1: Section 1 of the 128 x 128 data set ResNet-architecture.

S6-2: Section 2 of the 128 x 128 data set ResNet-architecture.

S6-3: Section 3 of the 128 x 128 data set ResNet-architecture.

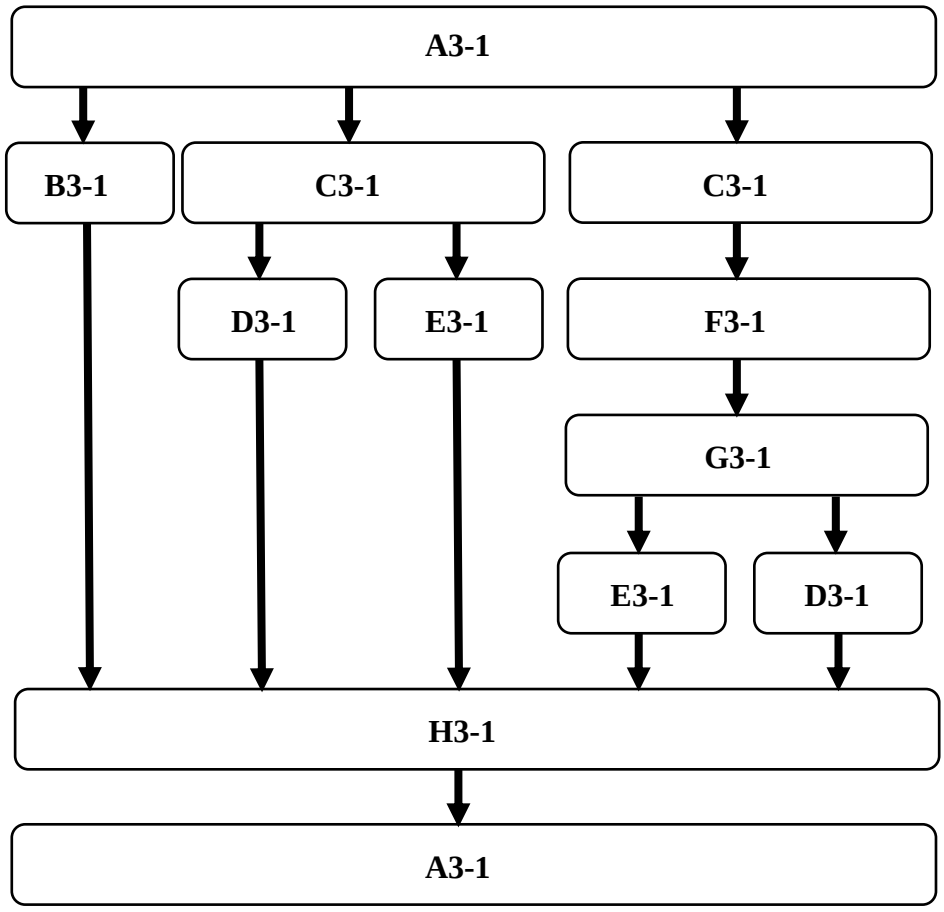
S6-4: Section 4 of the 128 x 128 data set ResNet-architecture.

S6-5: Section 5 of the 128 x 128 data set ResNet-architecture.

Figure 45. Full 128 x 128 data set ResNet-architecture.

Appendix 6. CNN-architecture with an Inception-v4-module for the 70 x 70 data set

The number of filters on the respective layers for the 70 x 70 Inception-v4 modified CNN varied from 298 to 362 (see Figures 46-48). The architecture had a total of 1 963 347 parameters, with all of them being trainable. Thus, the 70 x 70 Inception-v4 modified CNN-architecture required approximately 1 900 times more parameters to train than the base 70 x 70 CNN-architecture.



A3-1: 2D average pooling layer. Pool size: 4 x 4.

B3-1: 2D convolutional layer. Filters: 298, kernel size: 1 x 1, stride: 1 x 1, padding: same.

C3-1: 2D convolutional layer. Filters: 330, kernel size: 1 x 1, stride: 1 x 1, padding: same.

D3-1: 2D convolutional layer. Filters: 298, kernel size: 1 x 3, stride: 1 x 1, padding: same.

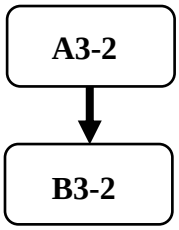
E3-1: 2D convolutional layer. Filters: 298, kernel size: 3 x 1, stride: 1 x 1, padding: same.

F3-1: 2D convolutional layer. Filters: 346, kernel size: 1 x 3, stride: 1 x 1, padding: same.

G3-1: 2D convolutional layer. Filters: 362, kernel size: 3 x 1, stride: 1 x 1, padding: same.

H3-1: Addition layer.

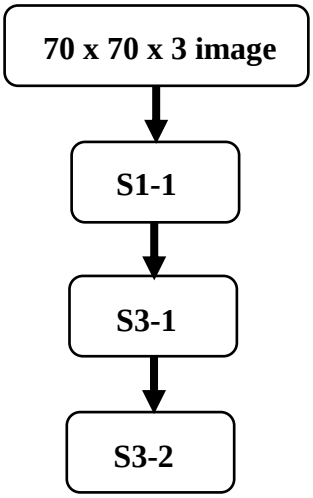
Figure 46. Section 1 of the 70 x 70 data set CNN-Inception-v4-module architecture.



A3-2: Flatten layer.

B3-2: Dense layer. Output space dimensionality: 1, activation function: Sigmoid.

Figure 47. Section 2 of the 70 x 70 data set CNN-Inception-v4-module architecture.



S1-1: Section 1 of the CNN-architecture.

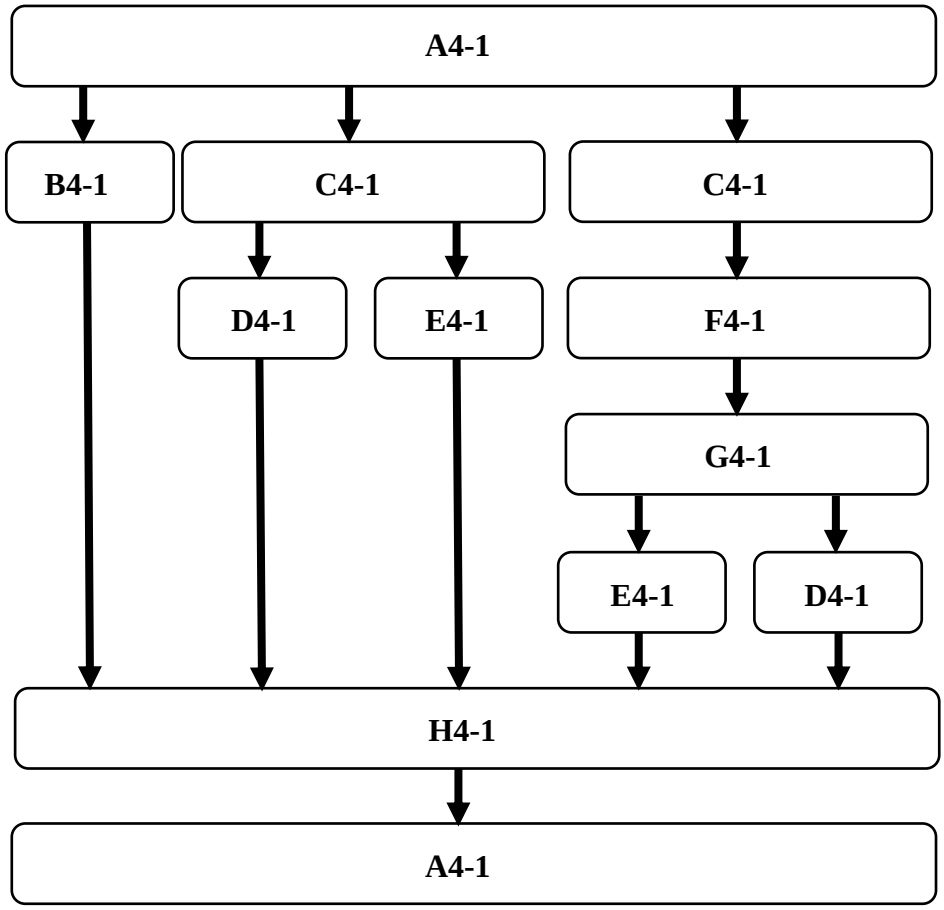
S3-1: Section 1 of the 70 x 70 data set CNN-Inception-v4-module-architecture.

S3-2: Section 2 of the 70 x 70 data set CNN-Inception-v4-module-architecture.

Figure 48. Full 70 x 70 data set CNN-Inception-v4-module-architecture.

Appendix 7. CNN-architecture with an Inception-v4-module for the 128 x 128 data set

As with the 70 x 70 Inception-v4 modified CNN, the number of filters on the respective layers for the 128 x 128 Inception-v4 modified CNN varied from 298 to 362 (see Figures 49-50). In total, the architecture had 1 963 347 parameters, with no non-trainable parameters. Compared with the 128 x 128 CNN-architecture, approximately 1 090 times more parameters were required to train 128 x 128 Inception-v4 modified CNN.



A4-1: 2D average pooling layer. Pool size: 8 x 8.

B4-1: 2D convolutional layer. Filters: 298, kernel size: 1 x 1, stride: 1 x 1, padding: same.

C4-1: 2D convolutional layer. Filters: 330, kernel size: 1 x 1, stride: 1 x 1, padding: same.

