CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Computer Science

Master's Thesis

Automated Detection and Quantitation of Langerhans Islets in Pancreatic Tissue

Bc. Jan Horák

May 2021
https://janhorak.info/pancreas
Supervisor: prof. Dr. Ing. Jan Kybic
Study programme: Bioinformatics

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Horák Jan** Personal ID number: **458177**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Bioinformatics**

II. Master's thesis details

Master's thesis title in English:

Automated Detection and Quantitation of Langerhans Islets in Pancreatic Tissue

Master's thesis title in Czech:

Automatická detekce a kvantifikace Langerhansových ostrůvků v pankreatické tkáni

Guidelines:

Histological structure, pancreatic damage and the number of langerhans islets are important markers of the severity of type 1 diabetes and various other diseases. Langerhans islet transplantation can be used as a treatment. Biopsy and identification of the number and functionality of islets are used to select a suitable donor. Automatic diagnostics will significantly speed up this process, ensuring constant accuracy and consistency of evaluation.
Scan image files of tissue sections stained with hematoxylin & eosin (H&E) are provided. Machine learning methods, especially deep learning, should be used to automatically find the region of interest in the scans, and detect and count Islet cells, acinar tissue, ductal areas, adipose and blood vessels within these areas.
1. Describe the existing methods for the analysis of histological scans and algorithms for the detection and segmentation of areas of interest and objects in these images.
2. Examine the source microscopy data, prepare suitable tools for annotating the data and preprocessing and prepare the dataset for training.
3. Examine various machine learning methods for pancreatic tissue analysis, segmentation and counting islet cells. Compare their accuracy and usability.
4. Design and implement the scan image analysis pipeline solving the given tasks, and evaluate it experimentally.
5. Create a simple web-application for users to easily load, process and evaluate data, provide scripts/interface to import/export data to/from the QuPath software.

Bibliography / sources:

Huang Y, Liu C, Eisses JF, Husain SZ, Rohde GK. A supervised learning framework for pancreatic islet segmentation with multi-scale color-texture features and rolling guidance filters. Cytometry A. 2016;89(10):893–902. doi:10.1002/cyto.a.22929
Chen H, Martin B, Cai H, et al. Pancreas++: automated quantification of pancreatic islet cells in microscopy images. Front Physiol. 2013;3:482. 2013 Jan 3. doi:10.3389/fphys.2012.00482
Habart David, Švihlík Jan, Schier Jan, Cahová Monika, Girman Peter, Zacharovová Klára, Berková Zuzana, Kříž Jan, Fabryová Eva, Kosinová Lucie, Papáčková Zuzana, Kybic Jan and Saudek František. Automated Analysis of Microscopic Images of Isolated Pancreatic Islets. Cell Transplantation, no. 12, pp. 2145-2156, December 2016.
Bankhead, P. et al., QuPath: Open source software for digital pathology image analysis. Scientific Reports (2017). https://doi.org/10.1038/s41598-017-17204-5

Name and workplace of master's thesis supervisor:

prof. Dr. Ing. Jan Kybic, Biomedical imaging algorithms, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **25.05.2021** Deadline for master's thesis submission: **21.05.2021**

Assignment valid until: **19.02.2023**

prof. Dr. Ing. Jan Kybic Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgement / Declaration

I would first like to thank my thesis advisor prof. Dr. Ing. Jan Kybic for his expert advice and encouragement throughout this thesis. Next, many thanks to Ondrej Šantavý, for helping me to correctly annotate pancreas images for training the deep neural network and Jan Mikula for testing the web application and giving me feedback throughout the development. Big thanks goes to Mr. Bulldops for helping me to get schwifty while writing this thesis.

Finally, I must express my very profound gratitude to my parents, sister and to my girlfriend for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them. Thank you.

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 21, 2021

.....

Abstrakt / Abstract

Pro určení správné diagnostiky onemocnění nebo studia jejich projevů, výzkumu nových lečebných metod a pozorování změn ve tkáni histologických vzorků je nutné ručně identifikovat jednotlivé části tkáně, zjistit zastoupení, nebo vzájemnou polohu v tkáni. Ruční segmentace histologických snímků může být velmi časově náročná a neúčinná, či zkreslená subjektivním biasem. Nástup technologických průlomů v oblasti automatického rozpoznávání obrazu umožnilo rychlejší a přesnější vyhodnocení s porovnání s člověkem. Využití automatická segmentace tkáně může v i tomto ohledu výrazně přispět. Mnoho nástrojů na zpracování medicínských obrazů a dat jsou navíc většinou proprietární, nebo jednoúčelové, či zastarelé a bez jednoduchého uživatelského rozhraní, znesnadňující jejich použití a přizpůsobení. V této diplomové práci jsem se rozhodl přiblížit řešení tohoto problému implementací state-of-art metod umělé intelegence v oblasti počítačového vidění na segmentaci histologických snímků pankreasu a detekci Langerhansových ostrůvků. Cílem bylo také demonstrovat jednoduchost použití tohoto nástroje pomocí webové aplikace pro snadné použití uživatelem, histologických expertem.

Klíčová slova: buňky, slinivka břišní, tkáň, Langerhansovy ostrůvky, segmentace, počítačové vidění, neuronové sítě, hluboké neuronové sítě, detekce objektů, předzpracování dat, Python

Překlad titulu: Automatická detekce a kvantifikace Langerhansových ostrůvků v pankreatické tkáni

To determine the correct diagnosis of the disease or study of its manifestations, research of new treatment methods and observation of changes in the tissue of histological samples, it is necessary to manually identify individual sections of the tissue, determine the representation or mutual position of the sections withing the tissue. Manual segmentation of histological images can be very time consuming and inefficient, or distorted by subjective bias. The advent of technological breakthroughs in the field of automatic image recognition has enabled faster and more accurate evaluation compared to human. The use of automatic tissue segmentation can make a significant contribution in this regard as well. In addition, many medical image and data processing tools are mostly proprietary, or single-purpose, or obsolete, and without a simple user interface, making them difficult to use and customize. In this diploma thesis, I decided to approach the solution of this problem by implementing state-of-art methods of artificial intelligence in the field of computer vision for the segmentation of the pancreas histological images and the Langerhans islets detection. The goal was also to demonstrate the ease of use of this tool by developing an easy-to-use web application for the end user - a histological expert.

Keywords: cells, pancreas, tissue, Langerhans islets, segmentation, computer vision, neural networks, deep neural networks, object detection, data preprocessing, Python

Contents /

Tables / Figures

Chapter 1
Introduction

1.1 Motivation

People with type 1 diabetes mellitus (T1M) do not produce their own insulin, an essential hormone that lowers blood sugar. Insulin is produced in the β-cells of the islets of Langerhans, which are dispersed in the pancreatic tissue. Type 1 diabetes mellitus is characterized by autoimmune destruction of β-beta cells and the complete or almost complete absence of insulin, which must then be administered by lifetime by injection or subcutaneous infusion. Its exact dosing according to the current needs of the patient is very difficult, as a result of which the blood sugar level easily exceeds normal values or, conversely, falls into the area of hypoglycemia. [1–2]

Lack of insulin secretion can also be replaced by transplantation of insulin-producing tissues. Currently, two basic options are used, namely transplantation of the entire pancreas or only isolated islets of Langerhans from the endocrine gland, which represents only about 1-2% of the total pancreatic tissue. Transplantation of the entire pancreas is a relatively difficult surgical procedure due to the high number of risks and possible complications, and therefore in many cases today, transplantation of islets by less invasive methods is preferred. [2]

Although islet transplantation is still considered an experimental treatment, many studies have shown that over the years, thanks to improved islet extraction and isolation techniques and proper donor selection, a transplant patient may not have to use insulin injections for one year, or even longer.

To speed up and refine research in the field of islet transplantation but also pancreatic diseases in general, it is desirable to use modern procedures and computer technology in observing changes in histological tissue, comparing samples over time and also with each other. [3]

Manual histology image segmentation can be very time consuming and ineffective and automatic tissue segmentation can hugely impact both speed and precision in this process. There are not many easy to use tools for customizable, yet precise automatic analysis of the histology (pancreatic) images. Some open source programs and tools are available, but their accuracy is far from the quality that could be used in clinical research. Therefore, I decided to help contribute to solving this problem using a custom solution with state-of-the art methods of AI when segmentating the pancreatic tissue.

1.2 Goals

(i) Describe the existing methods for the analysis of histological scans and algorithms for the detection and segmentation of areas of interest and objects in these images.
(ii) Examine the source microscopy data, prepare suitable tools for annotating the data and preprocessing and prepare the dataset for training.
(iii) Design and implement the scan image analysis pipeline solving the given tasks, and evaluate it experimentally.
(iv) Create a web-application for users to easily load and segment new scan images. Create a viewing module for displaying whole-slide images with aligned overlay layers of the output segmentation.

Chapter 2
Research

2.1 Machine Learning

Machine learning is a subclass of artificial intelligence based on the concept of extracting knowledge from observed states or experiences, followed by the application of knowledge to evaluate new observations.

