Automated Histopathological Evaluation of Tumor Images using CNNs

Master's Thesis in Biomedical Engineering

Jonatan Nyström

September 2020

Lund University - Faculty of Engineering
Department of Biomedical Engineering

Supervisor: Jonatan Eriksson

Examiner: Christian Antfolk

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Histopathology is the procedure used in medicine to optically examine microscope-images of tissue samples (biopsies) in order to study the manifestation of disease. It is used, for example, to diagnose the spread and type of cancer tumors, which have implications on the chosen treatment. Currently this type of analysis is done manually by trained professionals. It is time consuming and the diagnostic agreement between professionals varies. Due to this, there is a potential use for an automation of the process, using for example neural networks.

Convolutional Neural Networks (CNNs) has become increasingly popular for image analysis within the medical field. They have proven them self to be among the best techniques. CNNs has for example been successfully used to classify different types of lung cancer tissue in microscope-images.

This thesis evaluates three different CNN architectures (InceptionV3, VGG16 and compactVGG) on classification of tiles, from medical whole-slides of sliced tumor biopsies. The biopsies are from lymph node metastases, from patients with malignant melanoma (i.e. skin cancer). The data consists of 19 whole-slides, where four different tissue components, to be classified by the models, has been manually annotated by a pathologist. Furthermore, the thesis examines the effect of the chosen size of the image/tile being classified (*magnification* or tile size).

Summary of results: It is concluded that InceptionV3 for a tile size of 224 give the best results. With a prediction accuracy of 89.6%, 90.0%, 97.5% (the last result did not include ambiguous tiles) on 3 different test data sets. Its performance is very similar to VGG16.

List of Abbreviations

- **AUC**, Area Under the (ROC) Curve
- **CNN**, Convolutional Neural Network
- **H&E**, Hematoxylin and Eosin
- **ML**, Machine Learning
- **NN**, Neural Network
- **ROC**, Receiver Operating Characteristic
- **SGD**, Stochastic Gradient Descent

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Background

Histopathology is the procedure used in medicine to optically examine microscope-images of tissue samples (biopsies) in order to study the manifestation of disease. The biopsies (a removed tumor for example) are sliced into thin slices of tissue that is then analyzed using a microscope. The tissue is commonly stained with two coloring agents called hematoxylin and eosin (H&E). The coloring works as a visual aid for the pathologist analyzing the images. Histopathology is used, for example, to diagnose cancer patients. Potential questions are: how has the immune system responded, how has tumor responded to treatment and was the removed tumor even malignant in the first place? It is also used to determine the type of cancer, which have implications on the chosen treatment [7]. Currently this kind of analysis is done manually by trained professionals. It is time consuming and the diagnostic agreement between professionals depends on various factors. For example the skill of the examiner, the tumour differentiation (how much cancer cells differ from normal) and slide quality [8]. An automation of this process could prove to be beneficial for improved accuracy and to extract information that previously required additional and/or more costly methods. But also to reduce the workload of the human professionals and speed up the diagnostic process. For example, the tumor percentage area of the samples is one important factor of interest [9],[10]. Another factor of interest is the likelihood that the patient will develop a new tumor with in a given time period. Ideally one would be able to diagnose patients faster and give more specific diagnoses and predictions, enabling better treatment.

This thesis data consists of H&E images from biopsies of lymph node metastases, from patients with malignant melanoma (skin cancer). A metastatic tumor is cancer that has spread from other parts of the body [11]. In these cases the patients had skin cancer which then spread to the lymph nodes.

The application of Convolutional Neural Networks (CNNs), for different types of image and data processing, has been increasing rapidly the last couple of years, including in the field of medicine and more specifically in histopathology [8], [12]. A CNN is a statistical model that can be used to make predictions on for example images. A typical example is to predict whether a image contains a cat or a dog. Another example of the use of CNN models is to detect and classify sections of different types of cells and tissues in H&E images.

The use of CNNs for computer vision had a breakthrough at the ImageNet Large Scale Visual Recognition Contest (ILSVRC) 2012 [13], which is one of the major benchmark contests for computer vision tasks [14]. CNNs has since been applied for the particular use on cellular/medical image analysis. Moen et al. 2019 [13] summarizes the basic concepts of deep learning and its application in the area of cell images. They also give examples of architectures, such as *Mask R-CNN* [15], which has proven successful on cellular data.

1.2 Previous work on medical data

There are several fields within medical image analysis besides histopathology, where CNNs of various types can be used. In this section works on two other fields are first presented and then works done on histopathological images are presented. The purpose is to give an overview of the field, as well as to introduce a number of possible CNNs that could be used on medical data.

Non-Histopathological Data

A recent study on mammography(X-ray) images, by McKinney et al. 2020 [16], showed that an automatic approach based on Neural Networks (NN) improved the diagnostic agreement. The authors uses a CNN type of architecture as a second opinion, together with an initial opinion of a trained expert to diagnose breast cancer from mammography images. This enabled faster, more accurate diagnoses of early stage breast cancer, especially reducing false negatives.

Esteva et al. 2017 [17] uses InceptionV3 (pre-trained on the ImageNet data set) to classify skin cancer from images of skin lesions (moles etc.), achieving a performance equivalent of trained dermatologists. Guo and Yang 2018 [18] instead uses multiple ResNets on similar data. ResNet [19] is CNN-type of architecture that tries to reduce the issue of *vanishing gradient* when training networks and thus enables the use of even deeper networks.

Histopathological Data

Yu et al. 2016 [8] used the open-source software *CellProfiler* [20] to segment and extract features from H&E tumor images (mainly from The Cancer Genome Atlas (TCGA) [21]). The features are then used to classify the two major different lung cancer types, and to make survival prognoses on patients/cases using several different variations of Machine Learning (ML) methods, such as *Support Vector Machines* (SVM) and *Random Forest*. The two major types of lungcancer are Lung Adenocarcinoma (LUAD) and Lung Squamous Cell Carcinoma (LUSC). Coudray et al. 2018 [12] improves the classification results of Yu et al. by using the CNN-type architecture *Inception V3* [22]. It also extends the application to identifying mutations that previously required an additional method of immuno-histological staining.

Hong et al. 2020 [23] predicts endometrial cancer subtypes from H&E images, using a customized deep network called *Panoptes* and case/patient level, labelled data. The network takes advantage of the multi-resolution structure that H&E images has. I.e. there are several images of various resolution and size depicting the same sample.

Mahmood et al. 2019 [24] uses a deep General Adversarial Network (GAN) for nuclei segmentation on cells in H&E images, sampled from several different organs.

U-net [25] is another popular network, which has been previously used for cell-counting, cell-detection and cell-morphometry. It extracts features from several levels of resolution which is then used together to make the final classification prediction on a pixel-level basis (*semantic segmentation*).

Xception is a newer architecture, that has proven to beat previous ones at classification on the ImageNet data set, using depth-wise separable convolutions [26]. Kassani et al. 2019 [27] found that Xception performed the best, compared to several other models, on histopathological images of breast tumors.

Sun et al. 2019 [28] does a comparative study between a CNN and a Fully Convolutional Network (FCN) on histopathological whole-slide images. They found that they perform similar on high resolution data. A FCN uses $1x1$ convolution layers instead of a neural network, at the end.

1.3 Goals and Structure of the Thesis

Ideally, one would be able to quickly and cheaply detect cancer cells and classify their sub-types accurately, using an automated process. Furthermore, calculating other qualities such as the tumor percentage area as an indicator of the spread, would be valuable as well. An automatic classifier has the potential to reduce the need of additional costly/time-consuming methods for detection. This could enable fast, more consistent and potentially more specific and accurate diagnoses as well as treatments (different treatments may suit different sub-types for example). It could also enable predicting the spread of the cancer and/or urgency of the patient returning for additional check-ups. Even when the automatic classifier determines tumor regions less accurately than an expert pathologist it could still be of use as long as it is accurate enough, such that it gives a roughly correct estimate about the tumor region sizes.

This thesis will compare different CNN-type architectures for identifying sections of four different tissue types in microscopic H&E tumor biopsy images. The images are tiled into smaller sub-images (tiles), on which the classification is performed and the impact of the tiling-sizes (magnification level) on prediction performance is examined. The thesis focuses on determining the best models/pre-processing for the application of automatic histopathology. Using the prediction accuracy (the share of correctly classified samples) and Area Under the Curve (AUC, a measure on how well the model differentiates between classes) as performance metrics.

An initial test is done using four different models on an initial data set (with fewer samples) using one tile size of 224×224 pixels. After that, 3 types of models are chosen and each trained for 3 different tile sizes on a larger, extended data set with randomized, non-ambiguous tiles from 17 whole-slides. These models are first evaluated on the extended data set and then on all tiles from two new whole-slides.

Chapter 2
Theory and Models

In this chapter theoretical concepts used in the thesis are defined and explained.

This thesis uses what is commonly referred to as statistical models which tries to capture some relationships/features residing in data. The most common model in this context is linear regression. The solution to a linear regression problem, fitting coefficients to datapoints, is most often in a closed solution form.
Machine Learning (ML) however is an umbrella term for how more complex statistical models find their parameter values using numerical optimization methods. There are two fundamental types of optimizations or *learning* in this context, supervised and unsupervised. Supervised learning simplified means that there is an answer, or *ground truth* for what the model should output, given a certain input. This means that each input data sample has a corresponding sample output.
Unsupervised learning however, do not require a ground truth. Instead the idea is instead that there is a "hidden" set of features residing inside the data. The goal is then to extract features in such a way that similar input ends up having feature "close" to each other in the feature space. A typical example of an unsupervised model is *Variational Auto Encoders* (VAE) [29].

During the model optimization or *training*, the data is often split up into to three sets, Training, Validation and Test. The training set is used to find the best model parameters, the validation is used for hyper-parameter optimization and for choosing a final model. Finally, the test set is used to evaluate the performance of the chosen, trained model. It is important to note that the test set consists of data not previously seen by the model.

2.1 Models used for classification

2.1.1 Linear Regression

Linear Regression is one of the most basic models used, when doing statistical analysis [30]. This thesis uses a slightly modified version for classification. Multinomial logistic regression (regression for discrete classification of several classes) can be expressed similarly as with regular regression [30], see Equation 2.1

$$y_n = \boldsymbol{\beta} \cdot x_n. \tag{2.1}$$

The difference is that the $y_n(c)$ represents the *log-odds* of sample n belonging to class c.

More formally, the probability of a sample n belonging to a class c, can be expressed as in Equation 2.2,

$$Pr(d_{n,c}) = Pr(d_n = c|x_n; \boldsymbol{\beta}) = \frac{exp(\beta_{0,c} + \beta_c^T \cdot x_n)}{\sum_{j \in C} exp(\beta_{0,j} + \beta_j^T \cdot x_n)}, \tag{2.2}$$

where d_n is the true label/class and $\boldsymbol{\beta}$ is the correlation matrix whose elements are found during the optimization of the probabilities/log-odds of classifying correctly.

2.1.2 Neural Networks

A neural network [31], is a data driven model. The model parameters are fitted to data (commonly referred to as *training*), to capture statistical relations between the input-output variables of a system. The neural network model can, after it has been fitted or *trained* to some data set, then be used to make predictions about new data samples, where the real, true answer, is unknown (*inference*). This can be viewed in contrast to the more classical models within physics, which should at least theoretically, capture the fundamental, true dynamics, of a deterministic system. Hopefully capturing some "law of nature". Neural networks are constructed of nodes, links/*weights* between the nodes (the main parameters of the model) and non-linear *activation functions* (described further in section 2.2.2). In Figure (2.1) the structure of a "neuron" or node is presented.

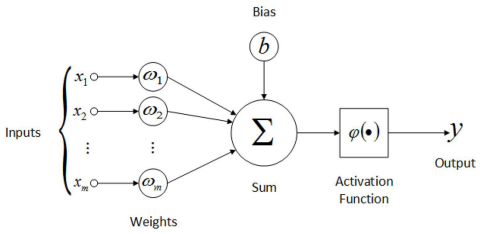


Figure 2.1: A neuron/node, source: [1]

Each x_i represents a signal/scalar-value which could be from, for example, either a node from a previous layer or a data-sample input pattern. Signals between layers are in this work referred to as *features*. The signals are weighted together, a bias b is added (which can also be seen as just a weighted constant signal of value 1) and the resulting scalar value is sent into a non-linear activation function $\varphi(\cdot)$. The resulting output y, is the output of the node.

$$y = \varphi(b + \sum_{i=0}^{m} x_i \cdot w_i) \tag{2.3}$$

Neurons and nodes will throughout this paper be used interchangeable. It is worth noting that when using a linear activation function, φ, this basic node becomes a regular linear regression model.

In Figure (2.2) the structure of a basic fully connected, feed-forward network is presented, also referred to as a *dense* network. Feed-forward in this case just refers to the fact that we only have connections/weights between each consecutive layer. I.e. there are no connections that skips layers and there are no weights that goes from a node back into itself or other nodes in the same layer. Fully connected refers to the fact that there is a weight between any node in any layer and all other nodes in the layers before and after, note however that this weight could be 0. I.e. the "fully connectedness" only refers to between layers, not for all nodes in the network and it do not include self references. This is the most basic, "vanilla" kind of network. The concept of a network being *deep* means that it has a lot of hidden (i.e. middle) layers, usually more than two. Deeper networks has proven to perform well were more shallow ones have failed in a number of cases [32].

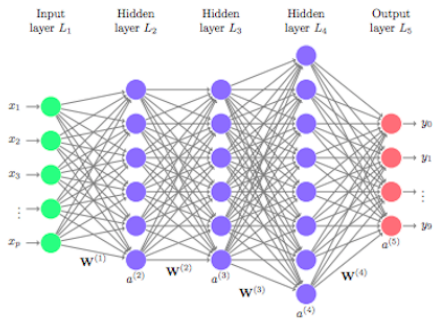


Figure 2.2: A deep neural network, source: [2]

There are many different types of Neural Networks. This thesis focuses on Convolutional Neural Networks (CNNs) [33] but many aspects brought up in this chapter holds true for other network architectures as well. Example of such architectures are Recurrent Neural Networks (RNNs) [34] which are used for time-series data. Auto-encoders, used for data-compression and to extract identifiable features even without *ground truth* data to train the model on. And General Adversarial Networks (GANs) [35] which pits two networks competing against each other. One generates data to try to "fool" the other, which tries to not be fooled.

2.1.3 Convolutional Neural Networks

CNNs [33],[36],[31], [37], have a similar construct to ordinary Neural Networks but are typically used for data with strong spatial correlations, such as images. Each layer is characterized by a set of *tensors* (a tensor is a general multi-dimensional version of matrices). The weights of the network are the elements of the tensors. The tensors are also referred to as *kernels*. In each layer the input goes through three stages for each kernel. *Convolution*, *Activation* and *Pooling*. Often the activation step is considered to be a part of the convolution step, as in Figure 2.3. Figure 2.3 presents an example of a CNN structure. It shows how a multimodal image (i.e. for example a RGB image. Mode in this case is the same as *channel*)

first goes through a convolution which results in a new "image" or *map*, that goes through the pooling step which results in a new map that is finally being vectorized. The vector is then used as input to a dense neural network. Note that Figure 2.3 uses the word layer instead of stage.