D4-1: 2D convolutional layer. Filters: 298, kernel size: 1 x 3, stride: 1 x 1, padding: same.

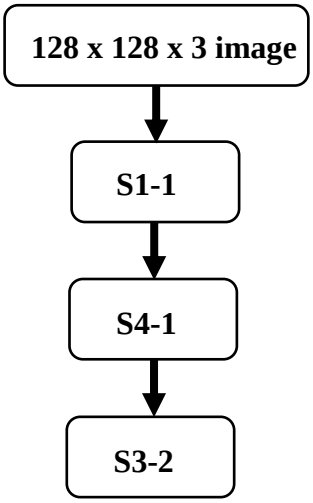
E4-1: 2D convolutional layer. Filters: 298, kernel size: 3 x 1, stride: 1 x 1, padding: same.

F4-1: 2D convolutional layer. Filters: 346, kernel size: 1 x 3, stride: 1 x 1, padding: same.

G4-1: 2D convolutional layer. Filters: 362, kernel size: 3 x 1, stride: 1 x 1, padding: same.

H4-1: Addition layer.

Figure 49. Section 1 of the 128 x 128 data set CNN-Inception-v4-module-architecture.



S1-1: Section 1 of the CNN-architecture.

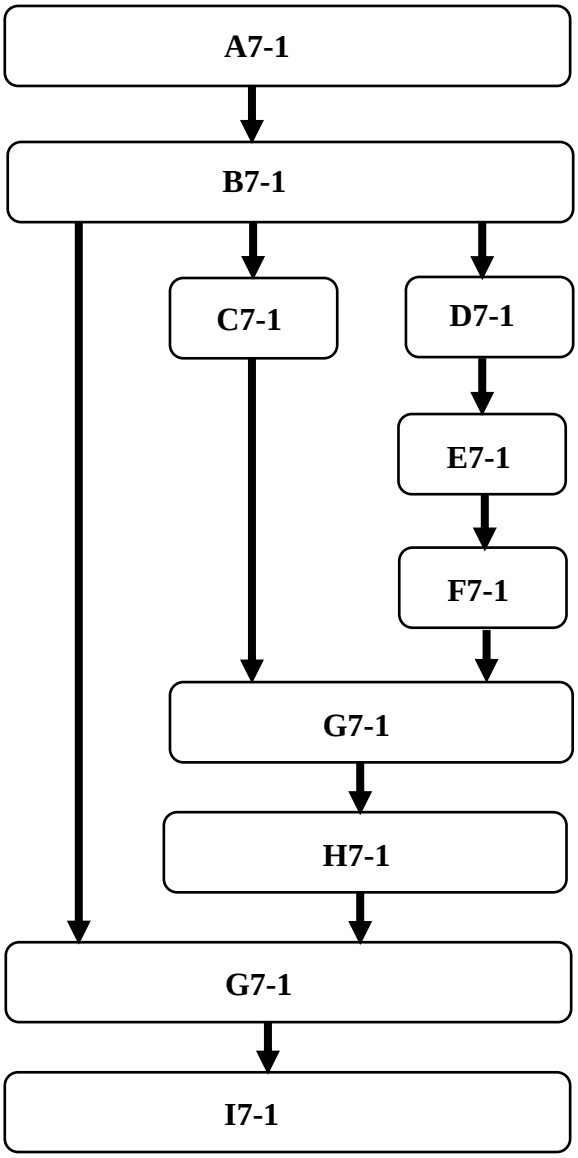
S4-1: Section 1 of the 128 x 128 data set CNN-Inception-v4-module-architecture.

S3-2: Section 2 of the 70 x 70 data set CNN-Inception-v4-module-architecture.

Figure 50. Full 128 x 128 data set CNN-Inception-v4-module-architecture.

Appendix 8. CNN-architecture with an Inception-ResNet-v1-module for the 70 x 70 data set

The number of filter on the respective layers for the 70 x 70 Inception-ResNet-v1 modified CNN-architecture varied from 4 to 57 (see Figures 51-52). The total parameter count for the architecture was 16 629, with eight non-trainable parameters. Compared with the base 70 x 70 CNN-architecture, the 70 x 70 Inception-ResNet-v1 modified CNN-architecture required approximately 16 times more parameters to train.



A7-1: Batch normalization layer.

B7-1: ReLU-activation layer.

C7-1: 2D convolutional layer. Filters: 57, kernel size: 1 x 1, stride: 1 x 1, padding: same, activation function: ReLU.

D7-1: 2D convolutional layer. Filters: 45, kernel size: 1 x 1, stride: 1 x 1, padding: same, activation function: ReLU.

E7-1: 2D convolutional layer. Filters: 49, kernel size: 1 x 3, stride: 1 x 1, padding: same, activation function: ReLU.

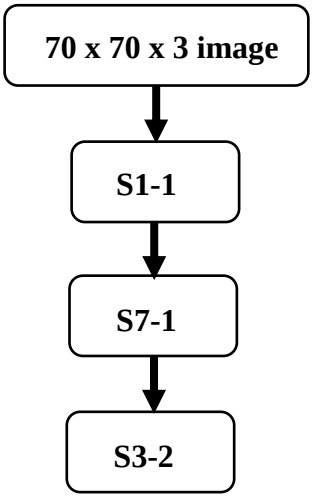
F7-1: 2D convolutional layer. Filters: 57, kernel size: 3 x 1, stride: 1 x 1, padding: same, activation function: ReLU.

G7-1: Addition layer.

H7-1: 2D convolutional layer. Filters: 4, kernel size: 1 x 1, stride: 1 x 1, padding: same.

I7-1: 2D average pooling layer. Pool size: 4 x 4.

Figure 51. Section 1 of the 70 x 70 data set CNN-Inception-ResNet-v1-module-architecture.



S1-1: Section 1 of the CNN-architecture.

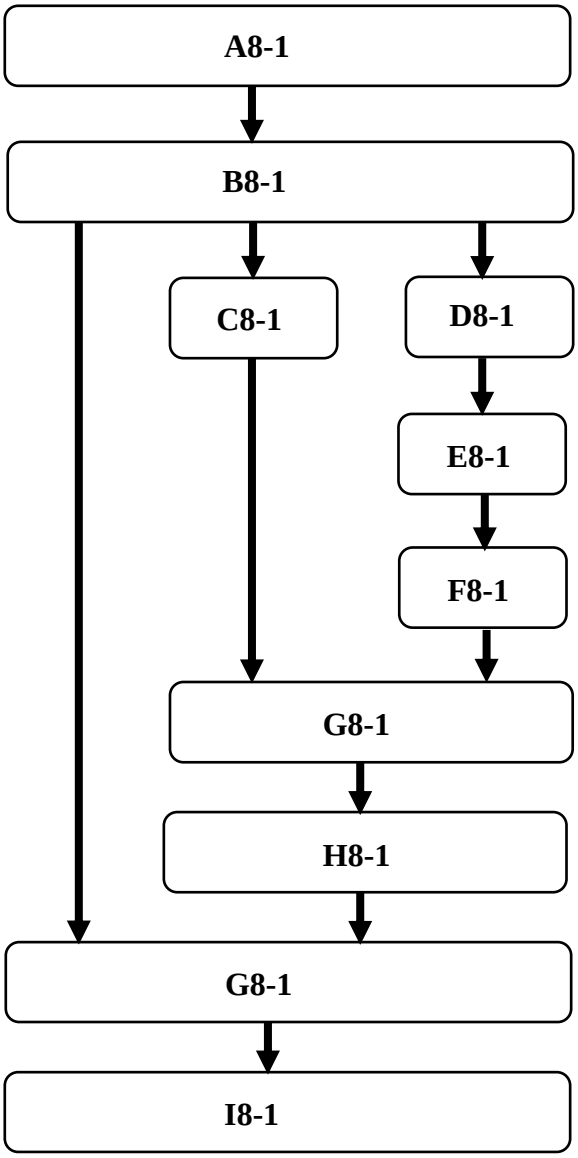
S7-1: Section 1 of the 70 x 70 data set CNN-Inception-ResNet-module-architecture.

S3-2: Section 2 of the 70 x 70 data set CNN-Inception-v4-module-architecture.

Figure 52. Full 70 x 70 data set CNN-Inception-ResNet-v1-module-architecture.

Appendix 9. CNN-architecture with an Inception-ResNet-v1-module for the 128 x 128 data set

The number of filter on the respective layers varied from 4 to 56 (see Figures 53-54). The total parameter count for the architecture was 16 047, with 16 non-trainable parameters. The 128 x 128 Inception-ResNet-v1 modified CNN-architecture required approximately 9 times more parameters than the base 128 x 128 CNN-architecture.



A8-1: Batch normalization layer.

B8-1: ReLU-activation layer.

C8-1: 2D convolutional layer. Filters: 56, kernel size: 1 x 1, stride: 1 x 1, padding: same, activation function: ReLU.

D8-1: 2D convolutional layer. Filters: 44, kernel size: 1 x 1, stride: 1 x 1, padding: same, activation function: ReLU.

E8-1: 2D convolutional layer. Filters: 48, kernel size: 1 x 3, stride: 1 x 1, padding: same, activation function: ReLU.