Formally, a machine learning solution, often referred to as the M model, can be seen as an approximation of the probability distribution P, a random variable x, representing the problem being solved. The model is often implemented by a parametric differentiable function M_θ and the process of finding a solution for a given task is called *training* and consists in finding the optimal set of parameters θ, often referred to as *model weights*. [4]

Model training depends on the quality and amount of training data. The optimal set of parameters is searched using the maximum likelihood estimate (MLE) principle so that the distribution of model $P_M\theta$ matches the probability distribution of random variable x in the best possible way. This process is implemented to minimize the so called *loss function*.

Machine learning tasks can be divided into two categories according to the properties of the training data - *supervised* and *unsupervised*. Data in the case of supervised learning have the form (x, y), where x is an input and y i a desired output. The aim is to best approximate the conditional probability distribution $P(y|x)$. [5]

In the case of unsupervised learning, the data take the form of singletons x, i.e. samples of unknown distribution. The goal is usually to find an unknown structure (clustering), be able to generate new samples, i.e. the approximate probability of distribution $P(x)$, or transform the data into some latent space with useful properties. [5, 4]

2.2 Neural Networks

Neuron (or Neural) network (NN) is a computer model heavily inspired by the human brain and nervous system. A neural network is defined as a set of neurons and the connections between them. Neurons are computing units which are interconnected by weighted connections weights. The basic model of a neural network is unidirectional graph, meaning a signal from one neuron enters several other neurons in one way. The learning process of a neural network takes place in such a way that we give the network some sample input data - called *training data* (representing for example an image, sound, text, etc.), for which we want to get an evaluation at the output, again in the form of arbitrary data - image, text, etc.). During learning, the neural network automatically optimizes the weights between the neurons so that

the output with the smallest possible error coincides with our desired ground truth output - called *label*. Multilayer neural networks has one or more *hidden layers* which reside in-between input and output layers and they perform nonlinear transformations of the inputs entered into the network. Illustration of a multilayer neural network can be seen in the following Figure 2.1. [6]

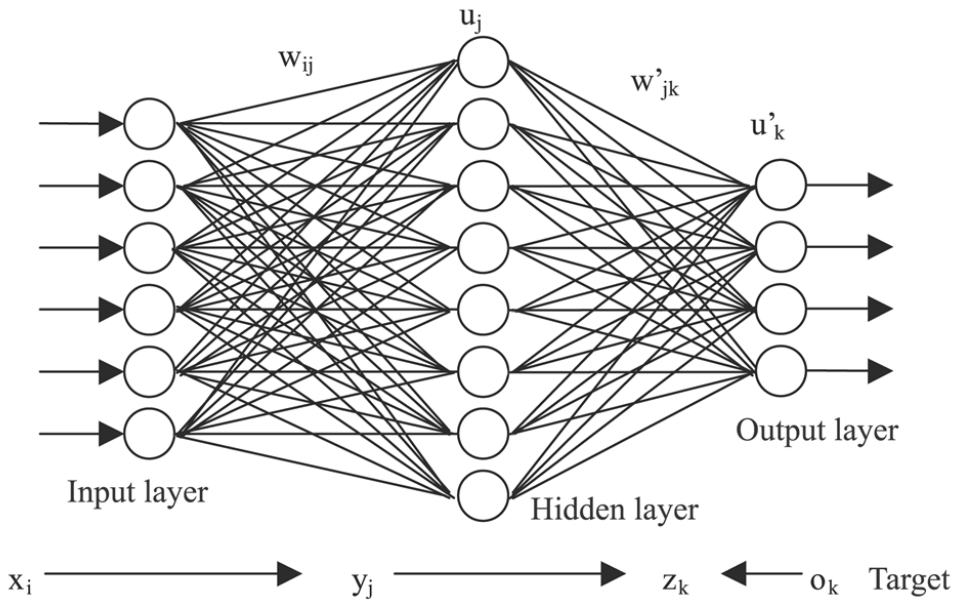


Figure 2.1. Example of Neural Network with hidden layers. Source: extremetech.com[1].

2.3 Single Neuron - Perceptron

The basic unit in a neural network is one neuron. The *perceptron* is then an algorithm (neural network) with a single such neuron - the perceptron. This artificial neuron is remotely inspired by natural neurons. It has several inputs x_i and one output. Each of the inputs is assigned a weight w_i. The perceptron first calculates its activation as a weighted sum of the inputs $\Phi = b + \sum_{i=1}^{n} w_i x_i$, where b is the so-called threshold (*bias*) and affects how much the weighted sum must be greater than 0 for the perceptron to activate. If this activation is greater than threshold 0, the output of the perceptron is 1, otherwise it is 0. [7]

2.4 Convolutional Neural Networks

A Convolutional Neural Network (CNN or ConvNet) is a subclass of ordinary NN, designed with assumption that the inputs are images, allowing us to encode certain properties into the architecture. The problem with regular Neural Networks is that they don't scale well to images, because even for a low resolution images, when using fully-connected structure of layers, we would have a very big numbers of weights/parameters, which would quickly lead to *overfitting* (explained in the following section) [8]. As there is a big

[1] http://www.extremetech.com/wp-content/uploads/2015/07/NeuralNetwork.png

overlap with introduction of CNN also in my previous Bachelor's thesis [9], I derive some of the following information from it.

There are four main operations in the CNN:

- Convolution
- Non Linearity (ReLU)
- Pooling
- Classification (Fully Connected Layers)

These operations are the basic building blocks of every Convolutional Neural Network.

2.5 Convolution

The primary purpose of Convolution in case of a CNN is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

Every image is considered by a computer as an matrix of numbers (where numbers specify pixel color intensity). We create a Convolution Layers by convolving or `sliding` a filter (sometimes referred to as a kernel) by N pixels (also called stride) across the input image, where the current region bellow the filter is called receptive field, and multiplying the the values in the filter with the original pixel values of the image, thus computing element wise multiplications. We add the multiplication outputs to get the final integer which forms a single element of the output matrix. The final output matrix is called Convolved Image, Activation Map or Feature Map. [10]

As an example, considering an I as an input image, matrix K as a filter/kernel of size $h \times w$, we can compute the Convolved Image $I * K$ as

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} * I_{x+i-1,y+j-1} \tag{1}$$

which can be illustrated with Figure 2.2.

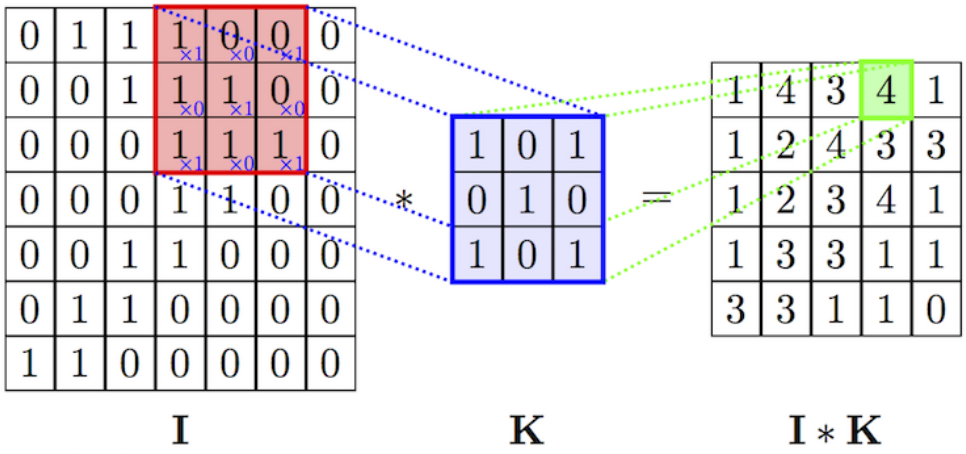


Figure 2.2. Overview of applying convolution. Source: [11].

2.5.1 Pooling Layer

Pooling layer is another essential block of well-performing CNN architecture. Pooling is also referred to as a downsampling. Several options exists (*Average Pooling*, *L2-Norm Pooling*), but the one being the most used and popular is *Max-Pooling*. Max-Pooling is an operation, where a filter convolves around each subregion (again with specified stride), and outputs the maximum number of this subregion to the output matrix. To better illustrate this process, see Figure 2.3.

Pooling layers are used in part to reduce *overfitting* by providing an abstracted form of the representation. It also reduces the computational cost by reducing number of parameters to learn. Lastly, it provides basic translation invariance to the internal interpretation, which is the reason some networks, where object specific positions are needed, don't use them at all. [11]

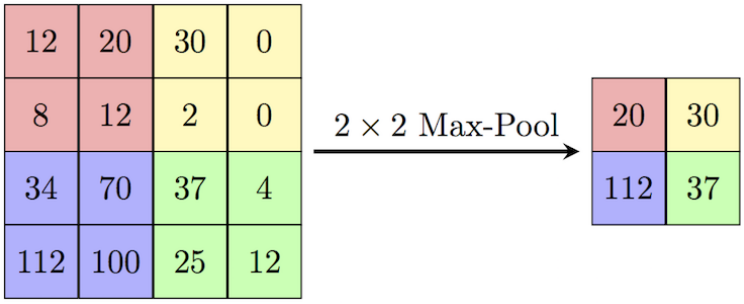


Figure 2.3. Max-Pooling with filter of size 2×2. Source: [11]

2.5.2 CNN Architecture and Training

Convolution Neural Network is commonly made up of only three layer types: *Convolution* (CONV), *Pooling* (POOL), and *Fully Connected* (FC) layers, where CONV and POOL layers are ordinarily repeated several times to create a Deep Neural Network and extract high-level features.