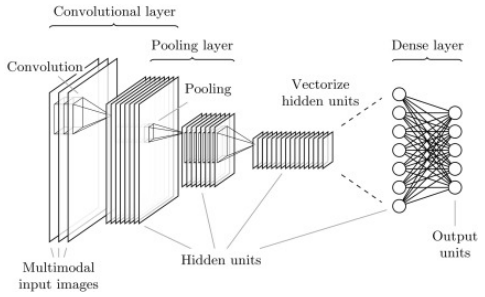


Figure 2.3: A CNN, source:[3]

In the convolution stage the kernel slides across the image multiplying and summing the values for each position, see Figure 2.4 (note that this works just as well with 3-dimensional inputs and kernels).

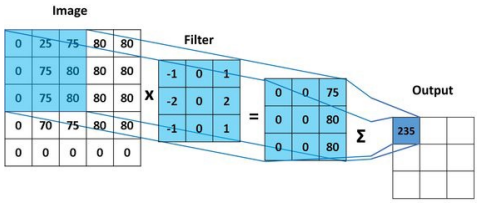


Figure 2.4: 2D-convolution, source: [4]

The step-size of the convolution is referred to as *stride*. Optionally one can also choose to add zero-padding to the images before the convolution.
In the activation stage the output of the convolution is then further processed element-wise through a non-linear activation function (further described in section 2.2.2). Finally, the results is filtered through the pooling stage. There are two types of pooling commonly used, *average-pooling* (equivalent to a convolution with a uniform kernel) and the non-linear *max-pooling*. Max-pooling also slides a window over the image, and for each position it outputs the maximum value of the elements within the window.

The entire process can be interpreted as extracting *features* from the input image. Where the resulting output is called a *featuremap*, i.e. where the features are positioned according to the position in the image from where they were extracted. Giving a "distorted" version of the original image/signal. Each kernel produces a featuremap as input to the following layer. The most basic implementations uses the same kernel dimensions and activation functions for a set of kernels in a convolution layer, but this is not necessary as can be seen with *InceptionV3*.

The general structure for all classifying CNNs used in this thesis is an initial Convolutional Network part, whos output is then used as input to a dense Neural Network. Which can be seen in the previous Figure 2.3. One can think of it as follows, the convolutional part extracts features from the images, such as color composition, shapes etcetera. Which the dense part then uses to classify the image.

VGG16

VGG16 [38] (the number 16 refers to the number of layers) is a CNN-model that has become influential due to its performance in "ILSVRC-2014" (ImageNet Large-Scale Visual Recognition Challenge), a competition for computer vision. It has, compared to, for example *InceptionV3*, a simplified structure, using only 3x3 convolution windows and ReLU activation functions. A implementation of the VGG16 architecture can be seen in Figure (2.5). Note that the feature-maps sizes in each layer is adaptable depending on the sizes of the kernels and stride, as well as the input size.

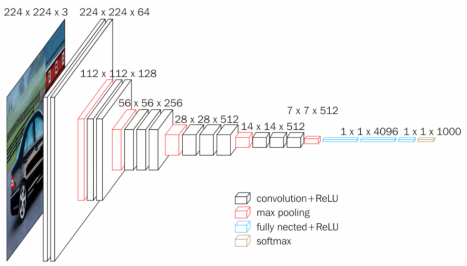


Figure 2.5: VGG16, source: [5]

InceptionV3

InceptionV3 [22] is one of the CNN-models used in this thesis. It is a continuation of the Inception architecture [39] which enabled deep and *wide* networks, with relatively low computational needs. The *width* refers to the number of nodes in each layer. It introduces a mini-network/module, see Figure 2.6, with a multi-level feature extraction using differently sized convolutional filters/windows. I.e. the idea is to extract features of different scale/magnitude in the same layer. As opposed to extracting large features in the initial layers and smaller in the deeper ones, which is perhaps a more direct intuitive approach.

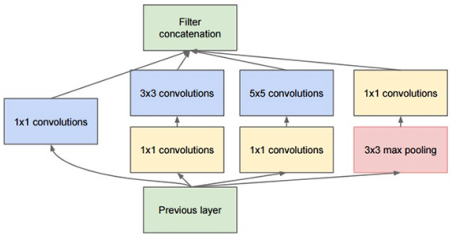


Figure 2.6: The inception module, source: [6]

2.2 Metrics and Functions

There is a need to evaluate the performance of models both during the *training* (fitting the model parameters to the data) and afterwards. Hence there is a need for good metrics. In this part the metrics used in the thesis is defined and explained, as well as important functions used for the models.

2.2.1 Loss function

The loss function is the metric that the model should try to optimize (the optimization is commonly referred to as *training*). It is a continuous function which measures the performance of the model. Typically this is based on how close (as measured by some norm) the network managed to get to the ground truth of the data. For a model doing classification, i.e. using a discrete ground truth (the class a sample truly belongs to), one can use the Categorical Cross Entropy, $E(\cdot)$, as loss function [30], [31], [40], [41], as defined in Equation (2.4).

$$E(w, X, d) = -\sum_{n=1}^{N_s}\sum_{i=1}^{N_C} d_{n,i} \cdot log(y_i(x_n)), \tag{2.4}$$

where, $d_{n,i} = 1$ if sample n belongs to class i, 0 else.
$y_i(\cdot)$ = the output of the network. When using softmax (described in section 2.2.2) at the model output, the output can be interpreted as the model assigning a "probability"/voting weight, of the sampe/input signal/pattern $(\cdot)$, to be class i. x_n =input sample n and X = all input samples.

One should note that there exist proposals of variations of this loss function for special cases. For example in the case of noisy labels as presented by Zhang and Sabuncu [42]. Where noisy labels can be understood as misclassifications in the ground truth.

2.2.2 Activation functions

Activation functions are the non-linear functions in the neurons. A intuitive analogy is where the function sets a threshold for whether the neuron should "fire" or not.

Softmax (Equation 2.5) is typically used at the output of a multi-class classifying network since it outputs a probability vector (positive elements that sums up to one) [31],

$$\varphi_c(a) = \frac{e^{a_c}}{\sum_{i \in C} e^{a_i}}, \tag{2.5}$$

where a is a vector of all the node outputs of the previous layer and C is the set of all classes.

Rectified Linear Unit (ReLU) (Equation 2.6),

$$\varphi(a) = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{else} \end{cases}, \tag{2.6}$$

where a is a scalar. There are many variations of the ReLU function, which all aims to alleviate the issue of vanishing gradient (as explained in section 2.3.4) [43]. The basic ReLU presented here however also introduces the "dying ReLU" problem [44] which also may end up preventing the training of the network. It describes when the network weights gets stuck in a domain where the ReLU only outputs zeros, resulting in a zero gradient and hence no update of the weights.

2.2.3 ROC and AUC

Receiving Operating Characteristic (ROC) is a performance measure for *binary* classifiers. It shows the sensitivity to a changed classifying threshold. If the scalar model output, y, is higher than the threshold value, the sample is deemed to belong to the certain class i.e. a "positive" sample. The ROC-curve plots the true positive rate (TPR or sensitivity) against the false positive rate (FPR), as the model threshold goes from 0 to 1. The true false rate (TFR) is called specificity. Hence the worst case is a straight-line, i.e. a 50/50 chance of true positive/false positive.

Area Under the Curve (AUC) is the area under the ROC-curve and can be interpreted as the classifier's ability to distinguish the different classes. Ideally, of course, the output should be 0 for all negative samples and 1 for all positive. These concepts can be extended to multi-label classifiers as well, using the "one vs all" approach. The relevant class works as the binary positive and the rest of the classes, as the negative. The True Positive Rate of class c is then defined as in Equation 2.7,

$$\mathrm{TPR}_c(th) = \frac{\mathrm{TP}_c(th)}{\mathrm{FP}_c(th) + \mathrm{TP}_c(th)}. \tag{2.7}$$

I.e. of all samples being classified/predicted as c, $\mathrm{TPR}_c(th)$ gives the percentage of predictions that was correct. Where $\mathrm{TP}_c(th)$ and $\mathrm{FP}_c(th)$ is the number of True and False Positives respectively. The False Positive Rate is similarly defined. For the case of zero predictions of class c, the rate is defined as zero. Using these extended definition hence results in one ROC-curve for each class (that has had any predictions).

To reduce the dimensionality of the measurements of the classifier performance, two averages are calculated. The first is later referred to as "micro" and is defined as the number of true/false positives (note that it is not the rate) over all classes divided by the total number of samples, N_s, see Equation 2.8.

$$\text{TPR-micro}(th) = \frac{\sum_{c \in C} \text{TP}_c(th)}{N_s} = \frac{\sum_{c \in C} \text{TP}_c(th)}{\sum_{c \in C} (\text{TP}_c(th) + \text{FP}_c(th))} \tag{2.8}$$

FPR-micro(th) is similarly defined. The other average is referred to as "macro" and is the sum of the sensitivity (TPR)/FPR of each class divided by the number of classes, N_C, see Equation 2.9.

$$\text{TPR-macro}(th) = \frac{\sum_{c \in C} \text{TPR}_c(th)}{N_C} \tag{2.9}$$

FPR-macro(th) is similarly defined. To further clarify, "micro" looks at all samples/predictions at the same time, uniformly weighting each classification with the number of samples. Where as "macro" looks at each individual class subset of the predictions one at the time. The network could for example get a very good TPR-micro(th) result without ever classifying class 0 correctly, especially with few samples of this class. But this would then show as a bad result in TPR-macro(th).

2.2.4 Confusion Matrices

A confusion matrix is a straight-forward visualization of the classifications of the network compared to the true class. The predicted class or *label* is represented by the x-axis and the true label by the y-axis. Figure 2.7 serve as an example of a confusion matrix, where the strength of blue represents the number of samples. For example, 21 samples with the true label 0 was predicted as label 1 by the network.

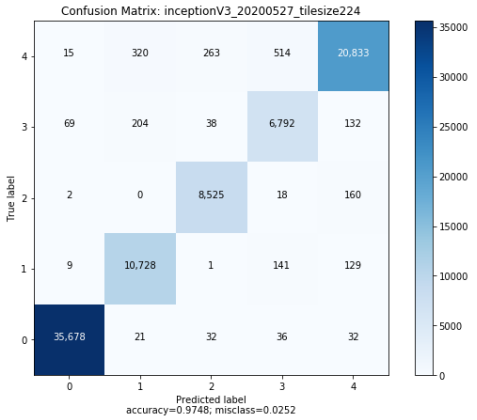


Figure 2.7: An example of a confusion matrix

2.2.5 Moment

If it exists, the n:th moment or momentum, of a one dimensional stochastic variable $x \in \mathbb{R}^1$ is defined as,

$$\mathbb{E}\{x^n\} = \int_{-\infty}^{\infty} x^n \cdot dF(x) \tag{2.10}$$

where $\mathbb{E}\{\cdot\}$ is the expectation operator, F the cumulative distribution function of x and the integral is the Riemann-Stiltje integral which is a generalization of the more common Riemann integral [45]. The various moments can be viewed as measures of various properties of the shape of the distribution (skewness,fatness etc.). The centralized moment of x is the moment of $y = x - \mathbb{E}\{x^1\}$. However this is a bit beyond the scope of this thesis. The first and centralized second moment is more commonly known as the mean and variance of a stochastic variable. Both moments can be estimated through discrete measurements of x.

2.2.6 Normalization

Normalization is a very broad, loosely defined term. But one context where it is used is when one wants to transform a Gaussian (normally distributed) stochastic variable (or samples of such) to a standard Gaussian distribution (zero mean and unit variance/standard deviation). This is achieved by subtracting the mean μ_x and scaling with the standard deviation (the square root of the variance) σ_x. This is also called *standardization*. Of course in practice it will be estimates of said properties.

$$y = \frac{x - \mu_x}{\sigma_x} \tag{2.11}$$

In general, normalization is often used to describe a transformation of data to a more desirable distribution or domain. A typical domain is for example the interval between zero to one.

2.3 Optimization / Training

Algorithms for optimization play a very important part in deep learning, since the objective of fitting the parameters (which can be in the millions) to the data, is not a trivial task. The ideal algorithm quickly finds a set of weights that makes the network generalize (further explained in section 2.3.5). The *curse of dimensionality* (as popularized by Richard E. Bellman [40]) is a concept worth mentioning in this context. Briefly, it means that the more parameters you try to estimate,the larger the uncertainty in the correctness of those estimates. However for neural networks the idea is not to find one true estimate of all parameters but to find any set of parameter values that seem to capture the correct behaviour, by exploring the *loss* of the model (the training). Poggio et al. 2017 [46] explores the issue of why and when deep networks avoids the problems of the curse of dimensionality.

2.3.1 Stochastic Gradient Descent

A basic training algorithm is called stochastic gradient descent (SGD) [47], [48], see Equation 2.12,

$$\omega_{t+1} = \omega_t - \alpha \frac{1}{m} \sum_{i=1}^{m} \nabla_\omega E(\omega_t, x_i, d_i), \tag{2.12}$$

where $\nabla_\omega E(\omega_t)$ is the gradient of the loss function with respect to the weights, α is the stepsize or learning rate. The *stochastic* (random) part is that the m number of data samples is a randomly chosen subset of all available datapoints (allowing for re-picking previously picked samples). These subsets will later be referred to as *batches*, and m as the *batch size*. One of the largest benefits of training on sampled subsets is purely from a computational cost standpoint. An *epoch* refers to when the model has trained on as many samples as there is in the entire training data set, equivalent to a certain number of batches.

2.3.2 ADAM

In this work the popular algorithm ADAM [49] (Adaptive Moment Estimation) is used. It is based on adaptive estimates of lower-order (first and second) moments and takes 4 hyper-parameters. The step size (α), the decay of each moment (β_i) and the numerical precision (ϵ). The step size to take in the parameter space for a given sample, can be interpreted / is popularly referred to, as the *learning rate*. In this work all the default values (as proposed by the original paper) are used, $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$, unless otherwise stated. The following Equations (2.13, 2.14, 2.15) describe the algorithm, where m, v is the first respectively second estimated moment. g is the loss gradient w.r.t. the weights, $g_t = \nabla_\omega E(\omega_t, x_i, d_i)$. t is the iteration index and i the sample index. The moments are initialized to zero, $m_0 = v_0 = 0$.

$$m_{t+1} = (\beta_1 m_t + (1 - \beta_1) g_t)/(1 - \beta_1^t) \tag{2.13}$$

$$v_{t+1} = (\beta_2 v_t + (1 - \beta_1) g_t^2)/(1 - \beta_2^t) \tag{2.14}$$

$$\omega_{t+1} = \omega_t - \alpha \cdot m_{t+1}/(\sqrt{v_{t+1}} + \epsilon) \tag{2.15}$$

2.3.3 Back-propagation

Back-propagation describes the process of how the errors made by the models predictions during training, "propagates", back through the models layers. And how the weights are adjusted layer by layer, thereafter [50]. In this work the gradient extension of back propagation is used [33]. Let W_l be the weights going into layer l. X_0 the original input and X_l the output of layer l. Let $F_l(W, X) = X_l$ be the input-output mapping of layer l and $E(W, X)$ the loss function (ignoring the true label data since they are a constant with respect to the training). Both are assumed to be differentiable. Then for a given layer l, the partial loss gradients w.r.t. the weights and hidden outputs, is described by Equations 2.16 and 2.17.

$$\frac{\partial E}{\partial W_l} = \frac{\partial F_l(W_l, X_{l-1})}{\partial W} \cdot \frac{\partial E}{\partial X_l}, \tag{2.16}$$

$$\frac{\partial E}{\partial X_{l-1}} = \frac{\partial F_l(W_l, X_{l-1})}{\partial X} \cdot \frac{\partial E}{\partial X_l}, \tag{2.17}$$

where $\partial F_l(W_l, X_{l-1})/\partial W$ and $\partial F_l(W_l, X_{l-1})/\partial X$ are the Jacobians of F w.r.t W and X, evaluated at (W_l, X_{l-1}). The process can be described as such, an input sample propagates through the network updating the outputs of the network's nodes/layers. Next, the error made by the model propagates back through the network updating the weights. Where the update is based on the error from previous outputs and weights.