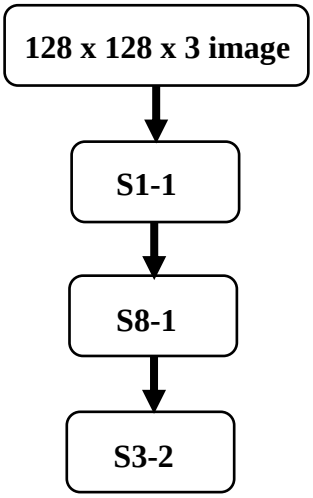
F8-1: 2D convolutional layer. Filters: 56, kernel size: 3 x 1, stride: 1 x 1, padding: same, activation function: ReLU.

G8-1: Addition layer.

H8-1: 2D convolutional layer. Filters: 4, kernel size: 1 x 1, stride: 1 x 1, padding: same.

I8-1: 2D average pooling layer. Pool size: 4 x 4.

Figure 53. Section 1 of the 128 x 128 data set CNN-Inception-ResNet-v1-module-architecture.



S1-1: Section 1 of the CNN-architecture.

S8-1: Section 1 of the 128 x 128 data set CNN-Inception-ResNet-module-architecture.

S3-2: Section 2 of the 70 x 70 data set CNN-Inception-v4-module-architecture.

Figure 54. Full 128 x 128 data set CNN-Inception-ResNet-v1-module-architecture.

Appendix 10. Plotted results for the CNN-architecture for the 70 x 70 data set

From the accuracy plots (see Figure 55), we can see that training accuracy increased over all 6 000 epochs. The validation accuracy peaked at around 89 % approximately on epoch 1 500. After epoch 1 500, validation accuracy began to decrease for folds 1 and 5, but stayed at approximately the same level for the other three folds. We can also see that the accuracies varied between epochs, with the training accuracies varying approximately 1 percentage point between epochs. For the validation accuracies, the corresponding accuracy variations between epochs were approximately 2-3 percentage points.

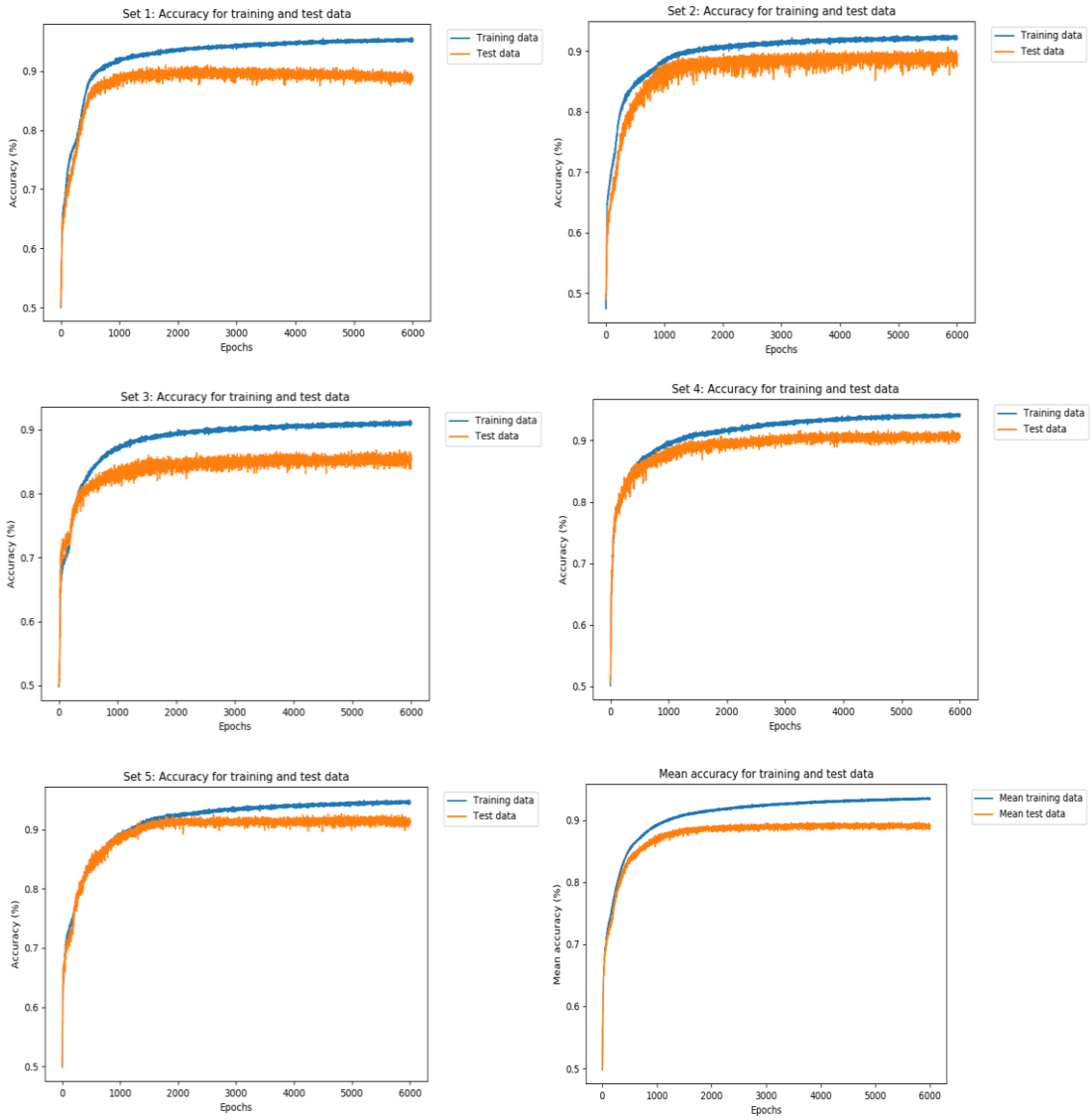


Figure 55. 70 x 70 data set CNN-architecture 5-fold cross-validation fold training and validation accuracies. (Upper left) First set, (Upper right) Second set, (Middle left) Third set, (Middle right) Fourth set, (Lower left) Fifth set, (Lower right) and Mean training and validation accuracies for all five cross-validation folds of the 70 x 70 data set CNN-architecture.

The double-tracked CNN used as the CNN of choice in thesis, achieved quite high accuracy results for 70 x 70 images. To determine if use of a double-tracked CNN is to be preferred to a single-tracked version of the same architecture, a shorter 5-fold cross-validation test with a single-tracked version of the CNN-architecture was also performed. The single-tracked CNN-test took approximately 17 hours to run.

From the mean accuracy plot (see Figure 56) we can see that the mean training accuracy for the single-tracked CNN was still increasing in epoch 1 000, however, the mean validation accuracy had levelled out at approximately 81 % after epoch 900. If we compare the observed single-tracked CNN mean validation accuracy at this point with the validation accuracy for the double-tracked CNN at the same point, we can see that the double-tracked CNN had a higher validation accuracy. At epoch 1 000, we can see that the mean validation accuracy for the double-tracked CNN was approximately 87 %, six percentage points higher than the mean validation accuracy for the single-tracked CNN. The reason why the double-tracked CNN performed better than the single-tracked version has not been investigated in depth. One reason might have been due to the double-tracked architecture being able to detect more patterns and combinations of patterns. The reason for this could have been the use of ReLU-activations on both tracks in the convolution layers before adding together the results of the convolutions in the addition layers.

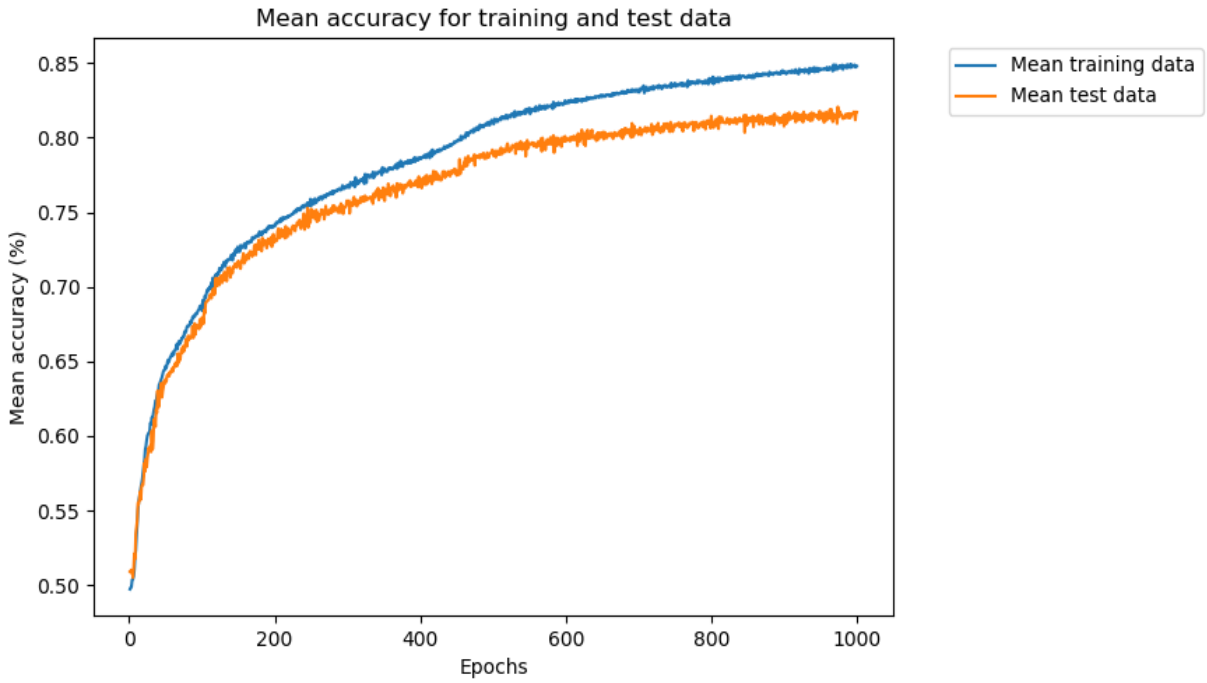


Figure 56. Mean training and validation accuracies for all five cross-validation folds of the 70 x 70 data set single-tracked CNN-architecture.