A Fully Connected layer is a normal Multi Layer Perceptron that uses a *softmax* activation function in the output layer.

The whole architecture is then trained by updating and adjusting filters/weights in the Neural Network though a training process called *backpropagation* in the similar way as with a normal NN. [10]

2.5.3 Pre-trained Networks

Learning the whole neural network is also very time consuming for powerful computers. Such learning takes days, sometimes weeks, for really large networks. Therefore, in practice, in order to reduce the time required to train the network, a partially pre-learned network is used, which only needs to be fine-tuned to our required data. Here is an example of several popular network architectures that are used as so-called *backbones* in many other architectures.

- *AlexNet* - the first network to become more popular with CNN in 2012 in the field of computer vision. The network has a similar architecture as LeNet, but has a deeper and larger convolution part (one CONV layer is immediately followed by a POOL layer). [12]
- *GoogLeNet* - a network from 2014. They dramatically reduced the number of parameters (to 4 million, compared to 60 miles from AlexNet). [13]

- *VGGNet* - have shown that network depth has critical parameters for good performance. It always uses the same convolution mask. 3x3 and a 2x2 mask for grouping. [14]
- *ResNet* - contains a special skip connection. It does not have a fully interconnected output layer. [15]

2.5.4 Underfitting and Overfitting

Not only when using Neural Networks, but when training any model with machine learning methods, it is important to make sure that the resulting model is not underfit or overfit to the data.

- *Variance* - refers to how much a model changes in response to the training data.
- *Bias* - refers to how much a model ignore the data.

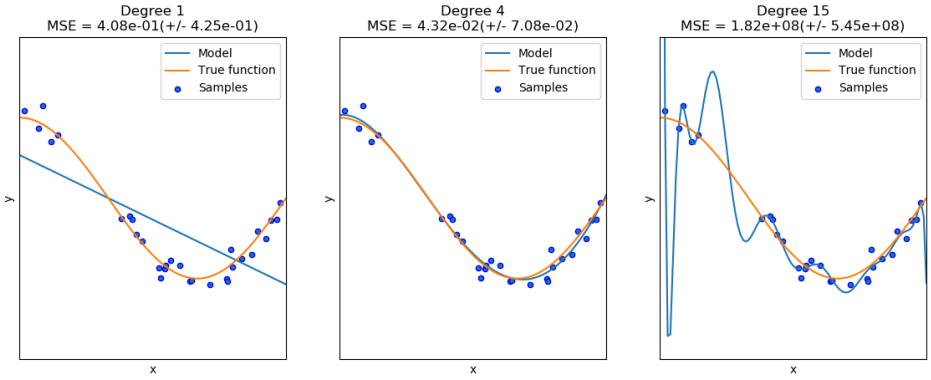


Figure 2.4. Underfitting (left), Good fit (middle), Overfitting (right). Source: [16]

Finding a good balance of *bias* vs *variance* is a critical concept in any Machine Learning modeling. *Underfitted* model has low variance and high bias, and thus the model is generalizing too much and fails to learn the underlying relationship between inputs and outputs.

In case of *overfitting*, model having high variance and low bias, results the model is too much relying on the training data and may therefore fail to fit additional data or predict future observations reliably.

Typically the dataset is being split into three parts:

- Training set.
- Validation set.
- Testing set.

When training the Neural Network, or any other Machine Learning model, we feed the model with data from training set, to update the parameters (weights in case of NN) and preform a *cross-validation*[1] with data from *validation set* to compare the performances of the prediction what were created on the training set. After we finish training we perform a prediction on our *test set* in order to see the accuracy on unseen data. Both overfitting and underfitting cause poor generalization on the *test set*. [17]

[1] `https://www.cs.cmu.edu/~schneide/tut5/node42.html`

2.5.5 Normalization

Normalization, known as feature scaling can be an important preprocessing step for many machine learning algorithms. By normalizing our data we put them on the same scale, which can significantly improve the ability for model to learn if the scales for different features are very different.

2.5.6 Standardization

An effective normalization technique is *standardization*, involving scaling the values of each feature in the data to have zero-mean and unit-variance:

$$x' = \frac{x - \bar{x}}{\sigma} \tag{2}$$

where x is the original feature vector, $\bar{x}$ is the mean of that feature vector, and σ is its standard deviation.

2.6 Computer Vision Tasks

In computer vision problem solving we typically deal with the following cases:

- Classification
- Detection
- Semantic Segmentation
- Instance Segmentation

When trying to solve any computer vision related tasks, we must first define what exactly we try to achieve, for choosing or implementing a suitable algorithm. Do we need just to classify a single object within the image? Do we need a semantic segmentation or instance segmentation? To better understand differences between these approaches let us look at figure 2.5.

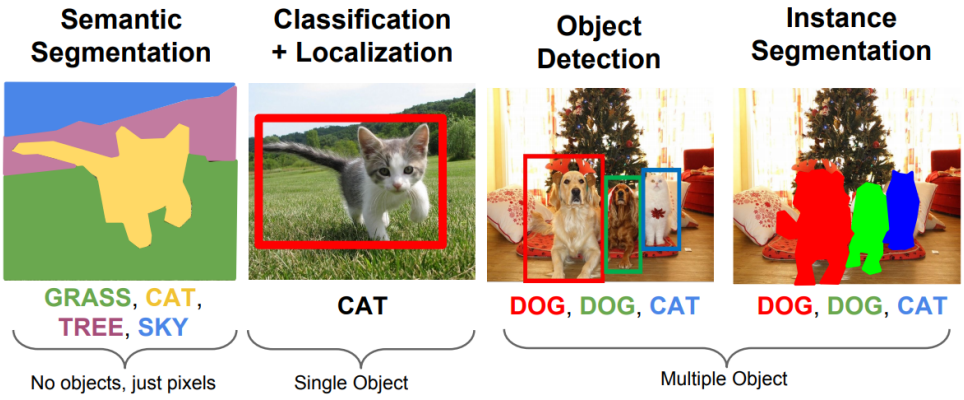


Figure 2.5. Computer Vision Tasks. Source: cs231n.stanford.edu[1].

2.7 Segmentation

Image segmentation is the process of dividing parts of an image into individual parts, which ideally form meaningful parts. The simplest case of segmentation

[1] http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

can be, for example, the separation of an object from the background. In the case of portrait photography, we usually want to isolate a person's face from the background. This would be a two-class segmentation and its output would be a binary image: the face area white, the background area black. In the case of histological scans of the pancreas, we may want to distinguish where the acinus, fat cells, blood vessels and the islets of Langerhans are located. In this case, we are talking about multi-class segmentation. This can result in a multi-color image, where each color represents just one of the selected classes. [18]

2.7.1 Thresholding

Thresholding is one of the oldest and simplest segmentation method. Despite the restricted usability it is a widely used and popular method. It can be used both alone and as part of other more sophisticated methods. The popularity of the thresholding lies in its simplicity, which results in easy implementation and very low computational requirements. Thresholding is based on the idea that objects and backgrounds have different intensity level. Thresholding will split the image into two regions based on pixel intensity values by the threshold value T. Pixels with value intensities higher than this threshold, will be marked as the pixels of the object and all other pixels are considered as the background pixels. Let us say, we define we have an image represented as a 2D matrix, then function $g(x, y)$ returns the intensity at the given position and $f(x, y)$ represent the background or object mapping as following:

$$f(x,y) = \begin{cases} 1 & \text{if } g(x,y) > T \\ 0 & \text{if } g(x,y) \leq T \end{cases} \tag{3}$$

By applying function $f(x, y)$ over whole image, we will get a new 2D matrix with pixel values 1 representing the object and 0 the background.

If the threshold T is of the same value for the whole image we call the algorithm as global thresholding. Otherwise we call it a local thresholding.

2.7.2 Watershed segmentation

Watershed transformation can be included among region-based segmentation approaches. This morphological method of segmentation is based on an idea derived from geography. The image is understood as a terrain or topographic relief that is gradually flooded with water.