2.3.4 Vanishing Gradient

Networks with many layers (i.e. *deep*), for most architectures, suffers from a *vanishing gradient*. This problem refers to a numerical issue during the network training, when calculating the weight changes (gradient of loss) in the back-propagation of the output. As the networks error "propagates" back through its layers, during the training, a multiplication is performed for each layer (due to the chain rule of derivation). If both values are less than one (which most often is the case due to normalizations) the result shrinks and vice versa. With the result that the differences goes to zero for layers close to the input and hence their weights will stay fixed. A typical network that has problems with vanishing gradient is Recurrent Neural Networks (RNN), typically used for time-series (where the depth corresponds to the length of the series) [51]. There are several methods of how to reduce this problem to enable deeper networks. One is to modify the activation function as described in Section 2.2.2. Another is to use skip-layers (i.e. to have some connections go to the second next layer for example), creating identity function modules within the network as described by He et al. [52].

2.3.5 Regularization

The overarching goal of any neural network is generalization. That is, the model should capture some important features/characteristics that holds true for data that the network has not seen before, i.e. beyond the training data it has learned the features from. This introduce a trade-off between over- and under-fitting to the training data. Over-fitting means that the network has adapted to/learned specific/unique characteristics of the training data set that do not generalize well outside of it. Under-fitting on the other hand means that the model do not capture enough characteristics. An indicator of over-fitting is when the network performs well on the training data but not on other data sets. Under-fitting is just characterized by poor performance. To avoid over-fitting different techniques are introduced under the umbrella term "regularization".

- Weight regularization, is typically performed using the L2-norm. This simply means that the norm of the network weights is added to the loss-function
- Dropout, means that during training, a certain percentage (the dropout frequency) of nodes are removed from the network at random. Creating a sub-network of the original.

- Stochastic learning / Batch training, has mainly computational benefits and also enables online-training (i.e. training sample by sample (or batch-wise)). It means that instead of training on the entire data set at the same time, the network trains on randomized *batches* (subsets of the entire data set). I.e. for each batch, the weights are updated according to the learning rate.

2.3.6 Early Stopping

Another issue during training is when one should stop the optimization and settle for a given set of weights. There are several approaches to this but in this thesis the *early stopping* approach is used. This means that the model is evaluated during training on a validation data set after each epoch . If the model performance on the validation data set stops improving during the training, the optimization stops and the weights resulting in the best performance on the validation data set so far is chosen. In this thesis the performance is measured by the prediction accuracy for determining when to stop.

2.3.7 Transfer Learning

Transfer Learning is a concept that has proven to work well on several domains [53]. The idea is that the weights of a network is initially trained on a large data set, such as ImageNet. After this, only the weights of the outer layers of the network is trained on another, more particular, data set. It can be interpreted as the initial layers learning general features of the images, such as shapes and color compositions, and the final outer layers making classifications, or "decisions", based on those features. For transfer learning to work, it assumes that the features learned from the initial data set is also present in the newer one. Note that the ImageNet do not consist of medical data and that it is not obvious for what depth of features (if any) learned on ImageNet, that would translate well for medical images.
Another, very similar concept is to initialize a network with pre-trained weights. The only difference here is that all weights get re-trained, unlike in "true" transfer learning where the initial weights stay fixed all the time.

Chapter 3
Material and Methods

In this chapter, the data, the pre-processing of the data, and the network models are presented.

The original medical whole-slide images were saved in a file-format called ".mrxs" due to their data size. The file-format includes several versions of the same image at different resolutions. This work, however, only makes use of the images at the highest resolution. Sub-images of the whole-slides were exported as image pairs using Qupath [54]. Qupath is a program for analysis and annotation of histopathological images. The colored annotation was exported as .png and the actual image as .jpg. These images was then imported into Python [55] as numpy arrays i.e. 3-dimensional tensors where the last dimension represents the 3 RGB-channels of the images. The original whole-slides was reconstructed and the resulting images was then split into tiles of suitable sizes. Using python, the major work of the thesis was done with the Keras [56] and Tensorflow [57] libraries. The communication with the graphic processor is done via these libraries, who rely on the CUDA library [58]. The training was done initially using a single GTX 1060 (6GB VRAM) and later, on a RTX 2060 Super (8GB VRAM).

Four main architectures are used to classify tiles: Linear regression, compactVGG (further described in Section 3.5.2), VGG16 and InceptionV3. Where the two prior works mainly as baselines. If a parameter is not specified it will be the default value chosen by Keras. The architectures used is summarized in Table 3.1.

Model	Description
Linear regression	sanity test, do we gain anything from using neural networks?
VGG16	large pre-trained, all 3x3 kernels
compactVGG	small VGG inspired architecture, do we gain anything from having larger networks?
InceptionV3	large pre-trained, using different sized kernels in same layers (the inception module)

Table 3.1: Descriptions of the four main models considered

3.1 Data and Pre-Processing

The data consisted of whole-slide images of Hemotoxylin-Eosin (H&E) colored tissue-slices from tumor biopsies (see Figure 3.1). The biopsies are from lymphnode metastases of patients with malignant melanoma (skin cancer). The images were manually annotated in Qupath by a pathologist. This of course means that the network will be heavily influenced by this particular pathologist and the possible errors that he/she has made. However, the idea is that, since the images are largely correctly annotated, the networks will be able to generalize beyond minor errors. Since the annotation is made manually there is some overlap where the same pixels is annotated differently. When exporting the images, an adhoc solution was used where the highest class number was used to decide what class the conflicting pixels should be. Due to the nature of how the annotation was done a lot of white background area was annotated as belonging to some class of tissue. This led to two decisions. First, a crude pre-processing/washing of background pixels seemed reasonable, where pixels was re-annotated as background if their mean RGB-value were above an arbitrarily set threshold. Second, networks for semantic segmentation (i.e. classifying on a pixel-based level) such as U-net [59] would not be considered.

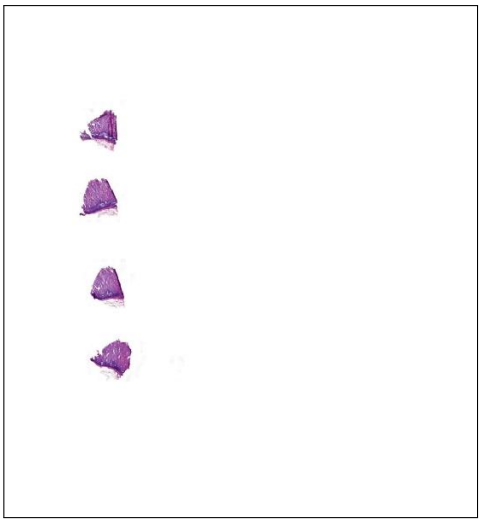


Figure 3.1: Four tissue slices on a microscopic whole-slide image

Figure 3.1 corresponds to roughly 66×71mm. Note that most of the whole-slide image consists of background. The pixel resolution for the whole-slides was 0.2425μm$\times 0.2426\mu$m (width $\times$ height). All slices on a whole-slide are from the same biopsy but at different heights and only one of the slices is annotated (annotation is not seen in Figure 3.1).

The annotated sections of the whole-slides was divided into sub-images and exported to .jpg(real image) and .png(annotation image) files using a Qupath-script. The reason for not exporting the entire whole-slide directly was purely from a execution time standpoint. The maximum size of the sub-images was chosen as roughly 4000x4000 pixels. An example of resulting image pairs can be seen in Figure (3.2), where a color in the annotation corresponds to a certain tissue type.

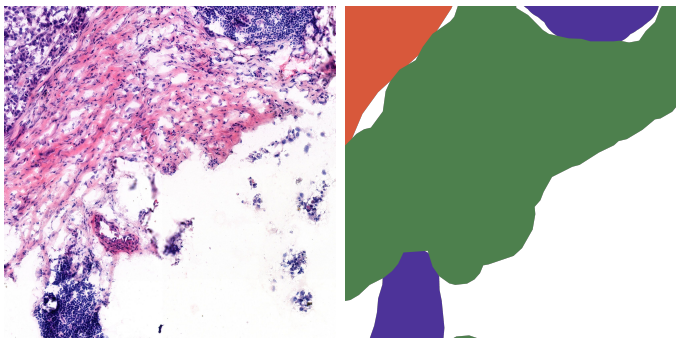


Figure 3.2: 4000x4000 image, original annotation to the right

These images was then put back together to re-create the original whole-slide, before tiling it into smaller tiles, to be used as input to the models. The initial size of the tiles was chosen as 224×224 pixels, since this is the dimensions of the ImageNet dataset images. This corresponds to a real area of 54.32μm×54.34μm. The tiles was converted to *numpy* arrays of datatype uint8 to save memory, since the pixels RGB-values range between 0 and 255. Three additional tilings were done as well but shifted half the tile size in the x and/or y direction respectively.

There was 5 different classes annotated. "Other" (background), "immune cells", "necrosis", "stroma" and "tumor". The tiles were randomly split up into three sets corresponding to training (4/6), validation(1/6) and test(1/6). I.e., such that a proportional number of tiles from each whole-slide ended up in each set. The training set is used to train the model weights, the validation set is used to evaluate the model during the training (for early stopping) and to choose a set of weights for the model. The test set is used to evaluate the final performance of the model. In Figure 3.3, examples of tiles from respective class is shown.

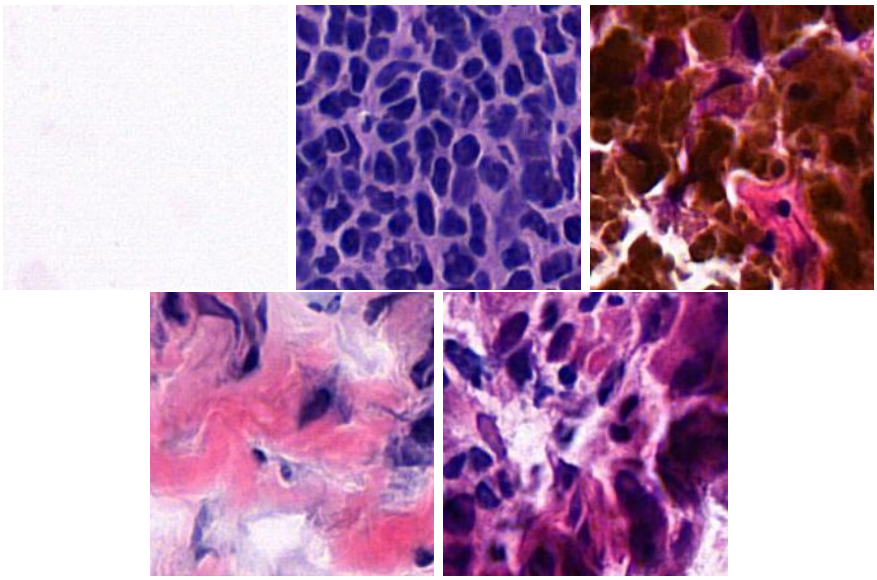


Figure 3.3: 224x224 tiles. Upper row left to right: Other, immune cells, necrosis. Lower row left to right: stroma, tumor.

After the images were tiled, the pixels of all tiles were automatically re-annotated. If the mean of a pixel value was larger than $255 - th_{wh} = 240$ it was re-annotated as background. I.e. equivalent of $th_{wh} = 15$. This threshold was chosen based on what seemed to give subjectively reasonable results. The main aim was simply to re-annotate the most visually obvious, background regions of the images.

3.1.1 Ground truth

Next the *ground truth* classification (true labels) of the tiles was done based on their annotated image pair. I.e. each tile was assigned a class/tissue type. If the percentage of pixels annotated as background was above a certain threshold ($th_{am} = 95\%$), the tile was classified as background. Else wise, if the percentage of pixels annotated as background and the dominating class was above the threshold, the tile was labelled as the dominating class. Otherwise the tile was deemed as *ambiguous* and thrown away. I.e. in a worst case scenario a tile with 48% pixels belonging to class a (not being "Other"/background) and 47% pixels belonging to background would still be classified as belonging to class a. However if the ratios had been reversed the tile would be classified as ambiguous. All tiles was then converted to images and saved to an appropriate folder. The data labels were saved as a numpy array. In total three main datasets was created and used as described in the following sections 3.2, 3.3 and 3.4.

3.2 Initial Dataset I

This was the initial dataset of 6 whole-slides, that was mainly used as a test for the code as well as to determine whether the more complex/larger CNNs improved the results compared to a basic regression model. Hence, only one tile size of 224 was used. During the tiling process for this set only one extra, diagonally shifted, tiling was performed to limit the size of the data set and thus the training time. The values presented in Table 3.2 is the number of tiles of a corresponding class and set. This includes the shifted tiles but not the ambiguously annotated ones. Using tile size 224×224. In table 3.2 the number of tiles for each class and set is presented.

Set\Class	Other	immune cells	necrosis	stroma	tumor	Total
Training	12408	5686	2803	6098	8001	34636
Validation	7082	3344	2038	2554	5067	20085
Test	7070	3345	2063	2468	5051	19997
Total	26560	12375	6904	11120	18119	74718

Table 3.2: Data distribution, tilesize 224, dataset I

Table 3.2 shows that the class distribution was not uniform. Having an uneven data distribution can impact the training, for instance having lots of samples of class A but few of class B will make the network to only learn to identify class A. A possible solution is to simple remove samples from the training set so that it becomes uniform. This should not be done for the validation or test set, since they would then no longer mirror the true performance. However there are lots of samples for all classes and the difference in number of samples is at least less than a factor ten (commonly used as order of magnitude), hence it was decided to not modify the training set.

3.3 Extended Dataset II

This was the extended data set used to train the final models and for examining the impact of magnification, i.e. tilesize, on the performance of the models. Several more whole-slides were added, giving a total of 17 whole-slides. The whole-slides was tiled for three different tile sizes of 448,224 and 112 pixels. Further more, during the tiling process for each tile size, three extra, shifted tilings was performed with a shift of half the tile size. The shifts was performed horizontal to the right, vertically down and a combination of both i.e. diagonally. The values presented in the following tables are the number of tiles of a corresponding class and set. This includes the shifted tiles but not the ambiguously annotated ones. Similarly to the Initial Dataset I, no tiles were removed to make the class distribution more uniform.

3.3.1 Dataset224 II

Using tile size 224×224. In table 3.3 the number of tiles for each class and set is presented. Note that "Dataset I", under section 3.2, is a subset of "Dataset224 II".