Appendix 11. Plotted results for the CNN-architecture for the 128 x 128 data set

From the accuracy plots (see Figure 57), we can see that training accuracy increased for all 2 000 epochs. The validation accuracy levelled out at approximately 90-91 % on epoch 1 000. We can also see that the inter-epoch accuracies varied. Training accuracies varied approximately 1 percentage point between epochs. As for the validation accuracies, the inter-epoch accuracy variations were approximately 2-3 percentage points.

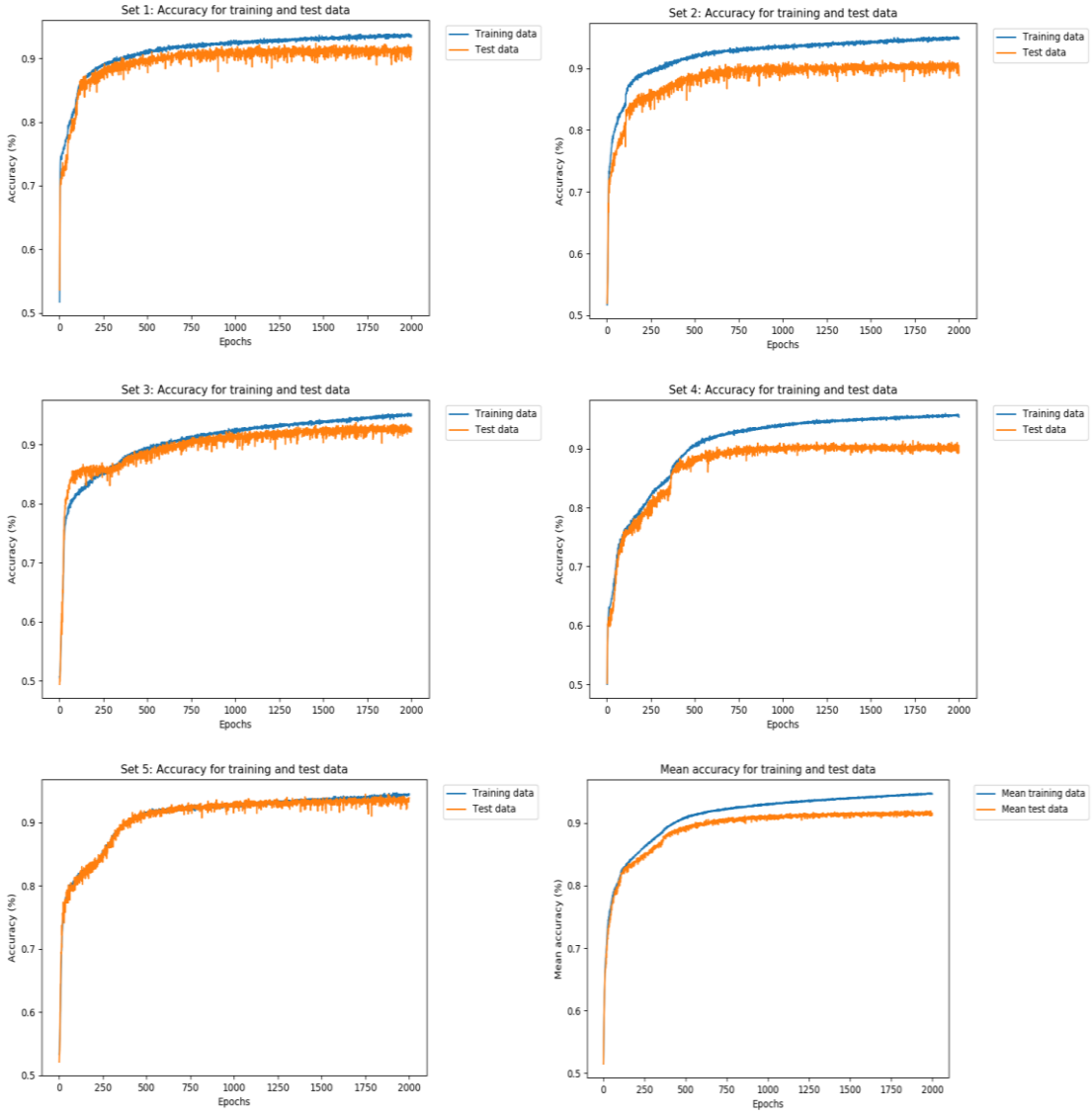


Figure 57. 128 x 128 data set CNN-architecture 5-fold cross-validation fold training and validation accuracies. (Upper left) First set, (Upper right) Second set, (Middle left) Third set, (Middle right) Fourth set, (Lower left) Fifth set, (Lower right) and Mean training and validation accuracies for all five cross-validation folds of the 128 x 128 data set CNN-architecture.

Comparing the accuracy plots for the 70 x 70 and 128 x 128 CNN-architectures we can see that they behaved similarly. However, the 128 x 128 architecture learned faster than its 70 x 70 architecture counterpart. The 128 x 128 CNN-architecture also achieved higher mean validation accuracy than the 70 x 70 architecture (91.4 % vs. 89 %). As the 70 x 70 and 128 x 128 CNN-architectures were identical, differences in mean validation accuracy were caused by differences in size and information content in the 70 x 70 and the 128 x 128 data sets.

Appendix 12. Plotted results for the CNN-architecture with an Inception-v4-module for the 70 x 70 data set

From the accuracy plots (see Figure 58), we can see that training accuracy increased over all 6 000 epochs, except for folds 1 and 5, for which training accuracy had levelled out at epoch 6 000. As for the validation accuracy, accuracy levelled out at epoch 2 000. Training accuracies varied approximately 1 percentage point between epochs, while validation accuracies varied approximately 2-10 percentage points between epochs.

Even though the 70 x 70 CNN-Inception-v4 architecture had a higher mean cross-validation accuracy than the 70 x 70 CNN-architecture (89.9 vs. 89 %), the difference in mean cross-validation accuracies appeared to be lower than the variation in the 70 x 70 Inception-v4 CNN-architecture mean training and validation accuracy plot. Thus, the 70 x 70 CNN-Inception-v4 accuracy results and plots suggested that adding only the Inception-C-module of the Inception-v4 architecture to a CNN-architecture does not improve the classification capability of a CNN-architecture.