The basins are filled with water from the starting points (local minima of the image). In places where water from two different river basins could merge, so-called *dams* are created. The process of gradual flooding is stopped when we reach the highest point of the terrain (image maxima). The result is a picture divided into regions, individual river basins separated by dams - these are simply called *watersheds*. [19]

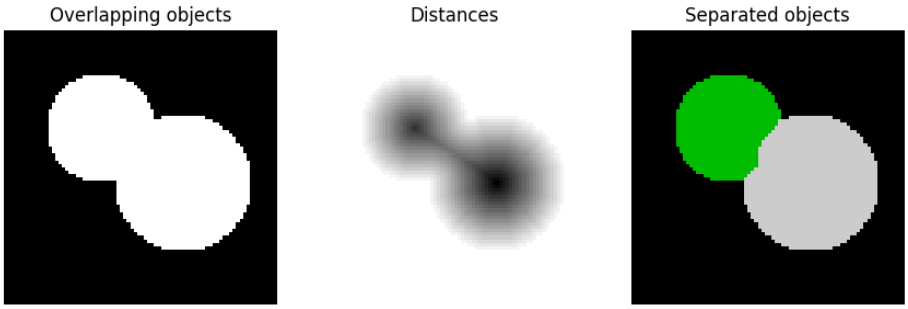


Figure 2.6. Watershred Algorithm. Source: scikit-image.org[1].

2.8 Semantic Segmentation

Semantic image segmentation is the understanding of an image at the pixel level, i.e. we want to assign a class of objects to each pixel. Semantic segmentation is used in many areas from multimodal medical image analysis to segmentation of multispectral satellite images.

It was not until 2014 that the CNN architecture for dense predictions without any fully interconnected layers was popularized. This made it possible to create segmentation maps for large-sized images. In addition to fully connected layers (FCN), one of the main problems when using CNN to segment layer is discarding point position information in the pooling process. However, semantic segmentation requires accurate alignment of sort maps, and therefore needs to preserve position information. This problem is solved by the architecture of the *decoder - encoder*. The encoder gradually reduces the spatial dimension by joining the layers, and the decoder gradually restores the object details and spatial dimensions. In this section I show two popular architectures *PSPNet* and *UNet*, which use the basic decoder - encoder idea, and further extend it with other features. [20]

2.8.1 PSPNet

Pyramid Scene Parsing Network (PSPNet) modifies the basic ResNet architecture by adding a dilation convolution and functions after the initialization association layer. The network of PSPNet uses the ability to globally connect information from differently sized regions. That is why it is called a pyramid scene - it advances from large parts of the image to smaller ones, where the details are only fine-tuned. Illustration of this architecture can be seen in Figure 2.7 [21–22]

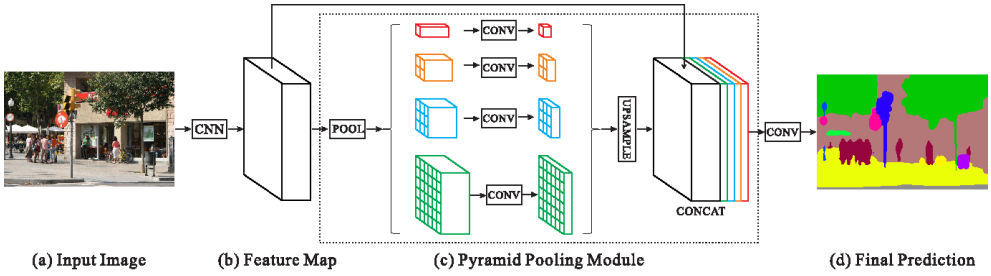


Figure 2.7. PSPNet Architecture. Source: [21].

[1] `https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_watershed.html`

2.8.2 UNet

The U-Net architecture consists of a context input path and a symmetric extension path that allows for accurate pixel localization. This network only needs fewer images to learn than previous networks. It is also very fast, segmenting an image of 512x512 pixels takes less than one second on a powerful GPU. U-Net simply concatenates the encoder mapping elements with the decoder mapping properties at each stage, creating a ladder-like structure, see Figure 2.8. The architecture allows shortcuts in each learning phase to obtain the relative properties that are lost in the associative layer in the encoder. [23–24]

There are multiple other architectures extending the basic idea of UNet, to name a few: Attention U-Net [25], Unet++ [26], and many others. The aim of this thesis is not to compare all possible architectures, which could possibly increase the accuracy by a few percent, but choose tried and tested well-behaving architecture train a model, and focus on data preparation and result presentation.

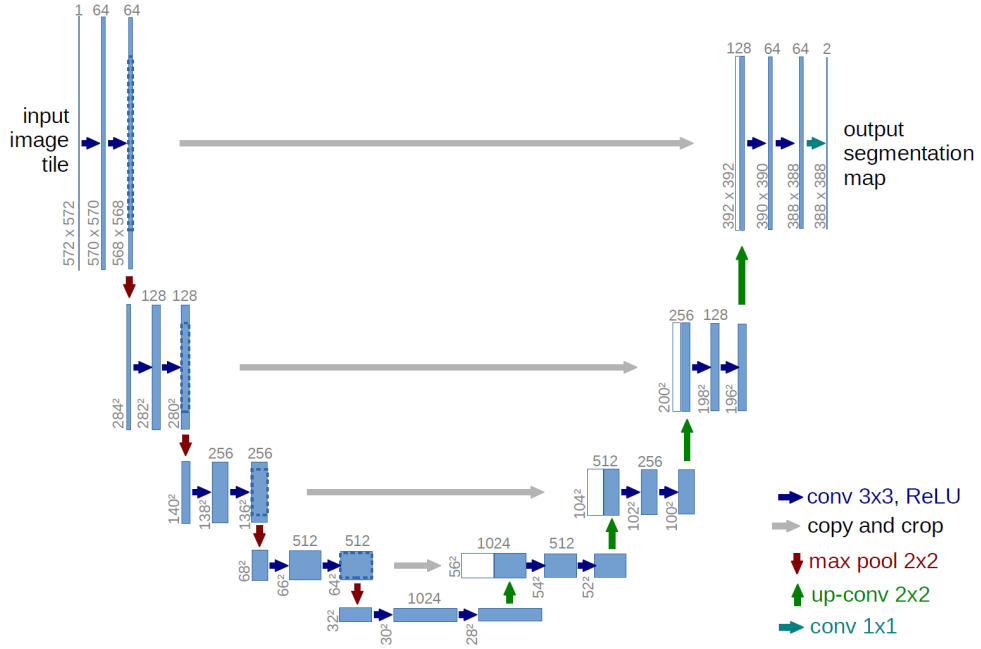


Figure 2.8. Example of Unet Architecture. Blue boxes represent multi-channel feature maps and white boxes copied feature maps. The arrows denote the different operations as shown in the legend bottom right. Source: [24].

2.9 Evaluation Methods

When evaluating a standard machine learning model, we usually classify our predictions into four categories: true positives, false positives, true negatives, and false negatives. From these categories we can calculate accuracy and other scores, which will be described in detail in the following section. However, for the segmentation tasks it is not always clear what is true positive, etc., because we do not have a single object to classify. Moreover, we may have multiple classes, which could cause class inbalance and accuracy score could became a misleading metric. For this reason it is suitable define different metrics,

which better describe the performance of our trained model when predicting segmentation masks. [27]

2.10 Metrics

The area of overlap between human annotator and a segmentation output of the algorithm is a region where the algorithm output pixels exactly match the annotated ground truth segmentation. These pixels are known as *true positives (TP)*. The pixels which were erroneously segmented by the AI algorithm in comparison with the human annotated ground truth are known as *false positives (FP)*. The pixels which were supposed to be segmented by the AI algorithm but were missed are known as *false negatives (FN)*. The *area of union* combines the segmentation results of both human annotation and AI algorithm output and then subtracts the true positives to prevent those pixels from being double counted. [27–28]

- IoU Score (Jaccard Index)

  IoU is a metric to calculate how accurate the predicted mask is with the ground truth mask. It is calculated as an area of overlap (between the predicted mask and the ground truth) and divided by the area of the their union. If the area of overlap equals the area of union, we have perfect segmentation and the IoU is equal to one. In that case, FP, TP and FN are all equal to zero. [28]

$$IoU = \frac{areaofoverlap}{areaofunion} = \frac{TP}{(TP + FP + FN)}$$

- Dice Coefficient

  Dice Coefficient is another frequently used metric measure of overlap between the prediction and the ground truth, when evaluating segmentation results. It is calculated as the area of overlap (between the prediction and the ground truth) multiplied by 2 and divided by the total area (of both prediction and ground truth combined). Same as the IoU score, this metric ranges between zero and one where a 1 denotes perfect and complete overlap. [29]

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

- IoU and Dice relation By using algebraic operations between Dice and IoU coefficient we can find this relation between these two metrics:

$$IoU = \frac{Dice}{2 - Dice}$$

2.11 Existing programs and algorithms

Here I list various existing software and algorithm which were developed for histology image inspection and cell classification or detection.

2.11.1 QuPath

QuPath[1] is an amazing tool for whole slide digital pathology image analysis, as it is open source, up-to-date, highly customizable and powerful. Annotating data using QuPath is fast and precise. The application also enables programmers to write custom scripts or plugins with Groovy scripting language.