Set\Class	Other	immune cells	necrosis	stroma	tumor	Total
Training	143887	44424	34403	28714	87874	339302
Validation	36346	11068	8859	7273	21976	85522
Test	35799	11008	8705	7235	21945	84692
Total	216032	66500	51967	43222	131795	509516

Table 3.3: Data distribution, tilesize 224, dataset II

3.3.2 Dataset448 II

Using tile size 448×448. In table 3.4 the number of tiles for each class and set is presented.

Set\Class	Other	immune cells	necrosis	stroma	tumor	Total
Training	32535	10765	8831	6906	21649	80686
Validation	8066	2720	2155	1694	5433	20068
Test	8091	2702	2198	1684	5443	20118
Total	48692	16187	13184	10284	32525	120872

Table 3.4: Data distribution, tilesize 448, dataset II

3.3.3 Dataset112 II

Using tile size 112×112. In table 3.5 the number of tiles for each class and set is presented.

Set\Class	Other	immune cells	necrosis	stroma	tumor	Total
Training	611872	178256	133275	115525	348461	1387389
Validation	152551	44585	33148	28948	87090	346322
Test	152483	44375	33248	28716	87192	346014
Total	916906	267216	199671	173189	522743	2079725

Table 3.5: Data distribution, tilesize 112, data set II

Note that all the final models used, for each respective tile size, was trained only on the training sets of Dataset II.

3.4 Dataset III

After the models had been trained on data set II and evaluated on its test data, the models was evaluated on tiles from two whole-slides not previously used, these tiles are referred to as Dataset III. Important to note is that since this tests purpose was to simulate a real situation where the actual annotation is not known, the ambiguously annotated tiles was not removed. However the pixel level annotations of the annotation tiles was still re-annotated for background pixels. This was done for the purpose of determining the class belonging of the tile, just the same way as for the other data sets.

3.4.1 MM1279-1-12-23

The whole-slide MM1279-1-12-23 and its corresponding data is also referred to as "testdata0". For visualization of the annotation each class is assigned a color. Figure 3.4 shows the re-constructed original whole-slide to the left and its annotations, before and after, re-annotating background pixels, to the right. The whole-slide has 19500×22000 pixels, corresponding to 4.7×5.3mm.

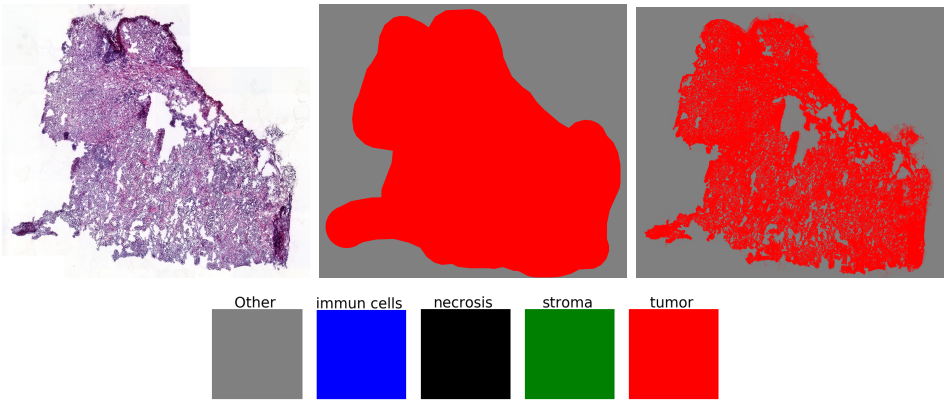


Figure 3.4: test whole-slide MM1279-1-12-23

The whole-slide in Figure 3.4 was chosen as a test slide due to the fact that it was annotated as only tumor, which was the the main tissue type of interest.

Table 3.7 shows the distribution after the re-annotation. It also includes the percentage of tiles classified as *ambiguous* as described in section 3.1.1, using a threshold of 90%.

Tilesize\Class	Other	immune cells	necrosis	stroma	tumor	Total	Ambiguous[%]
448	1069	0	0	0	1038	2107	7.55%
224	4461	0	0	0	4065	8526	6.19%
112	18241	0	0	0	15863	34104	5.74%

Table 3.6: Data distribution, testdata0

3.4.2 MM812-19-25-31

The whole-slide MM812-19-25-31 and its corresponding data is also referred to as "testdata1". Figure 3.5 shows the re-constructed original whole-slide to the left and its annotations, before and after, re-annotating background pixels, to the right. The whole-slide has 20700 × 28100 pixels, corresponding to 5.0 × 6.8mm.

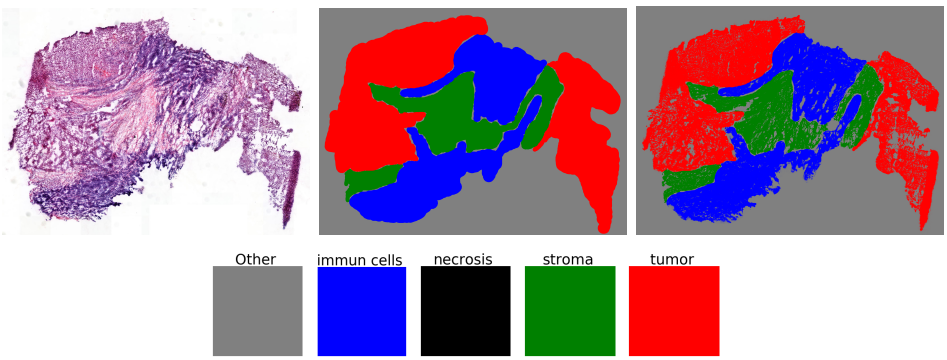


Figure 3.5: test whole-slide MM812-19-25-31

The whole-slide in Figure 3.5 was chosen due to the fact that it contained almost all tissue types, fairly evenly distributed, except necrosis (black).

Table 3.7 shows the distribution after the re-annotation. It also includes the percentage of tiles classified as ambiguous as described earlier, using a threshold of 90%.

Tilesize\Class	Other	immune cells	necrosis	stroma	tumor	Total	Ambiguous[%]
448	1335	483	0	314	720	2852	10.17%
224	5571	1882	0	1215	2832	11500	6.88%
112	22807	7452	0	4732	11009	46000	5.40%

Table 3.7: Data distribution, testdata1

3.5 Models

All versions of VGG16 and InceptionV3 was imported pre-trained on ImageNet via Keras. The networks was imported without the last fully connected part, i.e. only the convolutional layers. Instead a fully-connected hidden layer of 1024 nodes was added (with dropout frequency 0.2 and ReLU activation function) and then the final output layer of 5 nodes (which uses softmax as activation function). The input dimensions (i.e. tilesize) was set using a input argument to the networks.
If weights is not pre-trained on ImageNet, the weights of each layer are randomly initialized from the uniform Glorot distribution [60] by default [56]. This is a uniform distribution with limits $\pm\sqrt{6/(x_{in}+x_{out})}$. Where x_{in} and x_{out} is the number of values going in and out of the layer, i.e., corresponding to the number of weights in each direction. In appendix A, overviewing graphs of three different network implementations used are shown for 448x448 tiles.

3.5.1 VGG16

Ignoring the input-size changes for different tiling, three variations of VGG16 was used. All of which was imported using the Keras library. "vgg16_20200508" and vgg16_20200510 (which made use of transfer learning), were both trained on Initial Dataset I. And "vgg_20200527" which was trained on Extended Dataset II and also made use of transfer learning.

The models using transfer learning only trained the last three convolutional layers and the following dense neural network, see Figure A.1 in the Appendix.

All VGG16 models convolutional layers use stride 1 and one layer of zero-padding, to keep the x,y dimensions of the layers input-output tensors constant. The max-pooling layers use stride 2 which halfs the x,y dimensions (i.e. the position of the "pixels") of the tensors. The convolutional layers use the ReLU activation function. The structure of the initial convolutional part of the network is split into 5 blocks of 2 or 3 convolutional layers, using the same number of kernels and 1 max-pooling layer. At the end of the convolutional part it uses a global average pooling layer before the final dense neural network layer. The global part refers to that the window has the dimensions of the input tensor, i.e. no striding/convolution, which results in that the output is a vector with the length of the previous number of kernels used. Too compensate for the gradually decrease in the x,y size of the tensors the number of kernels between blocks is gradually increased.
The total number of trainable parameters/(non-locked) weights for the models is 7609861 regardless of tile/input size.

3.5.2 compactVGG

This is a smaller model that was used to examine if the large sizes of the other networks improved the results. It is based on the same structure as VGG with only 3 convolutional windows for the kernels and the ReLU activation function. The stride is set to 1 and it uses no zero-padding for the convolutional layers. The max-pooling layers use a window of 2 and a stride of 2 as well, with no zero-padding. The convolution part of the model consists of 5 blocks of one convolution layer and one max-pooling layer. The number of kernels used in each layer is gradually decreased from 9 to 3. The final dense neural network part have a hidden layer of 25 nodes (with dropout frequency 0.2), before the output layer of 5 nodes (one for each class). The ReLU activation function is used for all nodes except the output where Softmax is used.
The total number of trainable parameters/(non-locked) weights for the models of tilesizes 448, 224, 112 is 12032, 3107, 1307 respectively.

3.5.3 InceptionV3

The model was imported using the Keras library. An overview of its structure can be found in Figure A.3 in the Appendix. Further details on its implementation can be found at the Keras website [56] and the original paper [22].
The total number of trainable parameters/(non-locked) weights for the models is 23871653 regardless of the tile/input size.

3.5.4 Linear Regression

In this thesis the tile features used to identify and classify tiles in the regression model was the mean and standard deviation of each RGB-channel of the tiles. I.e. each tile generated a data point of 6 values used to classify 5 classes. The main purpose of this regression model was a sanity test of the other models. I.e. did they improve anything from this very basic model? Or could the classification be done based on these basic color metrics. This linear regression model is later referred to as "linreg_20200512".

3.6 Training

The training of the networks was done using the "fit" method in Keras [56]. It uses stochastic gradient descent in an attempt to optimize the models weights to the data. To generate data-batches for the optimization, the Keras function *ImageDataGenerator* was used [56]. Which in this thesis case, fetches a batch of images from a directory and both pre-process and randomly augments them. The purpose of the pre-processing here is just to force the data to have nice numerical values. Since the images pixels is represented in 8-bit integer RGB (white is for example represented by (255,255,255)) the images is first scaled by $1/255$, next the mean of all values in the image is subtracted ("centering") and the resulting image is scaled by the inverse standard deviation. Notice that this pre-processing must be done on validation- and test-data as well, before the model can make predictions. The random augmentations was only used during training and consisted of flipping the image vertically/horizontally and rotating the image between 0 to 45 degrees.

The batchsizes used during the training was 12,8 and 1 samples, times a model dependant factor bs, for the three different tilesizes 112,224,448. CompactVGG used bs =64, VGG16 bs =32 and InceptionV3 bs =16. The main reason for using different batch sizes was too make efficient use of the GPU memory. But different batch sizes can also impact the training, generally smaller batch sizes cause over training and vice-versa.

Chapter 4
Results and Discussions

In this chapter the results of the thesis is presented and discussed. All results are evaluated on the test set, for all data sets. I.e. the model has never before seen/trained on, the samples it is making its predictions on.

4.1 Dataset I

In this section a brief summary of the results is presented. Further details (ROC,AUC and confusion matrices) is shown in the Appendix B. In Table 4.1 the prediction accuracy of the models trained on Dataset I (which only used tilesize 224) is presented.

Model	Accuracy
linreg_20200512	0.685
myCNN_20200508	0.906
inceptionV3_20200508	0.945
vgg16_20200508	0.354
vgg16_20200510	0.935

Table 4.1: Prediction accuracy, Dataset I

The results from vgg16_20200508 implies that it had not been trained at all. This is most likely due to the issue of dying ReLU or vanishing gradient since it is a deep network and it did not make use of transfer learning. As opposed to vgg16_20200510. The simple linear regression model did not perform very well either compared to the other models. From these results, the conclusion was drawn that for future model testing, VGG16 would only be trained using transfer learning and linear regression would not be used.

4.2 Dataset II

In this section a brief summary of the results is presented. Further details (ROC,AUC and confusion matrices) is shown in the Appendix B. In Table 4.2 the test prediction accuracy of the models trained on Dataset II is presented

Tilesize\Model	compactVGG	InceptionV3	VGG16
448	0.959	0.948	0.974
224	0.929	0.975	0.963
112	0.903	0.961	0.939

Table 4.2: Prediction accuracy, Dataset II/Test

Note that this is the result on randomized tiles sampled from all whole-slides used in the training, all though these particular tiles has not been used for neither training nor validation. This can be compared to where the entire whole-slide is new as can be seen in the next section.

Both VGG networks seem to have a reduced performance as the tilesize decrease and the larger VGG16 consistently outperforms compactVGG with around 1.5-4%-units. InceptionV3:s performance does not share this pattern and also seem to have less variance in its performance w.r.t. the tile size.

4.3 Dataset III

In this section some of the test performances on the whole-slides MM1279-1-12-23/testdata0 and MM812-19-25-31/testdata1 are presented. The predictions are made by first tiling the whole-slides and then the networks make a prediction for each tile. After this the whole-slide is re-constructed, coloring the tiles according to the predicted class. Note that ambiguous tiles with several different true annotated classes is not removed. Instead the dominating class (determined as described in section 3.1.1) is used.

4.3.1 MM1279-1-12-23/testdata0

In Figure 4.1 the original whole-slide and the used ground truth annotation is shown.