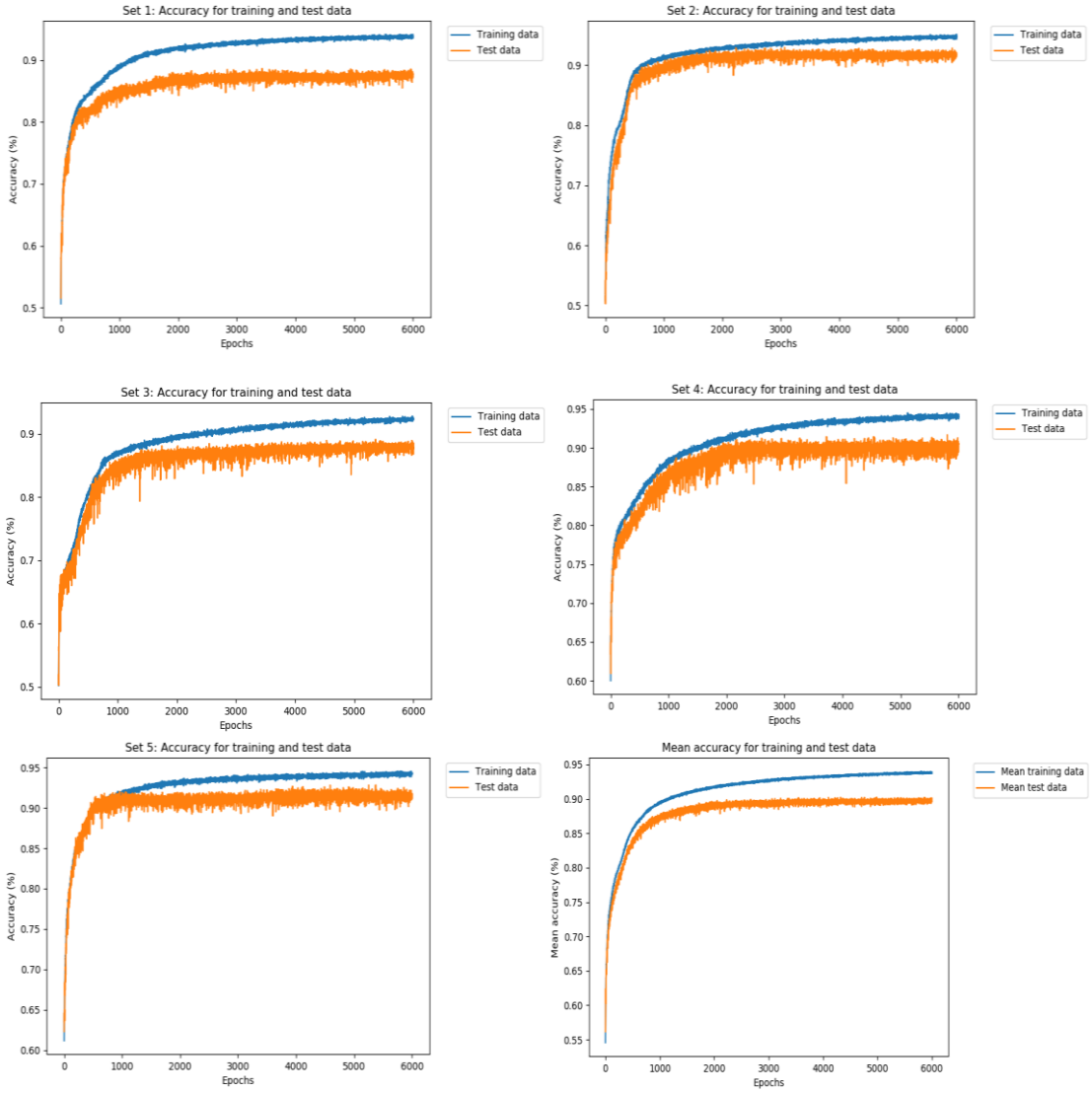


Figure 58. 70 x 70 data set CNN-Inception-v4-architecture 5-fold cross-validation fold training and validation accuracies. (Upper left) First set, (Upper right) Second set, (Middle left) Third set, (Middle right) Fourth set, (Lower left) Fifth set, (Lower right) and Mean training and validation accuracies for all five cross-validation folds of the 70 x 70 data set CNN-Inception-v4-architecture.

Appendix 13. Plotted results for the CNN-architecture with an Inception-v4-module for the 128 x 128 data set

From the accuracy plots (see Figure 59), we can see that training accuracy increased for all 2 000 epochs. As for the validation accuracy, accuracy levelled out within the 2 000 epochs, but variations were noted with respect to when accuracy levelled out (between epoch 250 and epoch 2 000). Training accuracies varied approximately 1 percentage point between epochs, while validation accuracies varied approximately 2-5 percentage points between epochs. As with the 70 x 70 CNN-Inception-v4 architecture, the differences in mean cross-validation accuracies between the 128 x 128 CNN-Inception-v4 architecture and the 128 x 128 CNN-architectures (92 vs. 91.4 %) appeared to be lower than the 128 x 128 CNN-Inception-v4 architecture inter-epoch variation. Thus, the results for the 128 x 128 CNN-Inception-v4 architecture also suggested that adding only the Inception-C-module of the Inception-v4 architecture to a CNN-architecture does not improve the classification capability of a CNN-architecture.

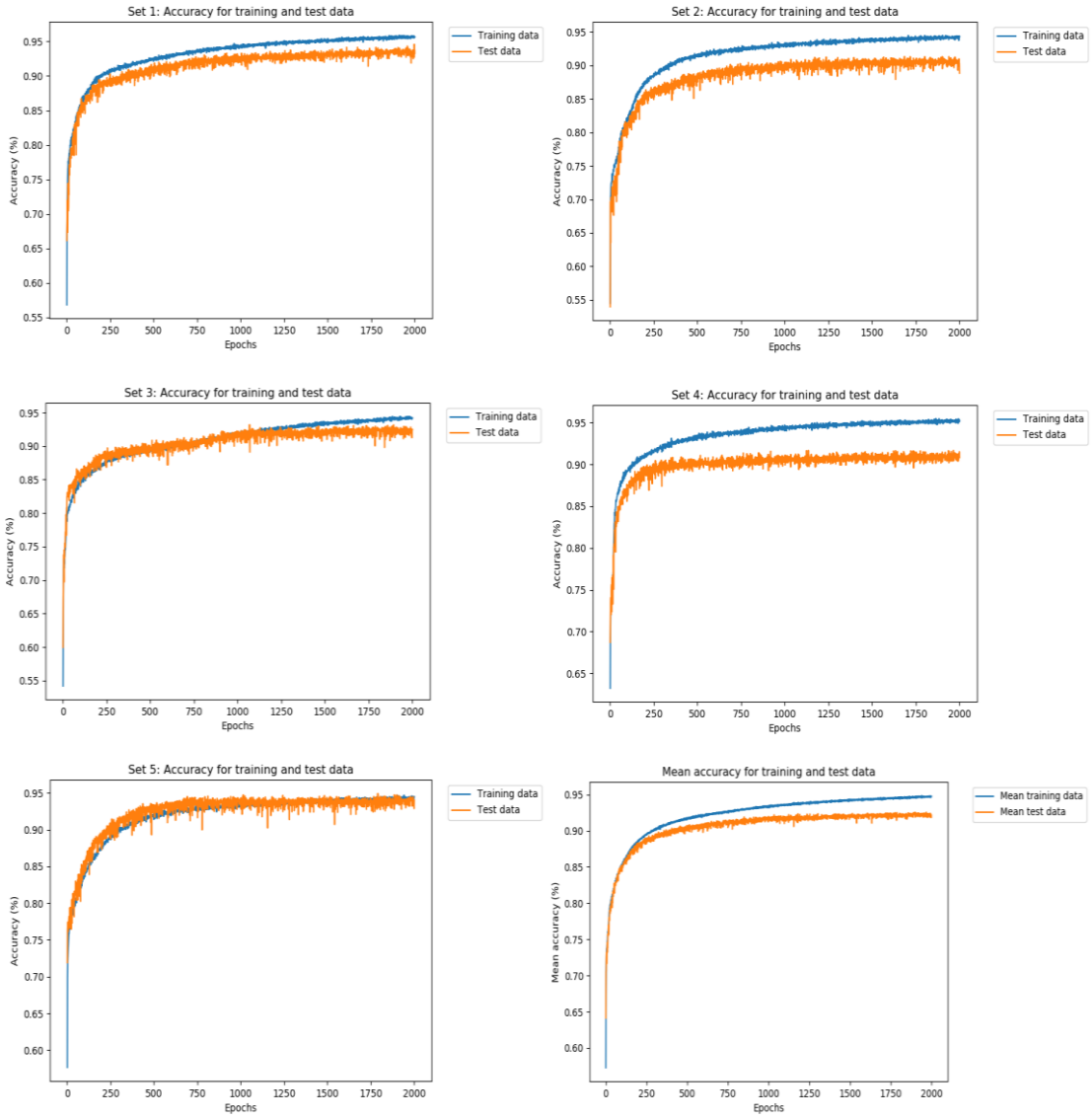


Figure 59. 128 x 128 data set CNN-Inception-v4-architecture 5-fold cross-validation fold training and validation accuracies. (Upper left) First set, (Upper right) Second set, (Middle left) Third set, (Middle right) Fourth set, (Lower left) Fifth set, (Lower right) and Mean training and validation accuracies for all five cross-validation folds of the 128 x 128 data set CNN-Inception-v4-architecture.

Comparing the accuracy plots for the 70 x 70 and 128 x 128 CNN-Inception-v4 -architectures we can see that they behaved similarly. Three differences set the 128 x 128 architecture apart from the 70 x 70 architecture. These differences were lower inter-epoch variation (about half of the 70 x 70 architecture), faster learning, and higher mean cross-validation accuracy. Besides the slightly different architectures, the slightly higher mean cross-validation accuracy for the 128 x 128 CNN-Inception-v4 architecture compared with the 70 x 70 CNN-Inception-v4 architecture (92 % vs. 89.9 %) could also be explained by differences in the two data set's constituent images.

Appendix 14. Plotted results for the ResNet-architecture for the 70 x 70 data set

From the accuracy plots (see Figure 60), we can see that training accuracy reached above 90 % before 20 epochs, and reached almost 100 % before 50 epochs, with the validation accuracy having exceeded 90 % before 50 epochs. Once these levels had been reached, neither training nor validation accuracies would increase anymore. We can also see that the training and validation plots sometimes experienced downward spikes before returning to the constant level. These downward spikes were fairly small for the training accuracies, at most decreasing five percentage points. The downward spikes for the validation accuracies, however, momentarily dramatically decreased, with decreases at most reaching approximately 40 percentage points.