Since QuPath is build in Java it is multiplatform and it can be run on Windows, Linux or Mac OS. It can also be run from the command line interface to perform a predefined task of a supplied script without the need to use a graphical interface and manually importing project, loading images and processing them. [30]

2.11.2 Pancreas++

Pancreas++[2] is an algorithm and software developed in 2013, which can be used for islet area investigation and $\alpha, \beta-$cell quantification, as well as position within the islet for either single or large batches of fluorescent images. The algorithm uses active contour models to quantify images accurately and quickly, resulting in an output of a spreadsheet format. The drawback of using a Pancreas++ is, that it only support a single image as an input which has to be exactly of resolution 256×256 pixels. Since it uses detection by color heavily, the user must make sure the alpha cells are green, beta cells are red, and all else neither green nor red. [31]

2.11.3 Cell Profiler

Cell Profiler[3] is an open-source application designed for biologist which enable fast and easy loading and automatic processing of microscopy images. It also enables exporting result data into either spreadsheet or database. [32]

Cell Profiler publication was cited more than 10 000 times up until today. Researchers in various medical fields used it to quickly identify cells and to save time compared to manual labeling. For instance, a pipeline termed MuscleAnalyzer pipeline for CellProfiler to automatically process immunofluorescence images of muscle cross-sections stained with laminin-α2 (to label muscle fibers) and DAPI (to label cell nuclei). It showed that instead of analyzing 67 images tissue images by the manual approach for 3 hours by an experienced investigator a server computer running the MuscleAnalyzer is able to process the same amount of images in roughly 11 minutes with similar accuracy. [33]

[1] https://qupath.github.io/
[2] https://www.nia.nih.gov/research/labs/pancreas
[3] https://cellprofiler.org/

Chapter 3
Tools Used

3.0.1 Python

Python[1] is an interpreted high-level programming language created by Guido Van Rossum, first released in 1991. It is designed to be easily readable and for general-purpose programming. Python has a huge set of libraries which can be used for machine learning and artificial intelligence programming (e.g. NumPy, SciPy, Scikit Learn, Tensorflow, Keras). Many of these libraries are using C or Fortran as a backend to preform heavy computation tasks very quickly.

3.0.2 Juypter

Project Jupyter[2] is a tool which enable users to write their code into a so called Notebook, providing an interactive computing thanks to running code directly after executing a specified part of code (called Cell). Running a code in a Cell displays the output directly after that Cell. This feature is very useful for rapid prototyping applications, data science or reports as the Notebooks can be easily exported into normal a Python code or a PDF document.

3.0.3 TensorFlow

TensorFlow[3] is an open source framework for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (GPU, CPU, TPU), and from desktops to clusters of servers to mobile and edge devices. It was originally developed by researchers and engineers from the Google Brain team within Google's AI organization. TensorFlow framework provides strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains. TensorFlow is an excellent choice for creating Deep Neural Networks with endless possibilities for fine-tuning each individual component of the network. [34]

3.0.4 Keras

Keras, on the other hand, is a high-level Neural Networks API. It is also open source. The main focus of this library is to bridge the gap between the low-level TensorFlow's computational functions and nice and easy user-friendly experience, with still taking advantage of fast environment for experiments. When prototyping or researching a Deep Neural Network it might be a tedious work to write and debug Neural Networks in pure TensorFlow framework and this is where exactly Keras could be a right choice. Keras backend can also

[1] `https://www.python.org/`
[2] `http://jupyter.org/`
[3] `https://www.tensorflow.org/`

be easily switched, and besides TensorFlow, CNTK[1] or Theano[2] can be also used. Since TensorFlow, CNTK and Theano all support both GPU and CPU training/predicting it is not a surprise that Keras supports them too. There exists two types of APIs for defining Neural Networks models in Keras:

- Sequential
- Functional

The *Sequential* model is a linear stack of layers and it is straightforward when defining a model architecture. The *Functional* API was designed to make it easier for defining more complex models, such as DAGs, models with shared layers, or models with multiple inputs and/or outputs. [35–36]

3.0.5 OpenSlide

OpenSlide is a tool for reading whole-slide images, written in C, but also provides binding for Python and Java. Openslide is widely used, because it can be used to read various virtual scan formats, and unifies and simplifies the process by providing a user to use only this tool and not to struggle with many other, often proprietary, formats and their own software for displaying whole-slide images.

OpenSlide can open following file formats:

- Aperio (.svs, .tif)
- Hamamatsu (.vms, .vmu, .ndpi)
- Leica (.scn)
- MIRAX (.mrxs)
- Philips (.tiff)
- Sakura (.svslide)
- Trestle (.tif)
- Ventana (.bif, .tif)
- Generic tiled TIFF (.tif)

The software package also includes an interface called Openslide Python API which includes a tile generator and a simple web-based viewer based on OpenSeadragon[3] JavaScript library.

[1] https://www.microsoft.com/en-us/cognitive-toolkit/
[2] http://deeplearning.net/software/theano/
[3] https://openseadragon.github.io/

Chapter 4
Problem Analysis

In this chapter I describe the data and methods I used in this thesis project. The programming language selection and tools are purely a subjective preference, and similar outcome could be of course achieved with any other programming language or data.

4.1 Data acquisition

There are not many large public databases of open source histology images, with easy access to download data, all the more so with the ability to filter images by tissue type. Histopathology scan datasets can often be found as accompanying files with various public challenges, such as Kaggle[1], etc.

Another great place to find not only microscopy scan images is GTEX Tissue Image Library [37]. I used this portal to find a suitable images for this thesis project. The process of downloading them and pre-processing is described in the following subsection.

4.1.1 GTEX Tissue Image Library

The GTEx Tissue Image Library contains detailed tissue histology images collected from numerous different tissue types from nearly 1000 postmortem donors. All tissues underwent stringent pathology review for tissue acceptability and each file contains details including the type of fixative, the degree of autolysis, as well as age range and gender. Additionally, the high resolution of each image allows for detailed viewing including pan and zoom. The scans can be downloaded as Aperio image files for further analysis. [37] As already mentioned all scans used, for this thesis project were obtained from this library.

[1] `https://www.kaggle.com/`

Chapter 5
Implementation

5.1 Data Preprocessing

My colleague who is a histology expert was given the GTEX scan data using QuPath histology software inside a project where I randomly pre-generated yellow boxes using the script `Generate BBoxes.groovy` and was asked to make annotations inside these boxes across various scan images to prevent overfitting on a single type of H&E scans.

Each box has a dimension of $1000 \times 1000 \mu m$, which should correspond to preview annotation tiles shared by the Seeker. Each yellow box containing annotation(s) was exported at highest possible resolution - resulting in 2023×2023 pixel images.

In total, 147 tiles were manually annotated across 13 scan images, which altogether with their description (patient sex, age and pathology category), can be seen in table 5.1.

Tissue Sample ID	Sex	Age Bracket	Pathology Categories
GTEX-ZF3C-2026	female	50-59	atrophy, fibrosis
GTEX-ZG7Y-0326	male	50-59	fibrosis
GTEX-ZLFU-0726	male	40-49	atrophy, fibrosis
GTEX-ZPCL-0726	female	60-69	
GTEX-ZPIC-0926	female	40-49	
GTEX-ZPU1-0226	male	40-49	atrophy, fibrosis
GTEX-ZT9W-0926	male	50-59	
GTEX-ZV7C-0726	male	50-59	fibrosis, pancreatitis
GTEX-ZVP2-0726	male	50-59	fibrosis
GTEX-ZVZP-0626	male	50-59	
GTEX-ZYFG-0826	female	60-69	
GTEX-ZYWO-1326	female	40-49	
GTEX-ZZPU-0726	female	50-59	cyst

Table 5.1. Used scans from GTEx Portal. Table is exported from [37].

5.1.1 Manual Annotation, QuPath

We closely inspected multiple Aperio scan images and decided to annotate the following classes, for future segmentation algorithm to distinguish:

- Islets
- Fat
- Vessels
- Ducts
- Tissue

- Other (background and other unwanted objects)
- Acinar

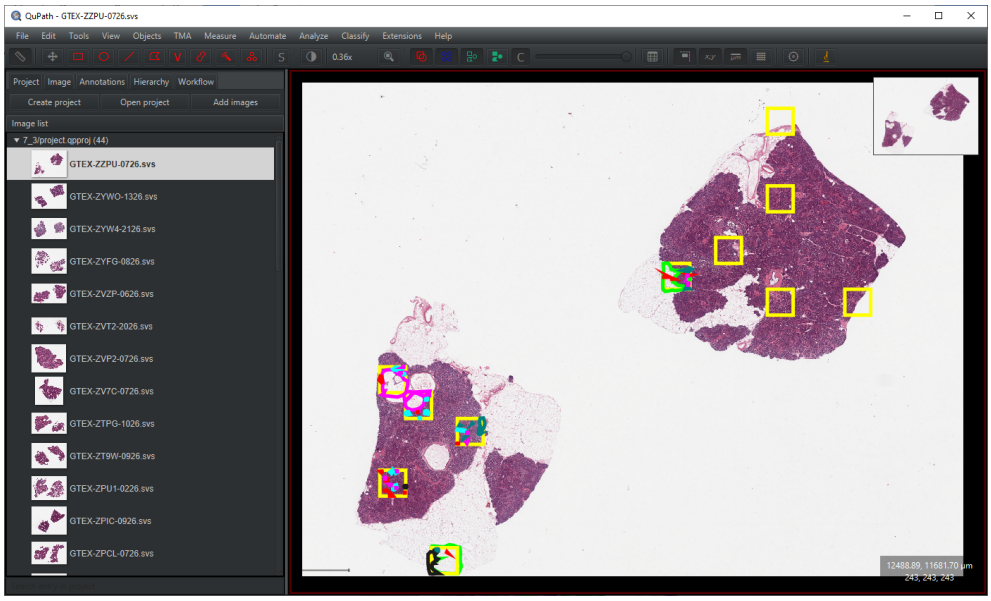


Figure 5.1. Preview of randomly pre-generated yellow boxes for annotator. Every annotation inside this box is then exported as a tile in maximal possible resolution. By randomness of box position the annotated data covers various objects on different locations of sample. This should prevent overfitting on a single object or scan sample.