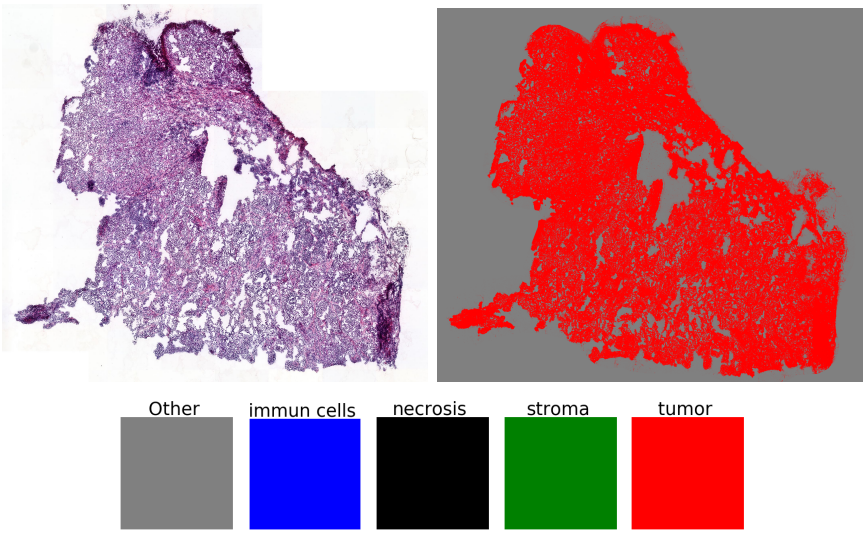
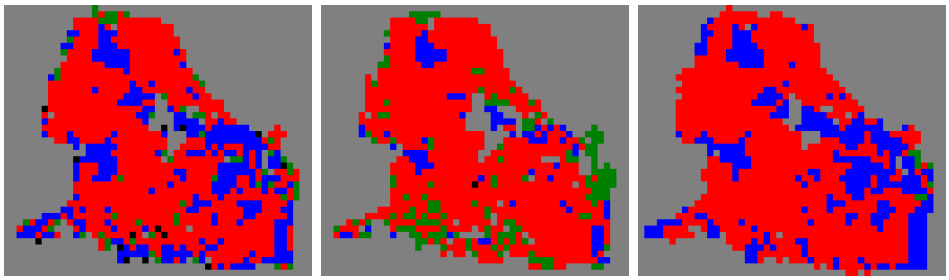
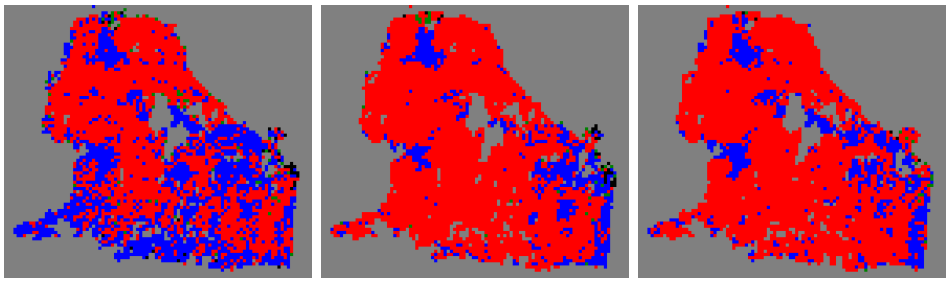


Figure 4.1: test whole-slide MM1279-1-12-23, ground truth annotation

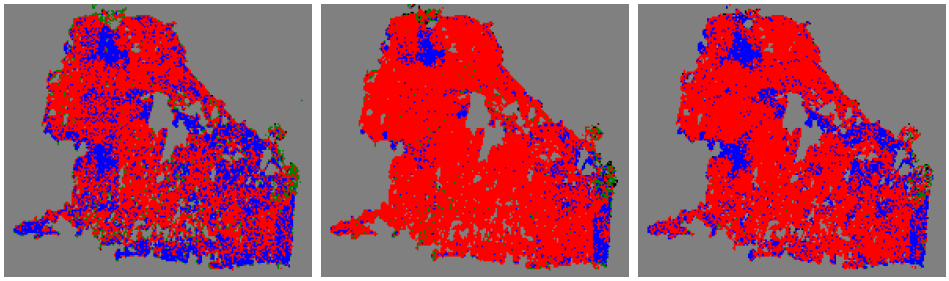
In Figure 4.2 the predictions of the three trained models, on each tilesize, is depicted. From left to right: compactVGG, InceptionV3 and VGG16. From up and down, tilesizes: 448,224,112



(a) tilesize 448



(b) tilesize 224



(c) tilesize 112

Figure 4.2: model predictions on MM1279-1-12-23

As can be seen in Figure 4.2, where the ground truth is that all tissue is tumor tissue (red), the networks have a hard time differentiating between tumor (red) and immune cells (blue). This makes sense, since they look similar for an untrained human eye as well. It is also interesting to note that InceptionV3 (the middle) seems to improve, differentiating from immune cells, as the tile size is reduced.

It can also be seen in the original whole-slide of Figure 3.4, that the sections in the upper middle and lower right, where the networks seems to predict immune cells, has differentiating (compared to other tumor areas) characteristics such as denser, smaller cells and a darker blue/purple color. This is further illustrated in Figure 4.3. This could be an example of incorrect labelling by the pathologist. If so, this would be very encouraging since the network then actually manage to perform despite errors found in the ground truth it has trained on. Another possible explanation is that the areas consists of an artefact called tissue-folding. This simply means that when the tissue-slice is put between the two glass disks, the tissue folds in on itself. Figure 4.3 shows the upper middle part zoomed in, of the whole-slide.

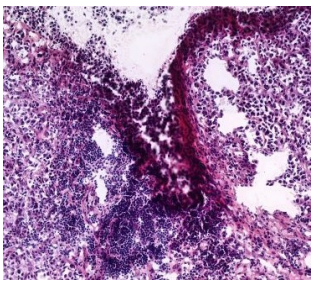


Figure 4.3: test whole-slide MM1279-1-12-23, zoomed in on upper middle area

In Table 4.3 the resulting prediction accuracy of the models are presented.

Tilesize\Model	compactVGG	InceptionV3	VGG16
448	0.816	0.872	0.802
224	0.785	0.896	0.887
112	0.812	0.915	0.874

Table 4.3: Prediction accuracy on MM1279-1-12-23

4.3.2 MM812-19-25-31/testdata1

In Figure 4.4 the original whole-slide (left) and the used ground truth annotation (right) is shown.

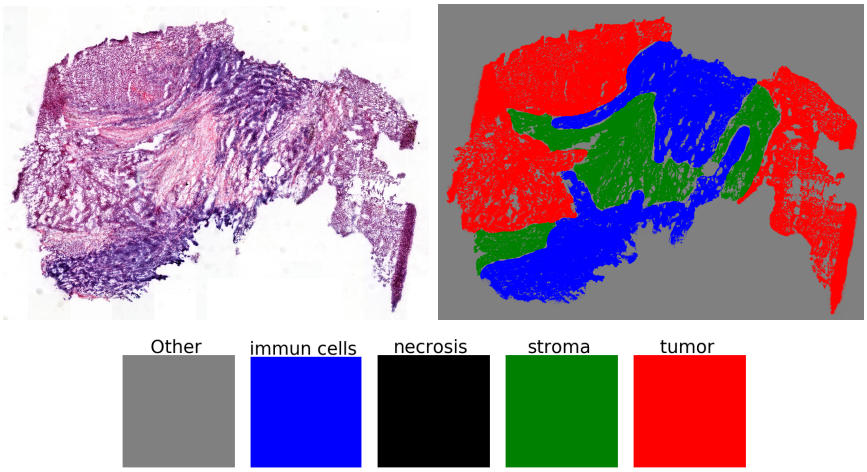
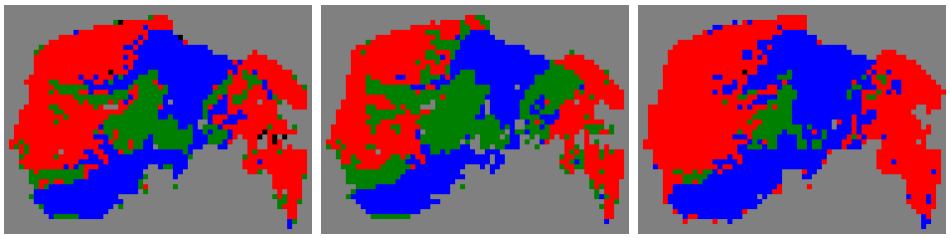
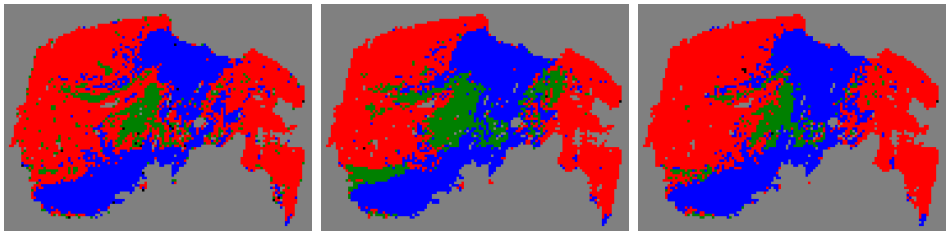


Figure 4.4: test whole-slide MM812-19-25-31, ground truth annotation

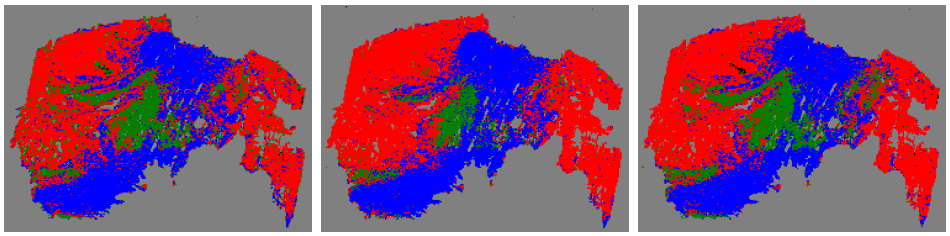
In Figure 4.5 the predictions of the three trained models is depicted. From left to right: compactVGG, InceptionV3 and VGG16. From up and down, tilesizes: 448,224,112



(a) tilesize 448



(b) tilesize 224

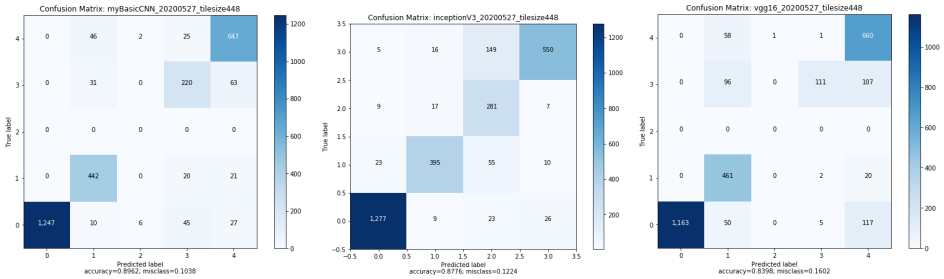


(c) tilesize 112

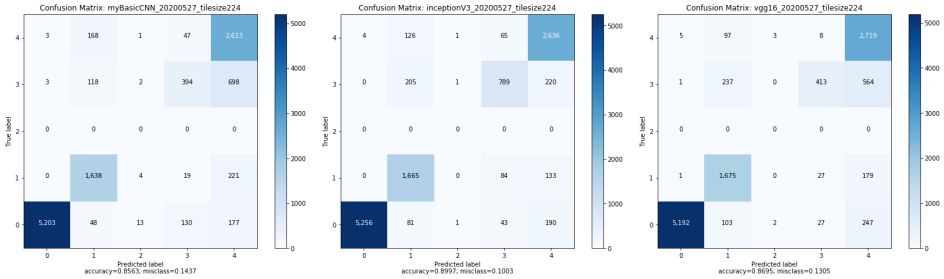
Figure 4.5: model predictions on MM812-19-25-31

Looking at the predictions in Figure 4.5 and comparing with the ground truth in Figure 3.5, one can observe that the models seems to get it mostly right. However it still has problems at the boundaries between different tissue types, even for smaller tile sizes. Especially, the models seem to confuse stroma (green), with immune cells (blue) and tumor (red). It is also clear that the same model, gets trained to predict different classes on the same areas, depending on the tile size. Ideally a lower tile size would just mean better resolution in the predictions, but this does not seem to be the case. This could perhaps be caused by the errors in the annotation, especially around tissue boundaries. Another reason for this could be artefacts from the way the discretized ground truth is established.

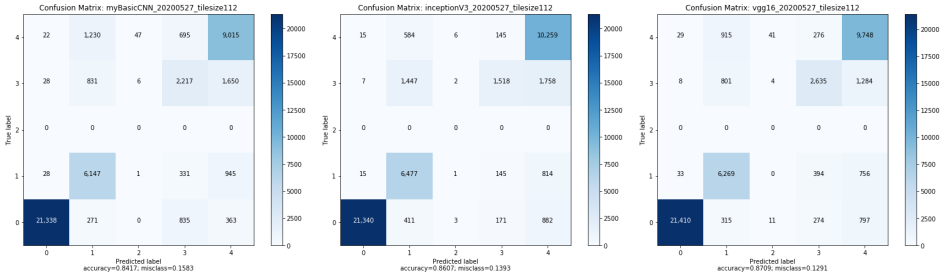
In Figure 4.6 the confusion matrices of the respective models are presented. From left to right: compactVGG, InceptionV3 and VGG16. From up and down, tilesizes: 448,224,112. Where 0:Other, 1:immune cells, 2:necrosis, 3:stroma, 4:tumor.



(a) tilesize 448



(b) tilesize 224



(c) tilesize 112

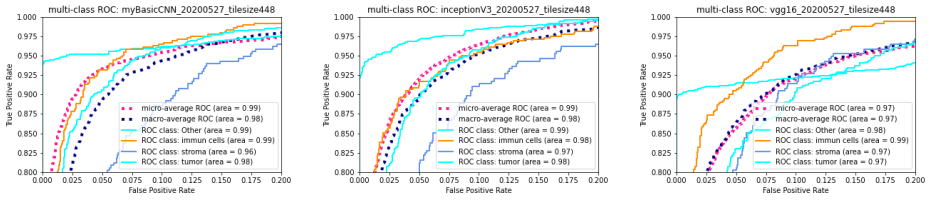
Figure 4.6: Confusion Matrices on MM812-19-25-31

In Table 4.4 the resulting prediction accuracy of the respective models are presented.

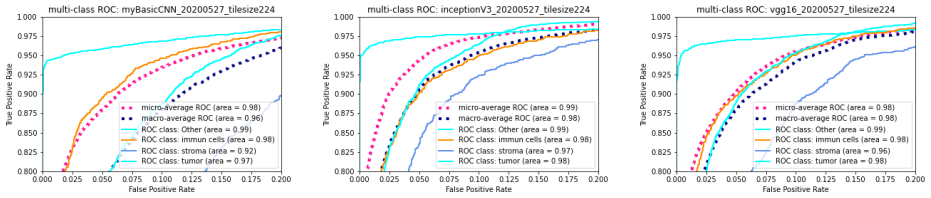
Tilesize\Model	compactVGG	InceptionV3	VGG16
448	0.896	0.878	0.840
224	0.856	0.900	0.869
112	0.842	0.861	0.871

Table 4.4: Prediction accuracy on MM812-19-25-31

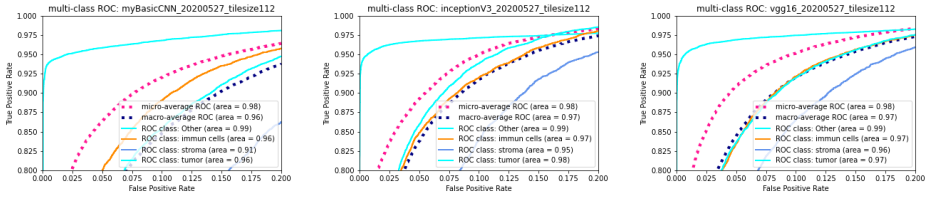
In Figure 4.7 the resulting ROC-curves of the respective models are presented. From left to right: compactVGG, InceptionV3 and VGG16. From up and down, tilesizes: 448,224,112. In Table 4.5 the corresponding AUC i presented.



(a) tilesize 448



(b) tilesize 224



(c) tilesize 112

Figure 4.7: Receiving Operating Characteristic (ROC) curves, MM812-19-25-31

Table 4.5 shows the AUC from the ROC-curves.