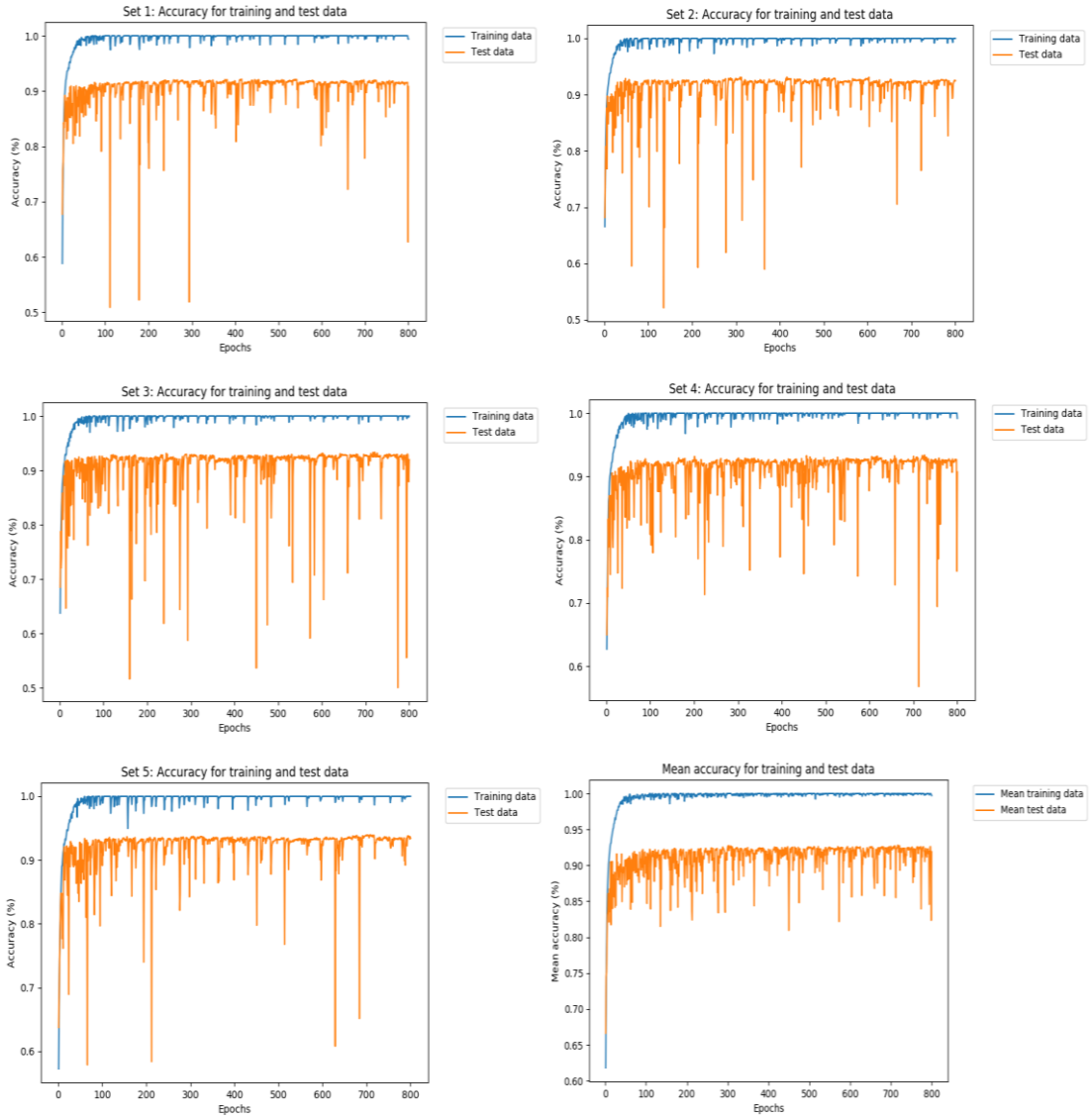


Figure 60. 70 x 70 data set ResNet-architecture 5-fold cross-validation fold training and validation accuracies. (Upper left) First set, (Upper right) Second set, (Middle left) Third set, (Middle right) Fourth set, (Lower left) Fifth set, (Lower right) and Mean training and validation accuracies for all five cross-validation folds of the 70 x 70 data set ResNet-architecture.

The reason why the ResNet-architecture was able to achieve a high mean validation accuracy might have been due to the number of filters on each layer in the respective sections. As we can see from the accuracy plots, accuracy reached the highest levels quite fast. However, accuracy also momentary decreased, with the largest decreases being around 40 percentage points. The layer filter counts might have caused the architecture to quickly learn some image patterns, resulting in high accuracy readings. For other image crop combinations, the different filters might have focused on many varying image aspects, leading to momentary drops in accuracy. Heavy use of network optimization might, thus, have allowed the ResNet-architecture to learn fast, but also made the architecture very bad at classifying some image crops.

Appendix 15. Plotted results for the ResNet-architecture for the 128 x 128 data set

From the accuracy plots (see Figure 61), we can see that the training accuracy reached above 90 % before 20 epochs, and reached 100 % after about 75 epochs, with the validation accuracy having exceeded 90 % before 20 epochs. Once the highest accuracies had been reached, accuracies mostly stayed at these levels with temporary drops in accuracy. Training accuracy drops were fairly small, at most decreasing five percentage points. The validation downward spikes could, however, momentarily drop as much as approximately 46 percentage points.

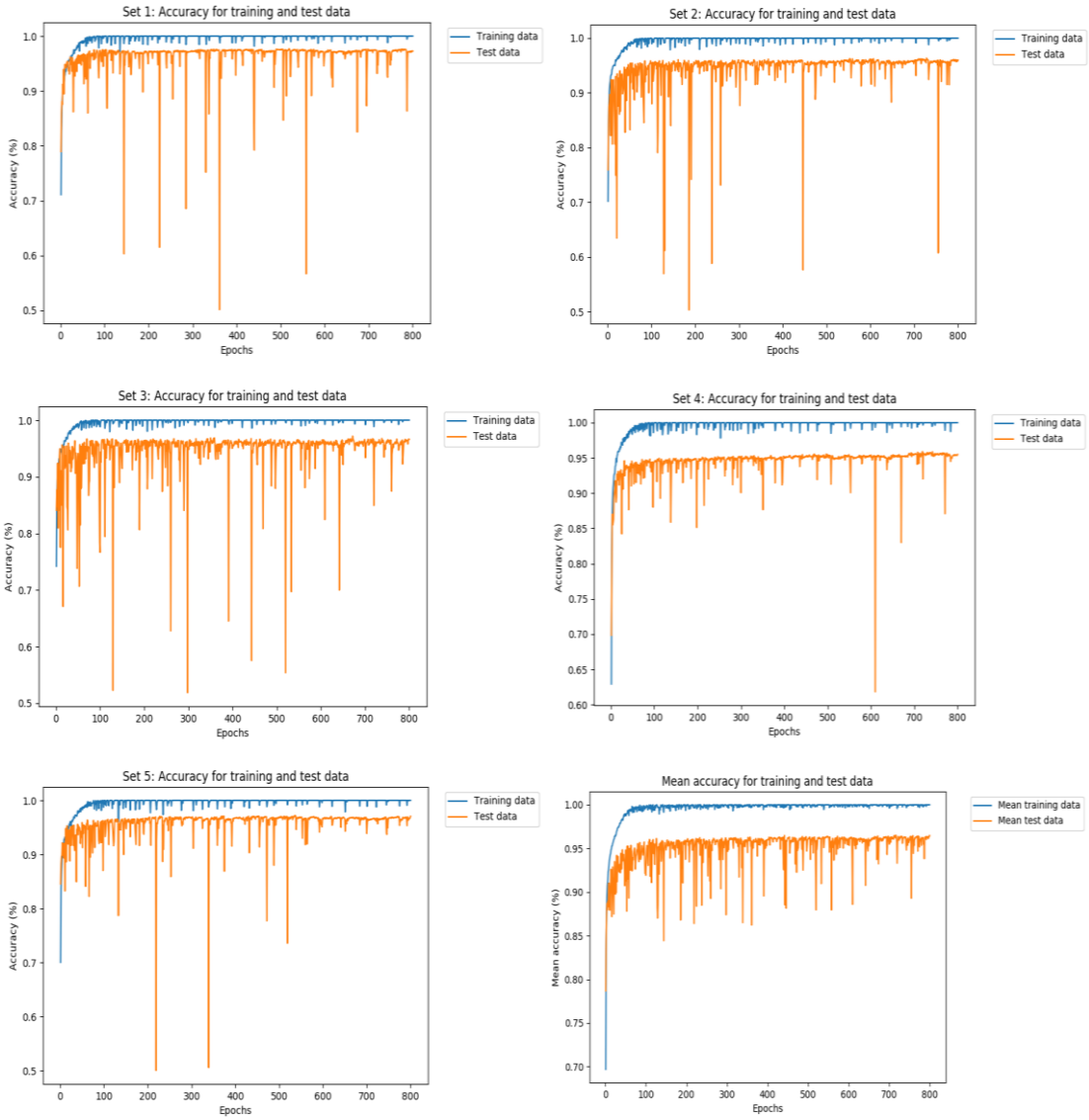


Figure 61. 128 x 128 data set ResNet-architecture 5-fold cross-validation fold training and validation accuracies. (Upper left) First set, (Upper right) Second set, (Middle left) Third set, (Middle right) Fourth set, (Lower left) Fifth set, (Lower right) and Mean training and validation accuracies for all five cross-validation folds of the 128 x 128 data set ResNet-architecture.

Comparing the accuracy plots for the 70 x 70 and 128 x 128 ResNet-architectures we can see that they changed in a similar manner. The two architectures differed in two ways, the 128 x 128 architecture could reach higher validation accuracies and also learn faster than its 70 x 70 architecture counterpart. As with the 70 x 70 ResNet-architecture, heavy optimization might have led the 128 x 128 ResNet-architecture to learn fast. In the same way as the 70 x 70 ResNet-architecture had momentary drops in accuracy, the 128 x 128 ResNet-architecture might also have had problems classifying some image combinations, leading to similar momentary drops in accuracy, due to focusing on certain patterns. The 70 x 70 and 128 x 128 data sets differed with regard to sampled tissue images and cropped parts of the images. Thus, it was not possible to explain the 128 x 128 ResNet-architecture's higher validation accuracy only based on the higher amount of information captured in each 128 x 128 image crop. Images crops created for the 128 x 128 data sets, might also have contained distinct cell patterns more easily classifiable than the image crops for the 70 x 70 data set.

Appendix 16. Plotted results for the CNN-architecture with an Inception-ResNet-v1-module for the 70 x 70 data set

From the accuracy plots (see Figure 62), we can see that the training accuracy levelled out after 4 000 epochs. As for the validation accuracy, accuracy peaked at approximately epoch 750, decreasing until levelling out at approximately epoch 5 000. Training accuracies varied approximately 1-3 percentage points between epochs, while validation accuracies varied approximately 2-8 percentage points between epochs.