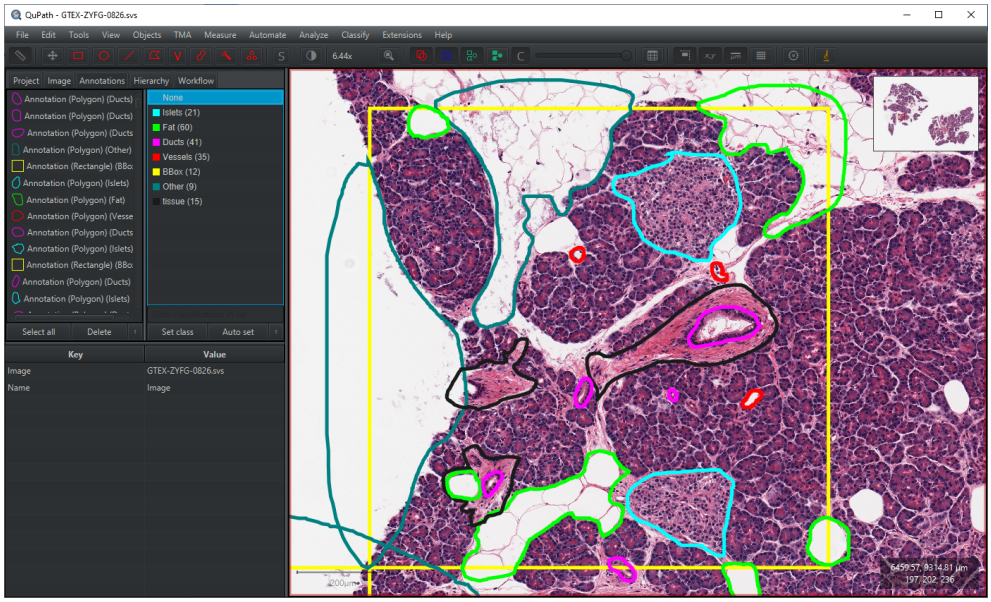


Figure 5.2. Manually annotated tile - inside a yellow pre-generated box. The tissue around ducts belong to the new "Tissue" class. Notice the misclassified Fat area - manual annotations may contain few errors, and the model still be able to learn well.

Some of the annotations were drawn with overlaps for simplicity. Later in the tile exporting step, I cut these overlaps based on class priority (as can be seen in numbering above), so each pixel belongs to only one class.

Annotating data using QuPath was fast and precise. The application also enables us to write custom scripts with Groovy scripting language and I have created multiple scripts to help us export/import and validate the data throughout the challenge solving.

The scripts written for this thesis includes:

- Generating random yellow boxes for annotator to define annotation tiles
- Exporting annotated areas as binary masks and image data
- Obtaining full resolution tissue crops
- Re-importing predicted segmentations for close inspection

5.1.2 Data Splitting

Multiple techniques and steps were used to correctly prepare my dataset for training. First the annotated data were exported from QuPath in the form of an image and binary mask for every scan and every tile (predefined yellow boxes).

Since the tile dimensions were still large for efficient and fast training, I decided to split this data to 4 sub-tiles of dimensions 512×512 pixels (with a little overlap in the middle).

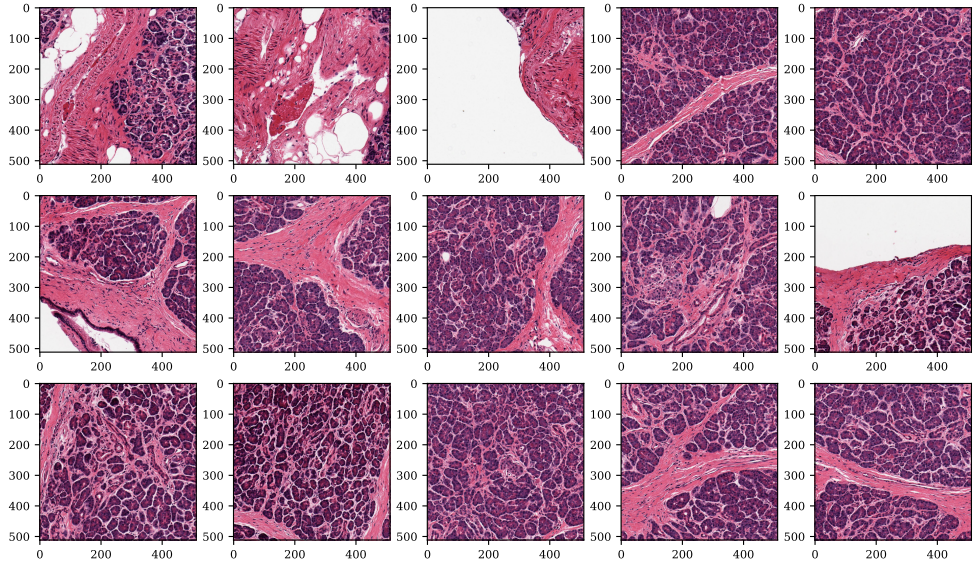


Figure 5.3. Preview of train data tiles.

Therefore I obtained 588 tiles in total. Data were then randomly split to training, validation and test subset as following:

- Training set: 80% - 470 tiles
- Validation set: 15% - 88 tiles
- Testing set: 5% - 30 tiles

5.1.3 Data Augmentation

Later during training I used a random data augmentation for both training and validation set in the form of randomly shifting, shearing, scaling, rotating, flipping the tiles as well as changing color slightly, blurring and shifting contrast.

Type of augmentation	Apply chance	Rate and Limit
Horizontal flip	0.5	
Vertical flip	0.5	
Random rotation by 90 degrees	0.75	
Shift and scale	1.0	0.1
Additive Gaussian noise	0.2	
CLAHE	0.2	clip limit 2
Random gamma	0.2	
Random brightness and contrast	0.2	
Sharpen	0.1	
Blur	0.1	3
Motion blur	0.1	3
Hue and saturation adjust	0.3	

Table 5.2. Used data augmentations.

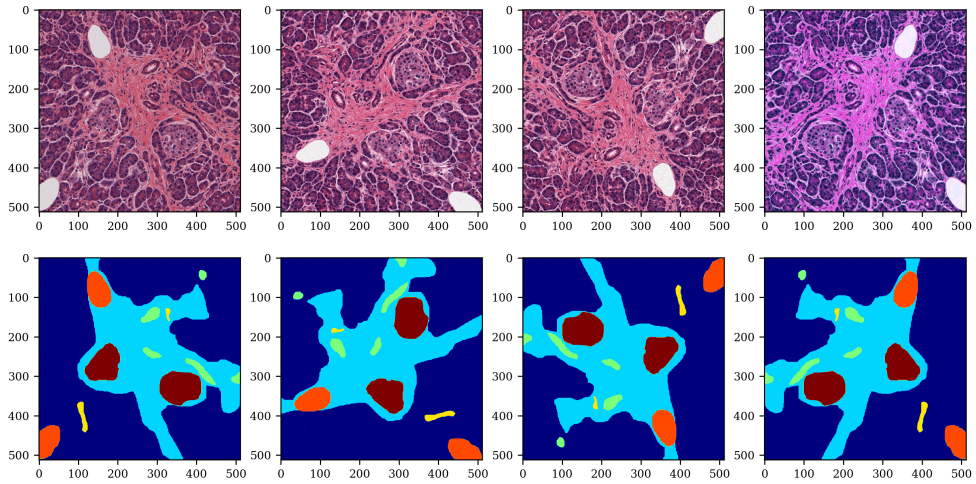


Figure 5.4. Four different augmentations of one sample tile with their respective mask adjusted by same transformation as the input image.

5.2 Model Training

For deep learning the model I used U-Net architecture, which is able to output a segmentation of classes for each pixel in the input tile image. For the implementation in Python I used a library called Keras (deep learning framework as mentioned in the Chapter 3) and I also used another excellent library called *Segmentation Models* [38] which enables really fast prototyping and tuning of deep neural segmentation models.

The input layer was of shape $512 \times 512 \times 3$ (RGB channels), and the n-dimensional array of shape $512 \times 512 \times 7$ where each layer contained a pixel probabilities of given pixel to belong to this class. I used a softmax function layer on the output, so I could see a prediction/likelihood of pixels being in the class to be able to set a custom threshold.