AUC							
Tilesize 448	Other	immune cells	necrosis	stroma	tumor	micro average	macro average
compactVGG	0.989	0.987	nan	0.957	0.984	0.985	0.979
InceptionV3	0.995	0.983	nan	0.967	0.983	0.987	0.982
VGG16	0.977	0.986	nan	0.966	0.969	0.974	0.975
Tilesize 224	Other	immune cells	necrosis	stroma	tumor	micro average	macro average
compactVGG	0.991	0.980	nan	0.921	0.967	0.980	0.965
InceptionV3	0.992	0.982	nan	0.969	0.984	0.989	0.982
VGG16	0.991	0.982	nan	0.960	0.981	0.984	0.978
Tilesize 112	Other	immune cells	necrosis	stroma	tumor	micro average	macro average
compactVGG	0.989	0.963	nan	0.912	0.957	0.975	0.955
InceptionV3	0.987	0.974	nan	0.948	0.976	0.983	0.971
VGG16	0.990	0.972	nan	0.956	0.973	0.984	0.973

Table 4.5: Area Under the Curve (AUC) MM812-19-25-31

Looking at Table 4.4 it is clear that the models have most difficult to distinguish stroma from the other classes. When one goes through the whole-slides one can note that areas with stroma often tends to be smeared out with a lot of background which could perhaps be a part of the problem.

Chapter 5
Summary

This thesis work focused on the impact of *magnification* when classifying tissue-types using CNNs. It did not find any major performance differences for the models on different tile sizes. However, InceptionV3 at tilesize 224 seems to give the best results generally. One should be aware though, that the Keras implementation of InceptionV3 is based on the version using 299×299 images as input, and hence there could be a bias towards this size, since the network was initially designed for it. The issue of possibly incorrect ground truth, especially for small areas and borders between different tissue types (i.e. if the annotation is not detailed/precise enough), is a factor that should be taken into account when viewing the results. It could implicate that the true performance is actually better than measured. Another annotation for the same data by a second pathologist would facilitate the interpretation of the results. Is the model making an error according to both pathologists?

There is a clear discrepancy between the performance on the test data from Dataset II and the two test whole-slides in Dataset III. This is a bit discouraging since it implies that the network somehow has overtrained on the whole-slides albeit not on the exact tiles it trained on. The percentage of ambiguous tiles is of course also an explanatory factor for the discrepancy. Obviously there is going to be variance between the whole-slides due to artefacts from how that particular slide was produced. For example, the tissue could become damaged or smeared from the glass or during the slicing, and differing amount of coloring agent (H&E) will produce variations in colors. So a section of tumor may look different from whole-slide to whole-slide. More whole-slides to train on than the 17 used, would hopefully help bridge the performance gap.

Using the overall accuracy as performance measure for the validation and test set was perhaps not the best choice since the sets was not uniformly distributed. This means that models performance on classifying, for instance background tiles, had a disproportionate impact on the measured performance. A better measure would perhaps be to look at the ratio between correctly classified tiles of class c, and all true and false positives of c. And then average the result for each class. One could also argue that classifying the background should be done by a separate model and should not be included at all when evaluating the performance since the background has very clear characteristics (homogeneous roughly white pixels) which is actually also used in the pre-processing.

5.1 Future Work

During the work for this thesis a lot of possible variations was not tested due to limited computational power and time. In this section some of those variations are listed.

- Hyper-parameter optimization on training (learning rate etcetera) and pre-processing (thresholds for ambiguous tiles etcetera)
- Add a class for ambiguous tiles
- Train on continuous labels of percentage of pixels belonging to a certain class instead of discretizing the ground truth
- Optimize/improve how pixels are re-annotated as background
- Since the annotations contains errors one could also try to use newer methods to try to take this into account, for example changing the loss function.

A more ambitious continuation would be to train several networks on different tile sizes and then either weight the predictions of overlapping tiles together by a fixed set of weights or use the predictions as input to a final network. One could also Introduce a model solemnly to classify background tiles.

It would also be interesting to have another pathologist correct the predictions of the network, mainly as a source of second opinion on the ground truth but also to correct the performance measures of the networks.

Another continuation would be to compare the CNNs performances with more statistical models than just the multi-nomial logistic regression model. One could test different Generalized Linear Models (GLMs), Supporting Vector Machines (SVM) and Bayesian Regression Tree / Random Forest models. One could also extend the explanatory variables/signals, using more than just the mean and standard deviation of the color channels used in this thesis.

References

[1] Marko Bursac, Danijela Milosevic, and Katarina Mitrović. Proposed model for automatic learning style detecting based on artificial intelligence. 05 2019.

[2] Brad Boehmke and Brandon M Greenwell. *Hands-on machine learning with R*. CRC Press, 2019.

[3] S Kevin Zhou, Daniel Rueckert, and Gabor Fichtinger. *Handbook of medical image computing and computer assisted intervention*. Academic Press, 2019.

[4] Sanketh Kini. *Convolution Visualization*. Available from: `https://medium.com/@kinisanketh/getting-started-with-cnn-18c03efc7d06` [Accessed 1 June 2020].

[5] Muneeb ul Hassan. *VGG16*. Available from: `https://neurohive.io/en/popular-networks/vgg16/` [Accessed 1 June 2020].

[6] Adrian Rosebrock. *Imagenet with Keras*. Available from: https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/ [Accessed 1 June 2020].

[7] Royal College of Pathologists (UK). *Histopathology*. Available from: `https://www.rcpath.org/discover-pathology/news/fact-sheets/histopathology.html` [Accessed 1 June 2020].

[8] Kun-Hsing Yu, Ce Zhang, Gerald J Berry, Russ B Altman, Christopher Ré, Daniel L Rubin, and Michael Snyder. Predicting non-small cell lung cancer prognosis by fully automated microscopic pathology image features. *Nature communications*, 7:12474, 2016.

[9] Alexander JJ Smits, J Alain Kummer, Peter C De Bruin, Mijke Bol, Jan G Van Den Tweel, Kees A Seldenrijk, Stefan M Willems, G Johan A Offerhaus, Roel A De Weger, Paul J Van Diest, et al. The estimation of tumor cell percentage for molecular testing by pathologists is not accurate. *Modern Pathology*, 27(2):168–174, 2014.

[10] D Ross Camidge, Mariana Theodoro, DeLee A Maxson, Margaret Skokan, Tara O'Brien, Xian Lu, Robert C Doebele, Anna E Barón, and Marileila

Varella-Garcia. Correlations between the percentage of tumor cells showing an anaplastic lymphoma kinase (alk) gene rearrangement, alk signal copy number, and response to crizotinib therapy in alk fluorescence in situ hybridization–positive nonsmall cell lung cancer. *Cancer*, 118(18):4486–4494, 2012.

[11] National Cancer Institute (US). *Metastases*. Available from: `https://www.cancer.gov/types/metastatic-cancer` [Accessed 1 June 2020].

[12] Nicolas Coudray, Paolo Santiago Ocampo, Theodore Sakellaropoulos, Navneet Narula, Matija Snuderl, David Fenyö, Andre L Moreira, Narges Razavian, and Aristotelis Tsirigos. Classification and mutation prediction from non–small cell lung cancer histopathology images using deep learning. *Nature medicine*, 24(10):1559–1567, 2018.

[13] Erick Moen, Dylan Bannon, Takamasa Kudo, William Graf, Markus Covert, and David Van Valen. Deep learning for cellular image analysis. *Nature methods*, pages 1–14, 2019.

[14] Princeton University Stanford University. *ImageNet*. Available from: `http://www.image-net.org/` [Accessed 1 June 2020].

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[16] Scott Mayer McKinney, Marcin Sieniek, Varun Godbole, Jonathan Godwin, Natasha Antropova, Hutan Ashrafian, Trevor Back, Mary Chesus, Greg C Corrado, Ara Darzi, et al. International evaluation of an ai system for breast cancer screening. *Nature*, 577(7788):89–94, 2020.

[17] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *nature*, 542(7639):115–118, 2017.

[18] Songtao Guo and Zhouwang Yang. Multi-channel-resnet: An integration framework towards skin lesion analysis. *Informatics in Medicine Unlocked*, 12:67–74, 2018.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[20] Anne E Carpenter, Thouis R Jones, Michael R Lamprecht, Colin Clarke, In Han Kang, Ola Friman, David A Guertin, Joo Han Chang, Robert A Lindquist, Jason Moffat, et al. Cellprofiler: image analysis software for identifying and quantifying cell phenotypes. *Genome biology*, 7(10):R100, 2006.

[21] US National Cancer Institute. *The Cancer Genome Atlas*. Available from: `https://www.cancer.gov/about-nci/organization/ccg/research/structural-genomics/tcga` [Accessed 1 June 2020].

[22] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[23] Runyu Hong, Wenke Liu, Deborah DeLair, Narges Razavian, and David Fenyö. Predicting endometrial cancer subtypes and molecular features from histopathology images using multi-resolution deep learning models. *bioRxiv*, 2020.

[24] Faisal Mahmood, Daniel Borders, Richard Chen, Gregory N McKay, Kevan J Salimian, Alexander Baras, and Nicholas J Durr. Deep adversarial training for multi-organ nuclei segmentation in histopathology images. *IEEE transactions on medical imaging*, 2019.

[25] Thorsten Falk, Dominic Mai, Robert Bensch, Özgün Çiçek, Ahmed Abdulkadir, Yassine Marrakchi, Anton Böhm, Jan Deubner, Zoe Jäckel, Katharina Seiwald, et al. U-net: deep learning for cell counting, detection, and morphometry. *Nature methods*, 16(1):67–70, 2019.

[26] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[27] Sara Hosseinzadeh Kassani, Peyman Hosseinzadeh Kassani, Michal J Wesolowski, Kevin A Schneider, and Ralph Deters. Breast cancer diagnosis with transfer learning and global pooling. *arXiv preprint arXiv:1909.11839*, 2019.

[28] Shujiao Sun, Bonan Jiang, Yushan Zheng, and Fengying Xie. A comparative study of cnn and fcn for histopathology whole slide image analysis. In *International Conference on Image and Graphics*, pages 558–567. Springer, 2019.

[29] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[30] Norman R Draper and Harry Smith. *Applied regression analysis*, volume 326. John Wiley & Sons, 1998.

[31] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[32] Hrushikesh N Mhaskar and Tomaso Poggio. Deep vs. shallow networks: An approximation theory perspective. *Analysis and Applications*, 14(06):829–848, 2016.

[33] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

[34] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[35] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[36] Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256, 2010.

[37] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[38] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[39] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[40] R. Bellman, Rand Corporation, and Karreman Mathematics Research Collection. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957. ISBN 9780691079516. URL `https://books.google.se/books?id=wdtoPwAACAAJ`.

[41] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[42] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in neural information processing systems*, pages 8778–8788, 2018.

[43] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[44] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[45] T Björk. Arbitrage theory in continuous time, 2009.

[46] Tomaso Poggio, Hrushikesh Mhaskar, Lorenzo Rosasco, Brando Miranda, and Qianli Liao. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. *International Journal of Automation and Computing*, 14(5):503–519, 2017.

[47] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[48] Augustin Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[49] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[50] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[51] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[52] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks, 2016.

[53] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[54] Peter Bankhead, Maurice B Loughrey, José A Fernández, Yvonne Dombrowski, Darragh G McArt, Philip D Dunne, Stephen McQuaid, Ronan T Gray, Liam J Murray, Helen G Coleman, et al. Qupath: Open source software for digital pathology image analysis. *Scientific reports*, 7(1):1–7, 2017.

[55] Python. *Python3.0*. Available from: `https://www.python.org/download/releases/3.0/` [Accessed 1 June 2020].

[56] François Chollet et al. *Keras library*. Available from: `https://keras.io` [Accessed 1 June 2020].

[57] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from `https://www.tensorflow.org/` [Accessed 1 June 2020].

[58] Nvidia. *CUDA library*. Available from: `https://developer.nvidia.com/cuda-toolkit` [Accessed 1 June 2020].

[59] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24574-4.

[60] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

Appendix A

Architectures

In this chapter the graphs of the three models used is shown. The question mark in the layers represents the unknown number of samples.

Figure (A.1) shows the layout of the implementation of VGG16 for the case of 448x448 tiles.

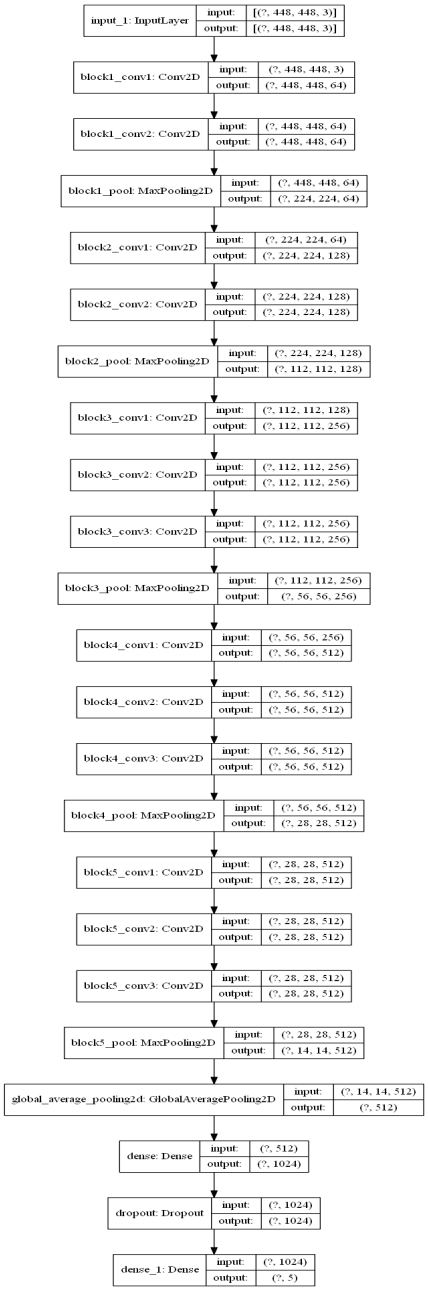


Figure A.1: Visualization of the implementation of VGG16 used

Figure (A.2) shows the layout of the implementation of "compactVGG" for the case of 448x448 tiles.

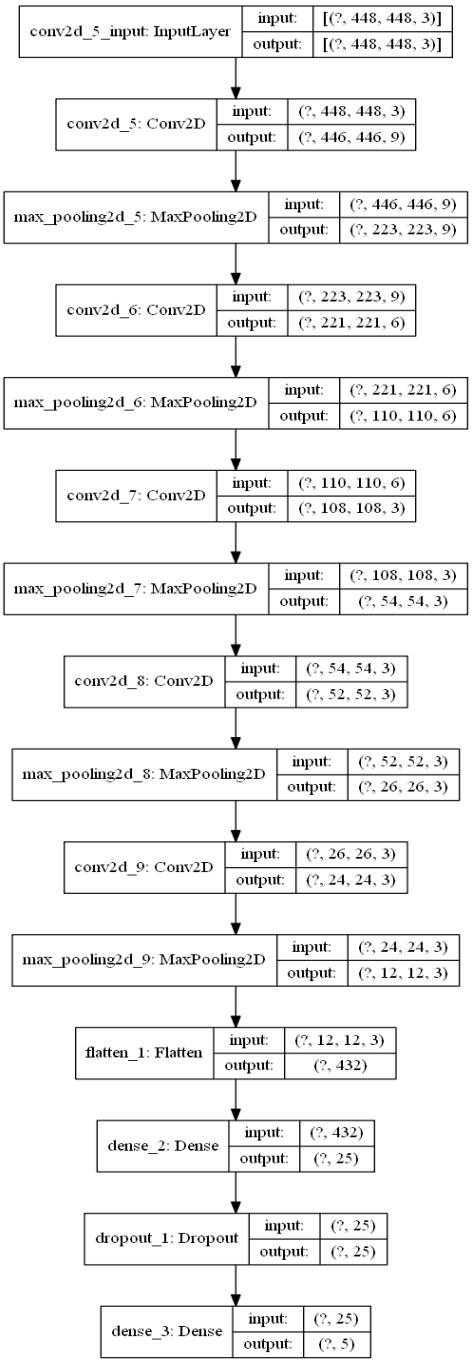


Figure A.2: Visualization of "compactVGG"

Figures (A.3, A.4, A.5) shows the layout of the implementation of InceptionV3 for the case of 448x448 tiles. The graphs are connected from the bottom of the left image to the top of the right in each Figure. And the bottom of the right image, is connected to the top of the left image, in the succeeding Figure.