After 6 000 epochs, the 70 x 70 CNN-Inception-ResNet-v1-architecture had a much lower mean cross-validation accuracy than the 70 x 70 CNN-architecture (84.2 % vs. 89 %). Thus, adding the Inception-ResNet-v1-module to the end of the 70 x 70 CNN-architecture appeared to worsen the validation accuracy of the base CNN-architecture. Looking at the accuracy plots for the Inception-ResNet-v1-modified 70 x 70 CNN-architecture, the highest validation accuracy was approximately 90 % at epoch 750 for fold 1. However, the lower peak validation accuracies for the other fold, lead the mean cross-validation accuracy to peak at approximately 86 %. The validation accuracy peak at epoch 750 and the subsequent decrease in validation accuracy suggested that the architecture for the 70 x 70 CNN-Inception-ResNet-v1-architecture was poorly optimized.

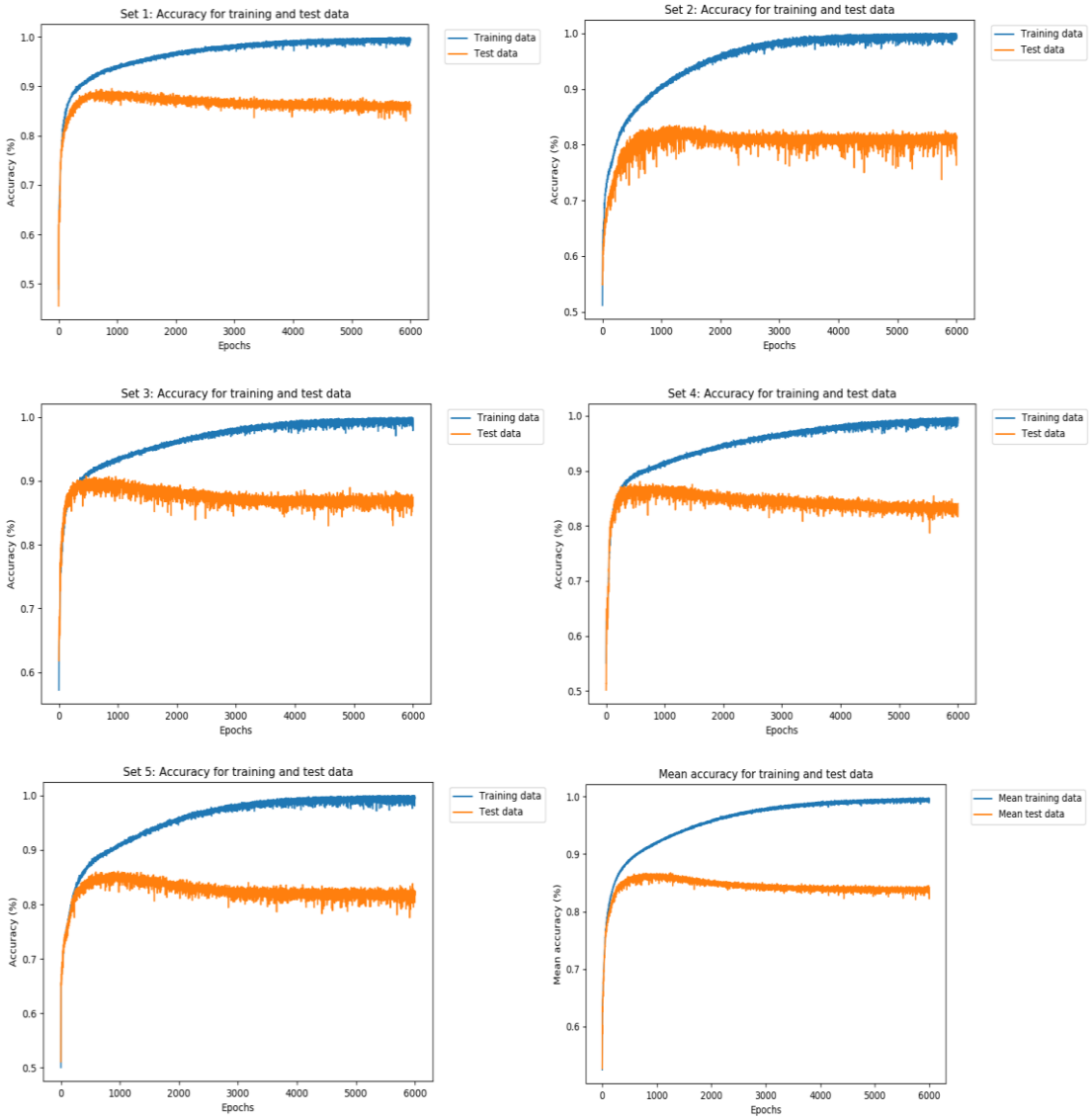


Figure 62. 70 x 70 data set CNN-Inception-ResNet-v1-architecture 5-fold cross-validation fold training and validation accuracies. (Upper left) First set, (Upper right) Second set, (Middle left) Third set, (Middle right) Fourth set, (Lower left) Fifth set, (Lower right) and Mean training and validation accuracies for all five cross-validation folds of the 70 x 70 data set CNN-Inception-ResNet-v1-architecture.

Appendix 17. Plotted results for the CNN-architecture with an Inception-ResNet-v1-module for the 128 x 128 data set

From the accuracy plots (see Figure 63), we can see that the training accuracy increased over all 2 000 epochs. The validation accuracy, on the other hand, levelled out between epoch 500 and 1 500 for all iterations. Training accuracies varied approximately 1 percentage point between epochs, while validation accuracies varied approximately 1-10 percentage points between epochs. After 2 000 epochs, the 128 x 128 CNN-Inception-ResNet-v1-architecture had a slightly higher mean cross-validation accuracy than the 128 x 128 base CNN-architecture 93.3 % vs. 91.4 %). However, the mean cross-validation accuracy differences between these two architectures was lower than the inter-epoch validation accuracy differences for the 128 x 128 CNN-Inception-ResNet-v1-architecture. Adding a CNN-Inception-ResNet-v1-module to the end of a base 128 x 128 CNN-architecture did, thus, not appear to increase the validation accuracy.

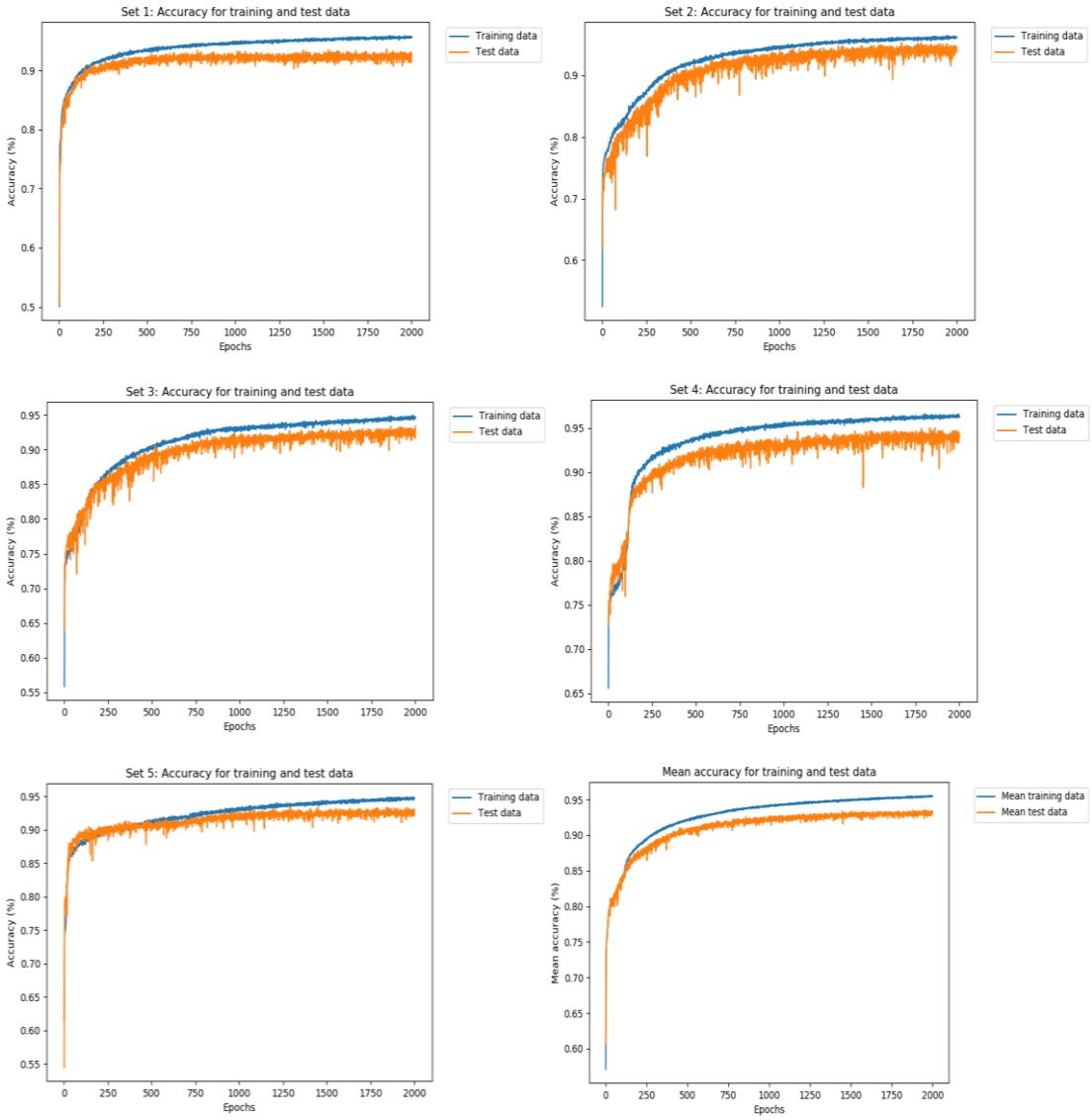


Figure 63. 128 x 128 data set CNN-Inception-ResNet-v1-architecture 5-fold cross-validation fold training and validation accuracies. (Upper left) First set, (Upper right) Second set, (Middle left) Third set, (Middle right) Fourth set, (Lower left) Fifth set, (Lower right) and Mean training and validation accuracies for all five cross-validation folds of the 128 x 128 data set CNN-Inception-ResNet-v1-architecture.