I used `Resnet34` [15] as a backbone for the U-Net network. The model was compiled with Adam optimizer [39] (LR 0.0005) and combined lossed function of DiceLoss and BinaryFocalLoss, both with ratio of 0.5. Next, both an IOU

Score and F-Score were monitored, which helped a lot the network not to overfit on specific classes.

The training was run with EarlyStopping[1] with patience 10 on IoU coefficient validation score, ReduceLROnPlateau[2] with planned 30 epochs, but stopped due to the EarlyStopping rule after 20 epochs. As an activation function I used which preform well when training a model on multiclass data. As a loss function, I used a combined loss function of Weighted Dice Loss and Categorical Focal Loss [40] (the losses were just summed together). The training progress and continuous validation on validation set can be seen in Figure 5.5 and the training and validation loss in Figure 5.6.

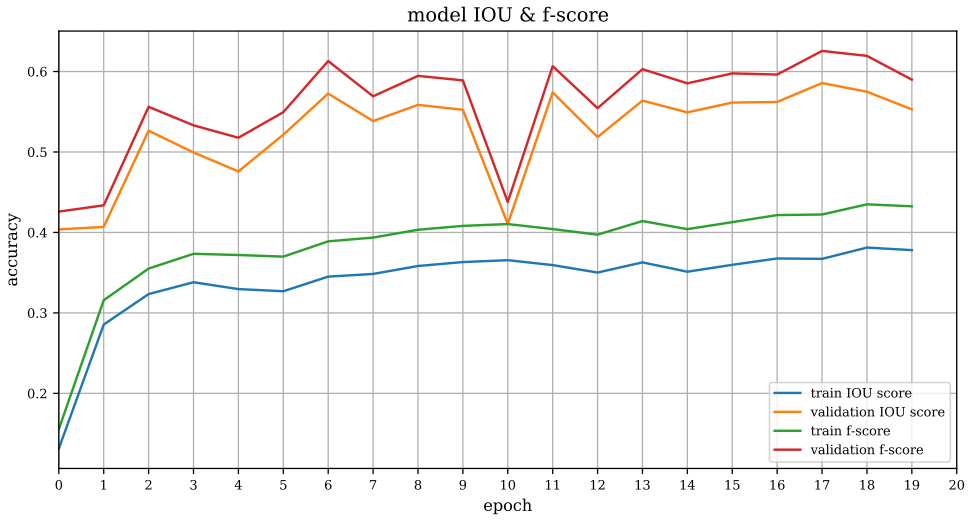


Figure 5.5. Model Training Score.

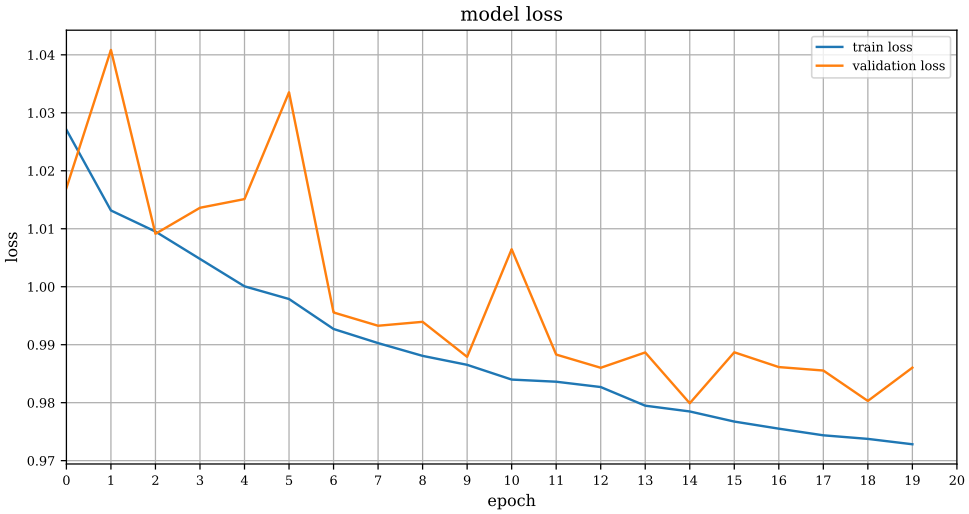


Figure 5.6. Model Training Loss.

Many of first tries failed, because model was not paying attention to some class (e.g. Islets). The issue was partially solved by computing class weights by including a class weights to the model. The class weights were computed by

[1] https://keras.io/api/callbacks/early_stopping/
[2] https://keras.io/api/callbacks/reduce_lr_on_plateau/

iterating over each training image and by counting pixels for each class. The total ratio of counted pixels was used to generate fixed weights to help balance the DiceLoss [41] while training. The resulting class weights distribution can be see in a bar plot in Figure 5.7.

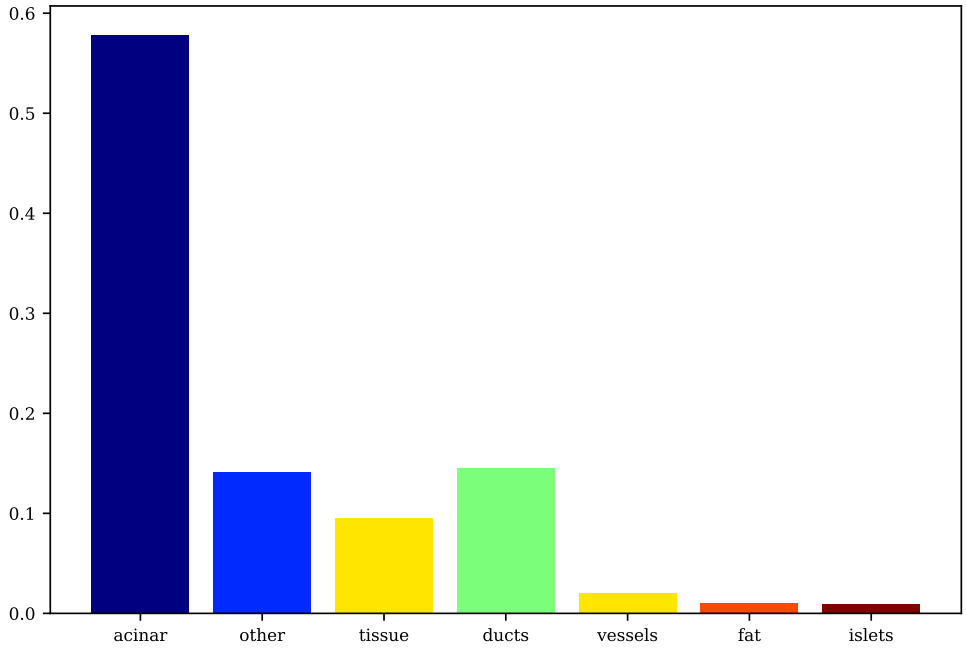


Figure 5.7. Class distribution by total pixel rate

- Acinar: 57.83%
- Other: 14.16%
- Tissue: 9.51%
- Ducts: 14.52%
- Vessels: 2.01%
- Fat: 1.07%
- Islets: 0.90%

5.3 Hardware Requirements

The software was tested with the following hardware setup. The training and all segmentation computing was performed with a computer of this specification. Lower specifications may also work, but the processing can be slower or RAM inefficiency may cause model not to be able to segment some larger files. The deep neural network algorithm and used framework can highly utilize graphics card acceleration and therefore a good graphics card (GPU) is highly advised - the inference should also work only with use of CPU, but the speed of segmentation will be approximately 10 times slower.

Testing configuration

- CPU: AMD Ryzen 5 3600 6-Core
- RAM: 32 GB
- Graphics card : NVIDIA GeForce RTX 2070 (8 GB VRAM)

- OS : Windows 10 Pro, Ubuntu 20.04.1 LTS (Focal Fossa)

Recommended configuration

- CPU: Intel Core i9-9900K (8-Core) / AMD Ryzen 9 3900X (12-Core)
- RAM: 64 GB
- Graphics card : NVIDIA GeForce RTX 2080 (8 GB VRAM)
- OS : Windows 10 Pro / Linux

5.3.1 Test Data Evaluation

After training finished, the best model weights were chosen, in terms of maximal IoU and F1 scores. Final model with imported weights achieves these results on test dataset consisting of 30 tiles, which were excluded during training phase:

- Loss: 0.987
- Mean IoU Score: **0.581**
- Mean F1 Score: **0.615**

Predicted segmentation on test samples can be seen in Figure 5.8, and compared with the Ground Truth annotations.