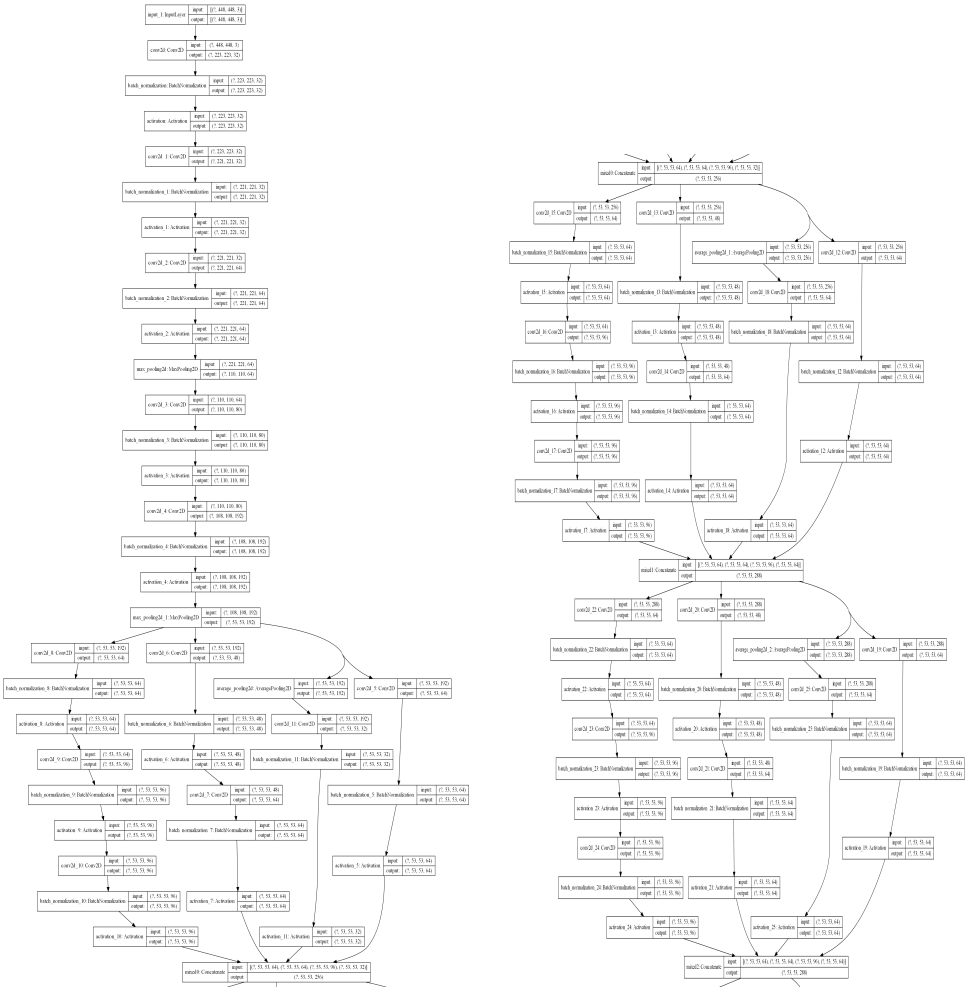


Figure A.3: Visualization of the first layers of the implementation of inceptionV3 used

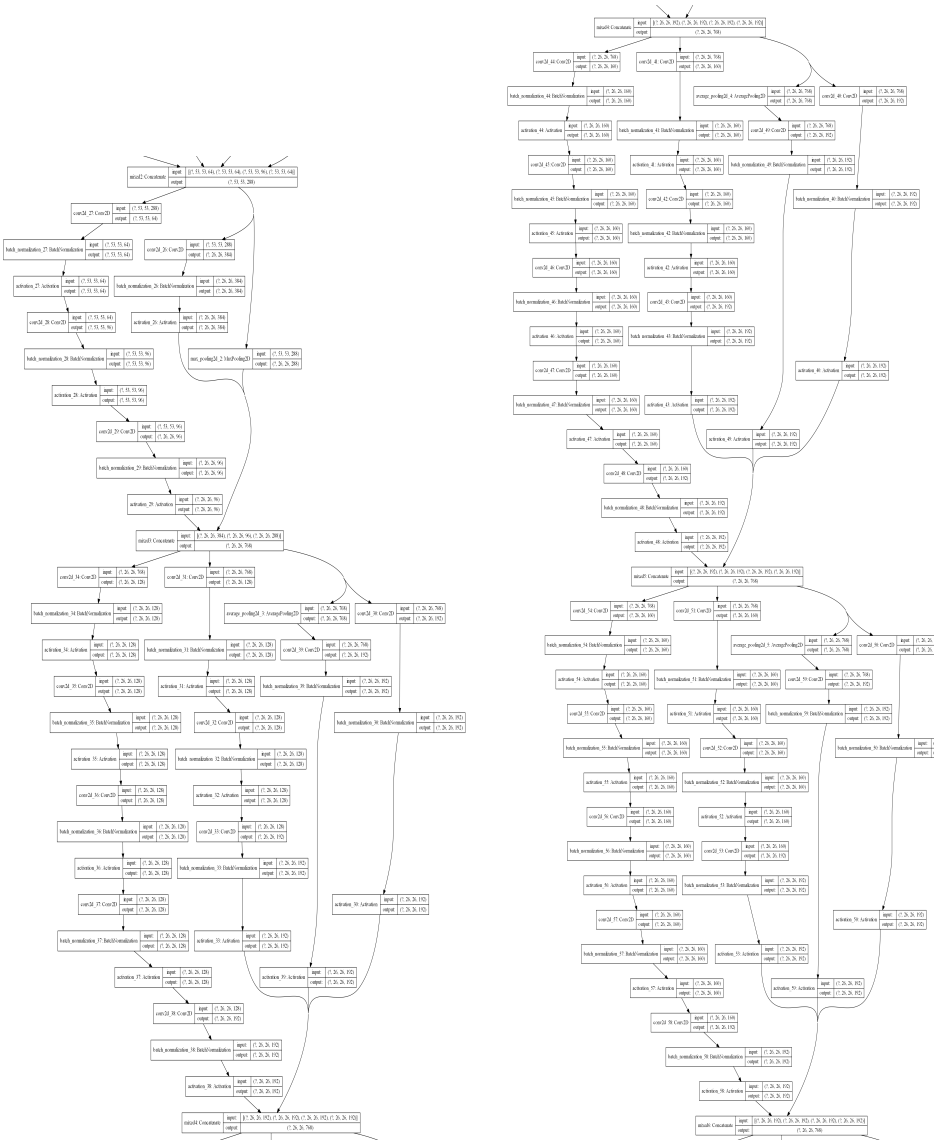


Figure A.4: Visualization of the middle layers of the implementation of inceptionV3 used

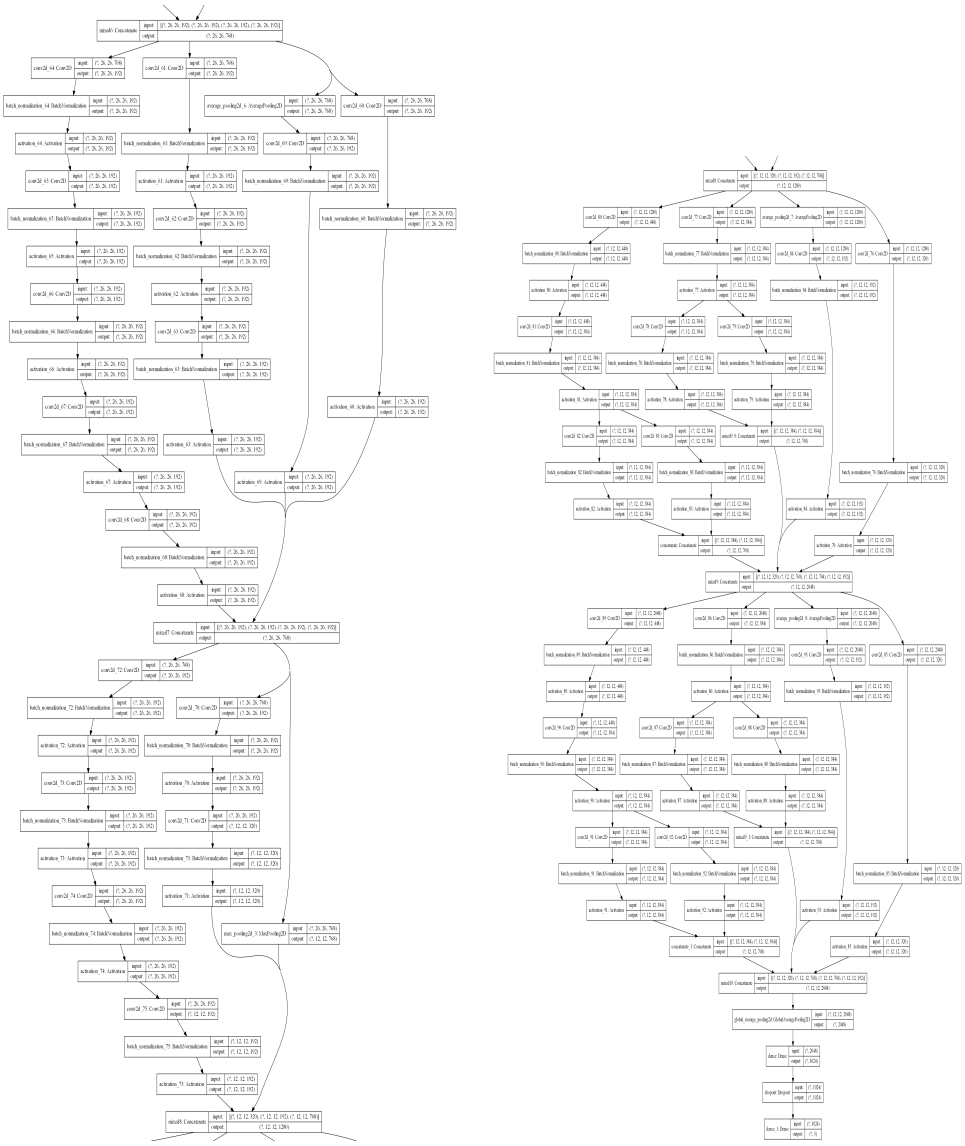


Figure A.5: Visualization of the last layers of the implementation of inceptionV3 used

Appendix B
Results

B.1 Model Performance on Dataset I

In this section the test performance on the randomized test tiles, after removing all ambiguous tiles, from the Initial Dataset I. The data set which was used as an early evaluation of models and to test the code. Figure B.1 shows the confusion matrices and B.2 shows the ROC-curves.

Figure B.1: Confusion matrices, Dataset224_202005/Test

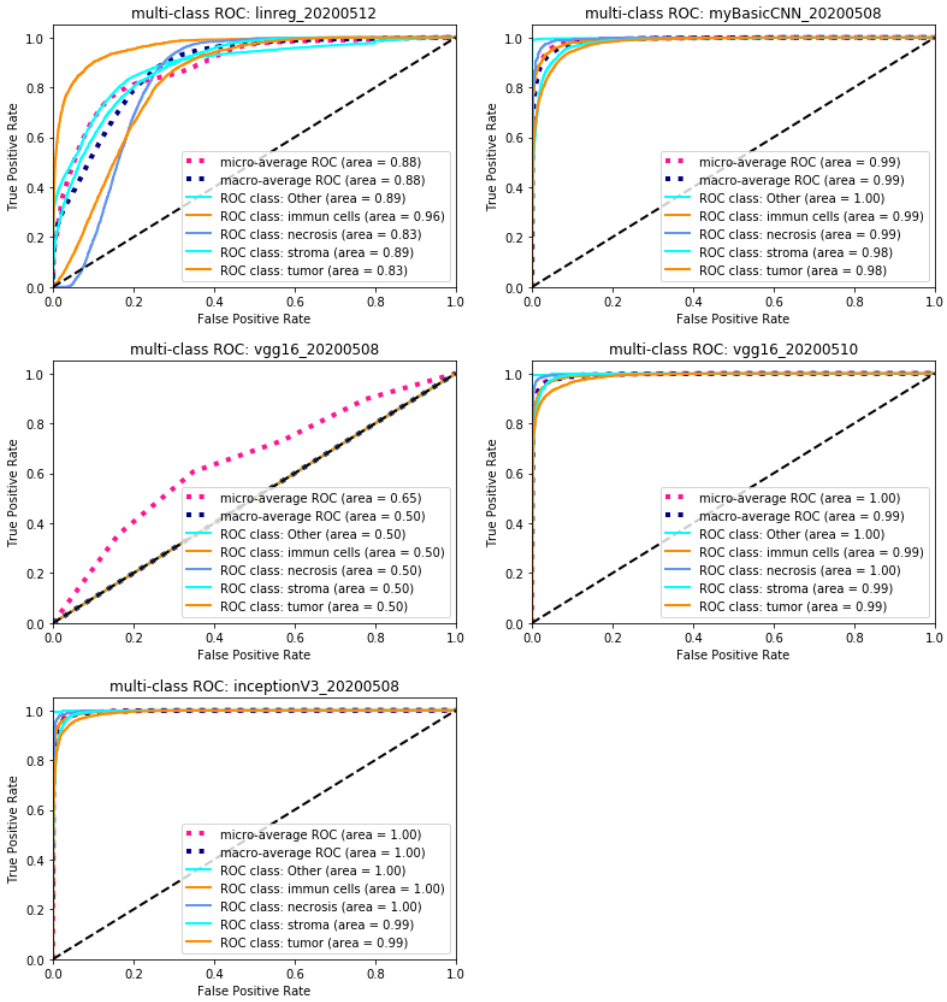
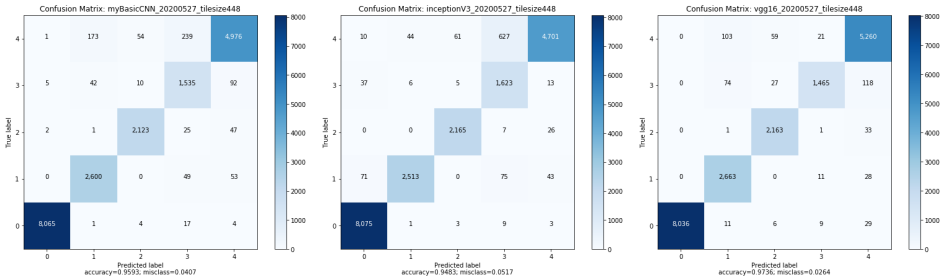


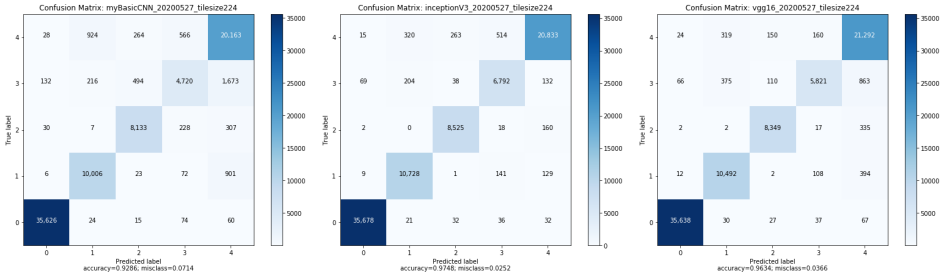
Figure B.2: Recieving Operating Characteristic (ROC) curves, Dataset I/Test

B.2 Model Performance on Dataset II

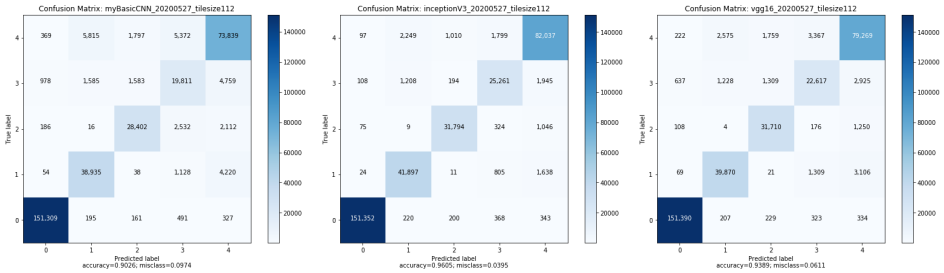
In this section the test performance on the randomized tiles, after removing all ambiguous tiles, from the Extended Dataset II, used for training the final networks, is presented.