Comparing the accuracy plots for the 70 x 70 and the 128 x 128 CNN-Inception-ResNet-v1-architectures we can see that they changed in dissimilar manners. While the 70 x 70 architecture validation plots peaked before epoch 1 000, only to get worse results, the 128 x 128 architecture validation plots improved, levelling out between epochs 500 and 1 500. The 70 x 70 architecture learned faster than its 128 x 128 counterpart, although the 128 x 128 architecture still learned quite fast, yielding validation accuracies exceeding 90 % before epoch 500. However, the 128 x 128 architecture could reach higher validation accuracies than the 70 x 70 architecture, and did not get worse with time.

Heavy hyper parameter optimization might have been a reason behind the fast learning of both CNN-Inception-ResNet-v1-architectures. Even though the 70 x 70 and 128 x 128 data sets differed with regard to the constituent tissue image samples, the differences in validation accuracy plots signaled that the 128 x 128 architecture had a better architecture design, not suffering from design flaws like its 70 x 70 counterpart. However, images in the 128 x 128 data set might also have contained more distinct cell patterns, besides containing more information. Data set differences might, thus, have given the 128 x 128 architecture a potential for higher validation accuracies than the 70 x 70 architecture, which potential was realized by an architecture better adapted to the dimensions of the input images.

Appendix 18. Comparison of the architecture results for the original and corrected data sets

Comparing the mean cross-validation accuracies for the original and corrected data sets (see Tables 2-3), we can see that the accuracy results for the corresponding architectures were within 2 percentage points of each other. Both the 70 x 70 and the 128 x 128 validation accuracies for the ResNet-architecture were slightly higher for the original data sets than the validation accuracies for the corrected data sets. However, the validation accuracy results for the 128 x 128, CNN, Inception-v4-modified CNN, and the Inception-ResNet-v1-modified CNN, were higher for the corrected data sets than the corresponding results for the original data sets.

Comparing the plots for the architectures using the original data sets (see Figure 64) with those for the architectures using the corrected data sets, we can see that the plots resembled each other with respect to both how accuracy changed and the final results. One difference between the plots for the original and the corrected data sets were the lower inter-epoch cross-validation accuracy differences for the original data sets, particularly noticeable for the ResNet-architectures. We can also see that the plots for the architectures using the original data sets had levelled out for the ResNet-architectures but were still growing for the non-ResNet-architectures. Thus, the higher validation accuracies for the corrected data sets could be explained by the fact that the training-test runs with the corrected data sets were performed until the validation accuracies levelled out, while the training-test runs with the original data sets were not run until validation accuracies could peak.

Even though the architectures constructed with the faulty data sets might have had inflated validation accuracies, indicated by the results for the ResNet-architectures and the lower inter-epoch cross-validation accuracy differences for the faulty data sets, the test results and plots of the two sets of architectures were very similar overall. Thus, we can see that reusing the hyper parameter settings and layer filter counts determined for the old architectures in the new architectures worked well. The reason why the architectures for the original and corrected data sets produced similar results might have been that the image crops selected for the original data sets were partly the same as those selected for the corrected data sets. Thus, the architectures sets might have been tested on partly the same images. Another reason might be due to low intra-tissue image variations. If the tissue images sampled from varied little with respect to each other, it would not matter much which image crops were selected as test images would all be similar to each other.

Table 2. 5-fold mean cross-validation accuracies for architectures with original data sets.

70 x 70 ResNet	93,3%
128 x 128 CNN	90,4%
128 x 128 CNN with Inception-v4-module	90,4%
128 x 128 ResNet	97,6%
128 x 128 CNN with Inception-ResNet-v1-module	91,4%

Table 3. 5-fold mean cross-validation accuracies for architectures with corrected data sets.

70 x 70 CNN	89%
70 x 70 CNN with Inception-v4-module	89,9%
70 x 70 ResNet	91,9%
70 x 70 CNN with Inception-ResNet-v1-module	84,2%
128 x 128 CNN	91,4%
128 x 128 CNN with Inception-v4-module	92%
128 x 128 ResNet	96,5%
128 x 128 CNN with Inception-ResNet-v1-module	93,3%

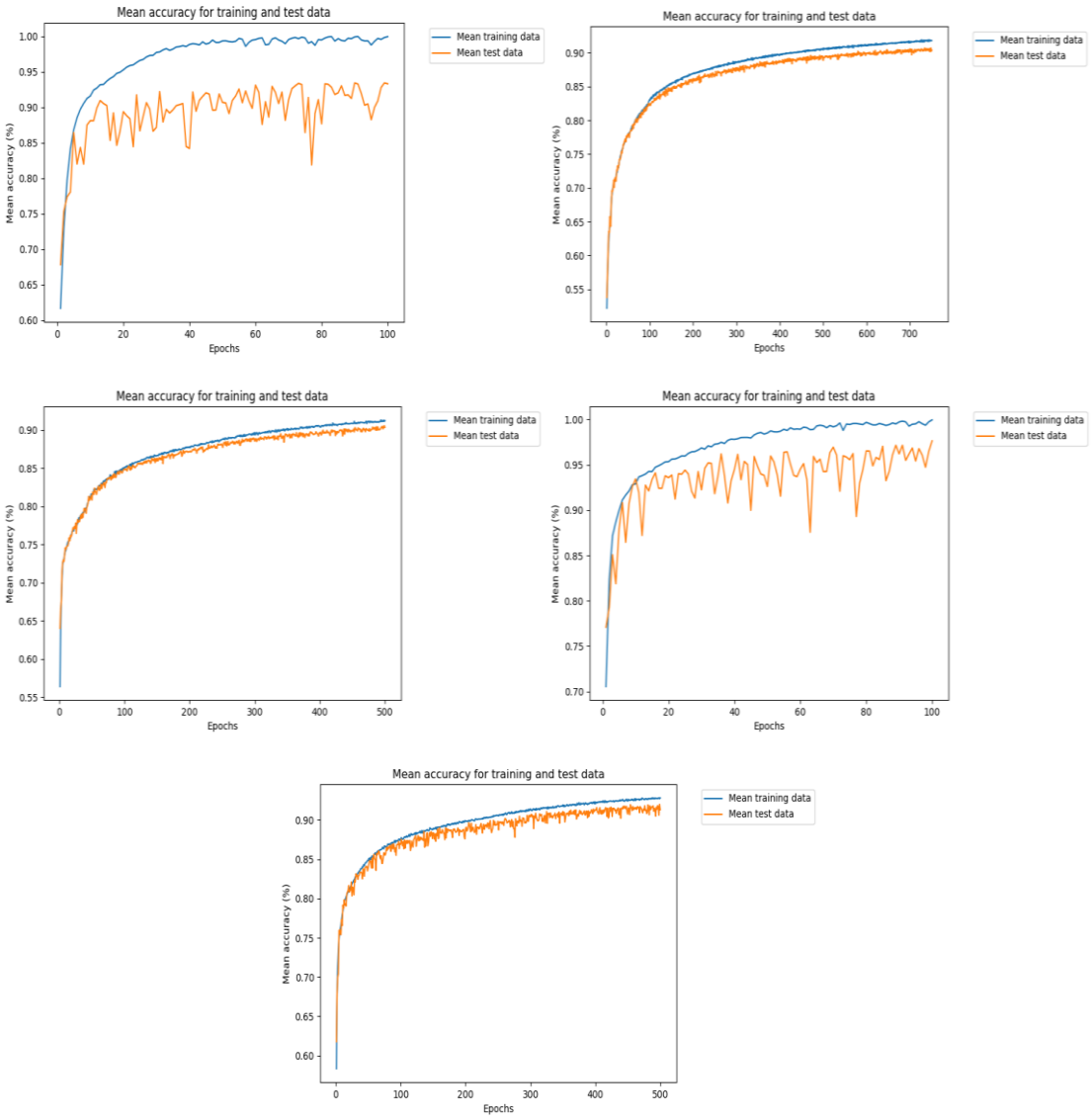


Figure 64. Mean training and validation accuracies for all five cross-validation folds of the original data sets. (Upper left) 70 x 70 data set ResNet-architecture, (Upper right) 128 x 128 data set CNN-architecture, (Middle left) 128 x 128 data set CNN-architecture with Inception-v4-module, (Middle right) 128 x 128 data set ResNet-architecture, (Bottom) and 128 x 128 data set CNN-architecture with Inception-ResNet-module.