(a) tilesize 448



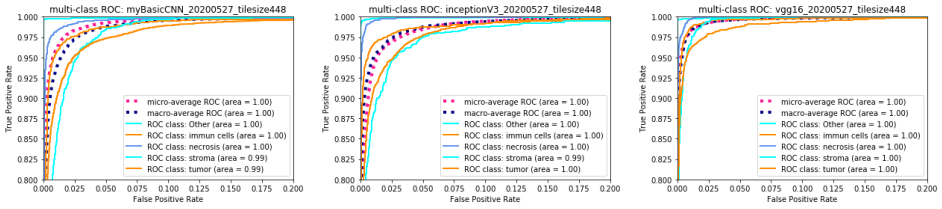
(b) tilesize 224



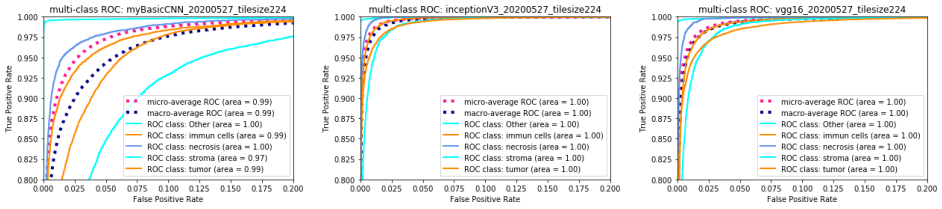
(c) tilesize 112

Figure B.3: Confusion Matrices, Dataset II/Test

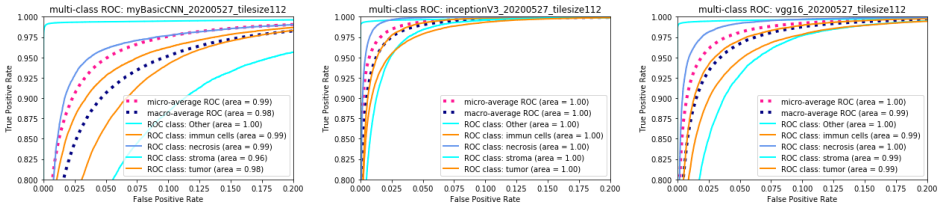
In Figure B.4 the resulting ROC-curves of the respective models are presented.



(a) tilesize 448



(b) tilesize 224



(c) tilesize 112

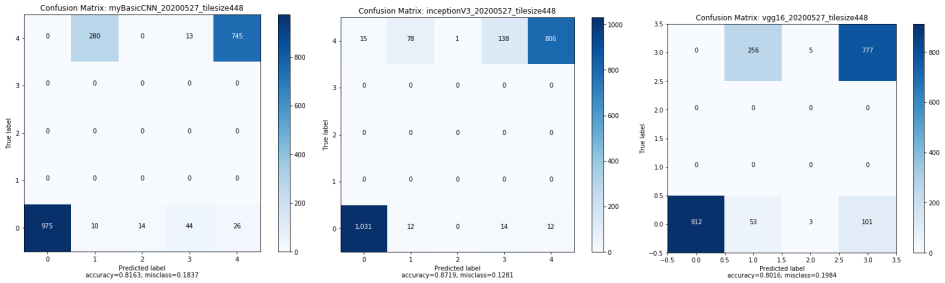
Figure B.4: Receiving Operating Characteristic (ROC) curves, Dataset II/Test

B.3 Model Performance on Dataset III

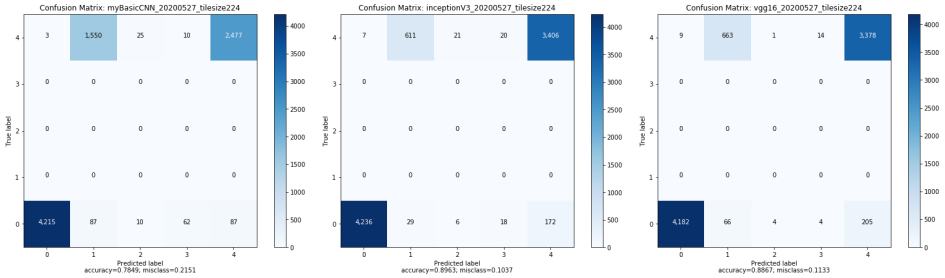
In this section the test performance on the whole-slides MM1279-1-12-23/testdata0 and MM812-19-25-31/testdata1 are presented (for the final models trained on Dataset II). The whole-slides are tiled and then the networks make a prediction for each tile. Note that ambiguous tiles with several different true annotated classes is not removed. Instead the dominating class (determined as described in section 3.1.1) is used.

B.3.1 testdata0

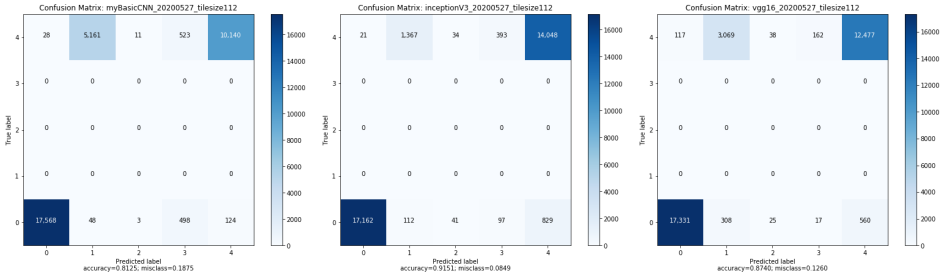
This section presents the models (trained on Dataset II) performance on the whole-slide MM1279-1-12-23/testdata0. In Figure B.5 the confusion matrices of the respective models are presented. Where 0:Other, 1:immune cells, 2:necrosis, 3:stroma, 4:tumor.



(a) tilesize 448



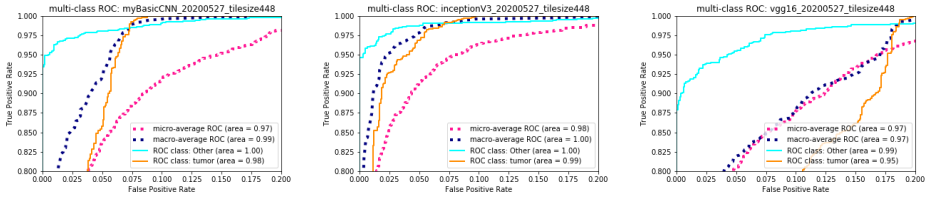
(b) tilesize 224



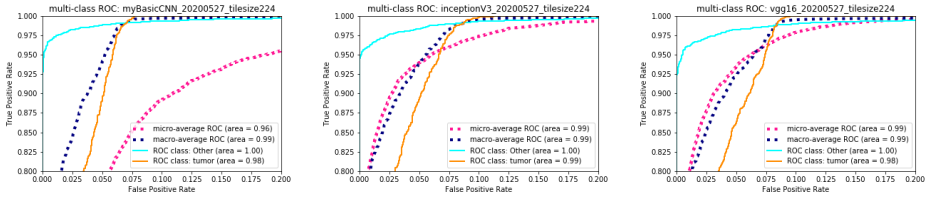
(c) tilesize 112

Figure B.5: Confusion Matrices, MM1279-1-12-23

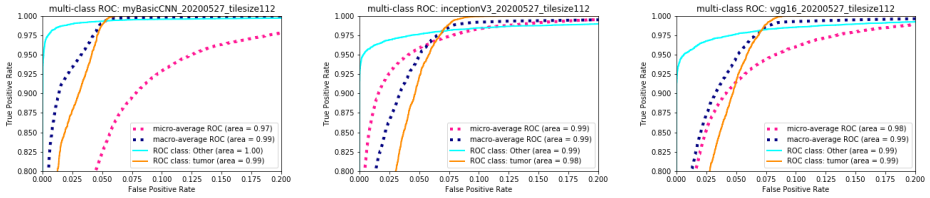
In Figure B.6 the resulting ROC-curves of the respective models are presented.



(a) tilesize 448



(b) tilesize 224

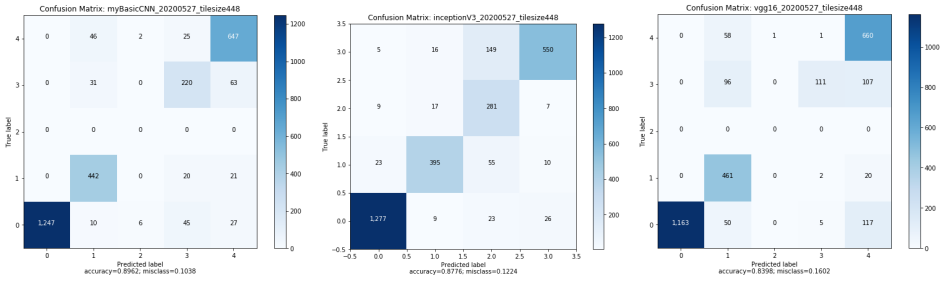


(c) tilesize 112

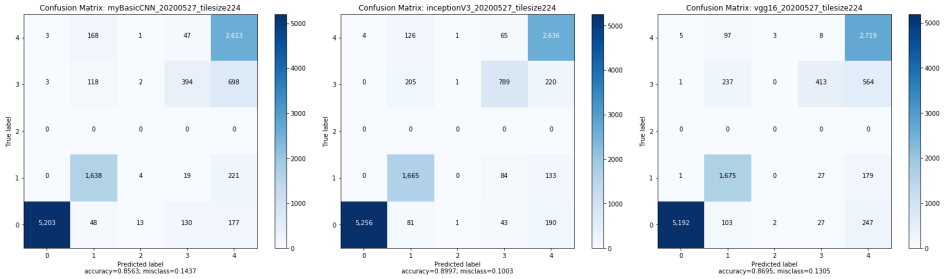
Figure B.6: Receiving Operating Characteristic (ROC) curves, MM1279-1-12-23

B.3.2 testdata1

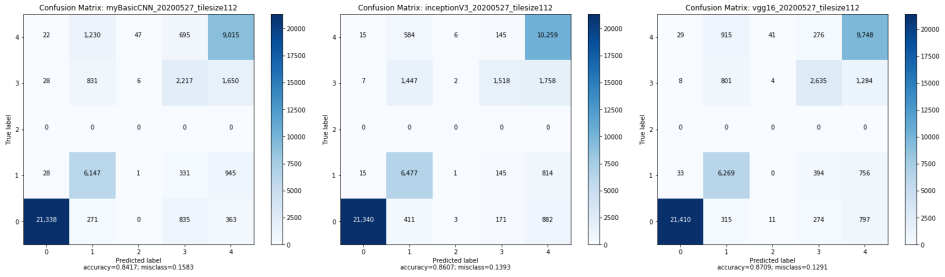
This section presents the models (trained on Dataset II) performance on the whole-slide MM812-19-25-31/testdata1. In Figure B.7 the confusion matrices of the respective models are presented.



(a) tilesize 448



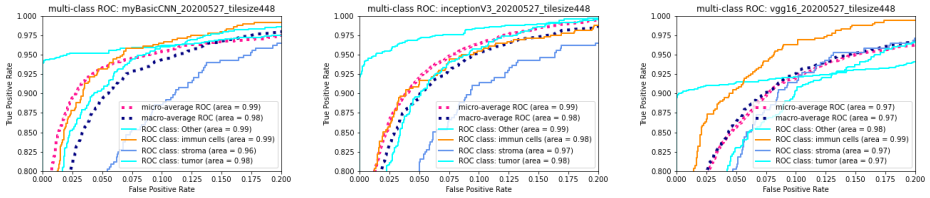
(b) tilesize 224



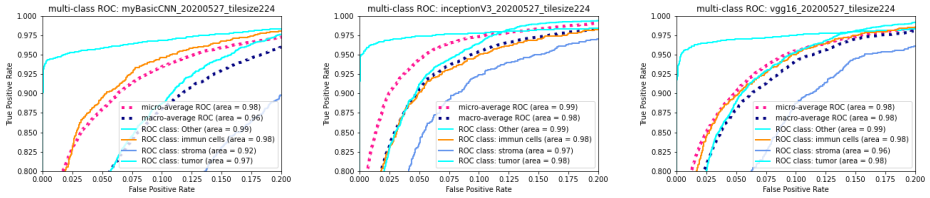
(c) tilesize 112

Figure B.7: Confusion Matrices on MM812-19-25-31

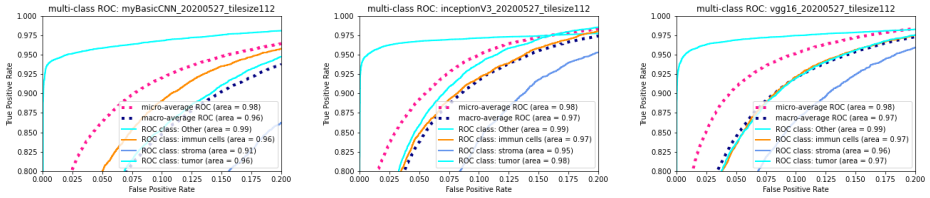
In Figure B.8 the resulting ROC-curves of the respective models are presented.



(a) tilesize 448



(b) tilesize 224



(c) tilesize 112

Figure B.8: Receiving Operating Characteristic (ROC) curves, MM812-19-25-31