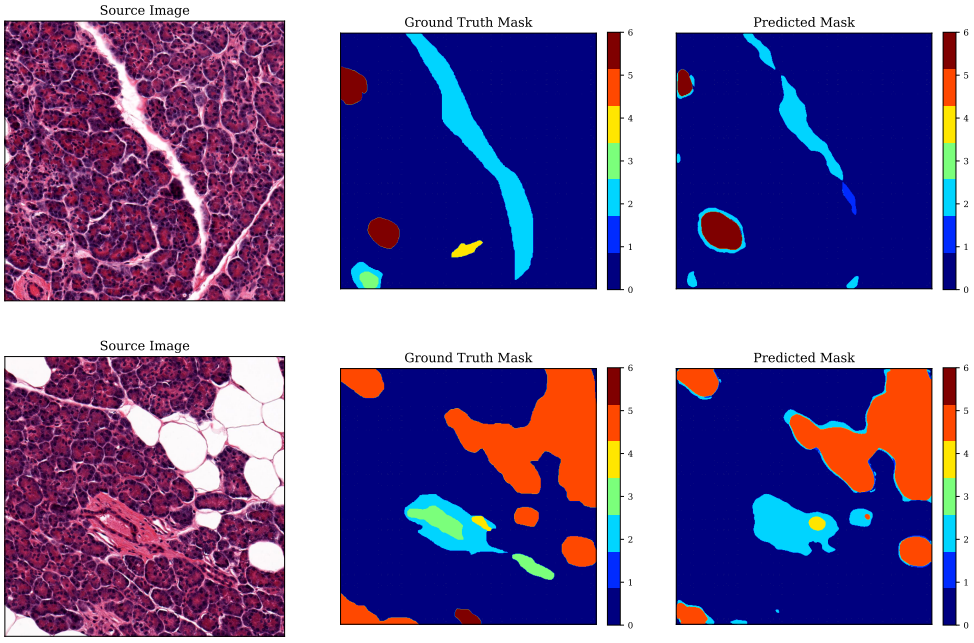


Figure 5.8. Test data sementation output and comparasion with GT.

5.4 Whole-slide scan tiling and segmentation

Since the model only accepts images of dimensions 512×512 pixels, all data need to be tiled first in order to predict the segmentation, therefore every sample of the desired set of scans to be segmented was fed to the trained model tile by tile of size 512×512 pixels. The samples were cropped automatically from the scan using a tissue detection script inside QuPath and their image data was fed to Python algorithm.

5.4.1 Exporting masked samples from QuPath

Because the scan is usually very big and vast majority of the image is just a background and not the tissue we are interested in, tiling and predicting each position in the image would slow down the process significantly and algorithm would also need more memory to save each tile. Therefore, I used a function in QuPath called `SimpleTissueDetection2` which use a simple segmentation method of thresholding to detect only the tissue sample and separate it from the background. We use this detection to make a boundary and mark it as an annotation `crop`. Then we draw a bounding box around each detection and we crop only this part of image to be then processed in Neural Network segmentation. The Groovy code for this tissue sample detection and annotation with parameters to match most of the samples correctly is shown in the following listing:

```
// remove all existing annotations
removeObjects(getAnnotationObjects(), false)

detectParams = '{"threshold": 239,' +
'"requestedPixelSizeMicrons": 70.0, ' +
'"minAreaMicrons": 600000.0,' +
'"maxHoleAreaMicrons": 2000000.0,' +
'"darkBackground": false,  "smoothImage": true,' +
'"medianCleanup": true, "dilateBoundaries": true,' +
'"smoothCoordinates": true,' +
'"excludeOnBoundary": false,  "singleAnnotation": false}'

runPlugin('qupath.imagej.detect.tissue.SimpleTissueDetection2',
  detectParams);

def cropClass = getPathClass('crop')
def annotations = getAnnotationObjects()
for (a in annotations) {
    def roi = a.getROI()
    def pathClass = a.getPathClass()
    def classificationName = pathClass == null \
      ? 'None' : pathClass.toString()
    a.setPathClass(cropClass)
}
```

5.4.2 Overlapping method, merging and averaging

When predicting, because the neural network sometimes wrongly predicted the class of pixels near the edge of tile, I also incorporated a tile overlap when tiling the image. The overlap value can be adjusted. If we increase the overlapping factor, we get more accurate results on tile boundaries, as they are averaged from multiple tiles and therefore the prediction values are softened.

In the following illustrations I show the method I use for splitting the source image to tiles, predicting the segmentation on each separately and then the process of merging them again into single matrix and averaging the result.

First step consists of tiling the image. Let us have an input image of 5×5 pixels as can be seen in Figure 5.9 in subplot **a)** on left. The red box

of dimensions 3×3 pixels represents one tile currently being cut, and blue dashed lines the areas of other tile cuts. In this case the overlapping factor is 1/3 - the cutting window of this size is therefore moved by 2 pixels on each cut. We obtain 4 tiles and perform the segmentation on each. In subplot **b)** we can see the segmented tiles, where each pixel value represents the likelihood if this pixel to be a labeled of the given class.

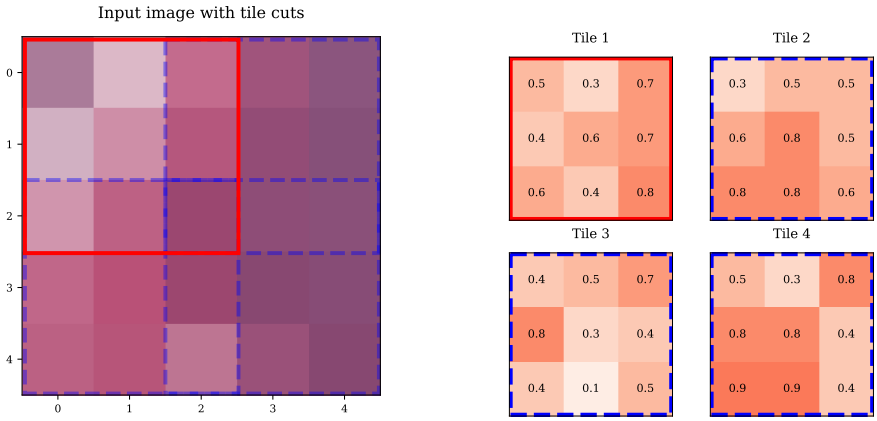


Figure 5.9. Example of tiling input image

After segmenting each tile separately, we create new matrix of the same width and height as the source image before cutting. Each class is represented as one dimension in this matrix and all values are initially set to zero. Next, we populate this matrix by placing the segmented tiles of predictions to their respective original location, summing the current values pixel-wise, as illustrated in Figure 5.10

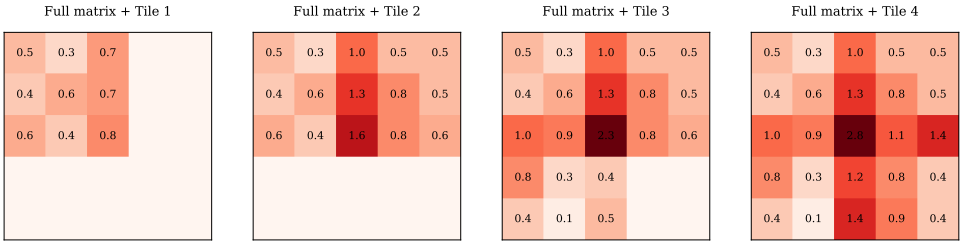


Figure 5.10. Example of summing the segmented tiles.

Finally after assembling the full matrix of probabilities, we need to normalize the pixel values with overlays, by dividing these values with number of occurrences of this field. Mathematically, we can create an extra occurrence matrix, where we increment positions of overlapped values and use Hadamard division, defined as

$$\mathbb{A} \oslash \mathbb{B} = (a_{ij} \div b_{ij}) = \begin{pmatrix} a_{11} \div b_{11} & \cdots & a_{1n} \div b_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} \div b_{m1} & \cdots & a_{mn} \div b_{mn} \end{pmatrix}, \tag{1}$$

where $\mathbb{A}$ is the summed matrix of overlapped predictions and $\mathbb{B}$ is the occurrence matrix of same size. The matrix $\mathbb{C} = \mathbb{A} \oslash \mathbb{B}$, is the final resulting matrix with normalized values element-wise. In our example we can see this operation illustrated in Figure 5.11.

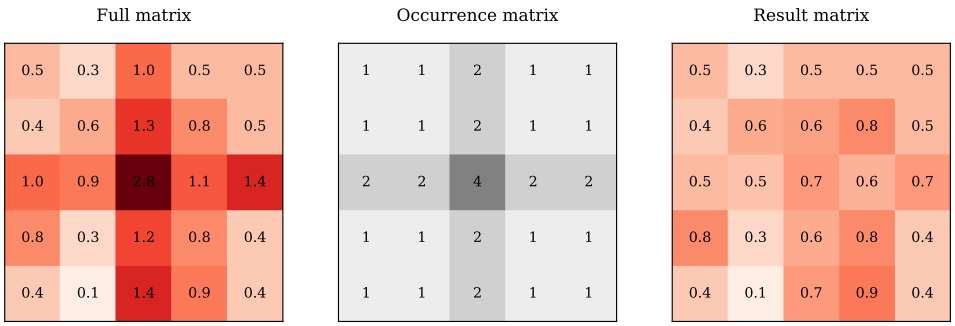


Figure 5.11. Illustration of matrix element-wise division called Hadamard division.

Because we have multiple classes, we populate each dimension of Result matrix with one class. After every single tile prediction we then compute the average class likelihood from merged predicted tiles and then the class with highest probability is assigned to the pixel. This process is illustrated with 3 sample classes in Figure 5.12.

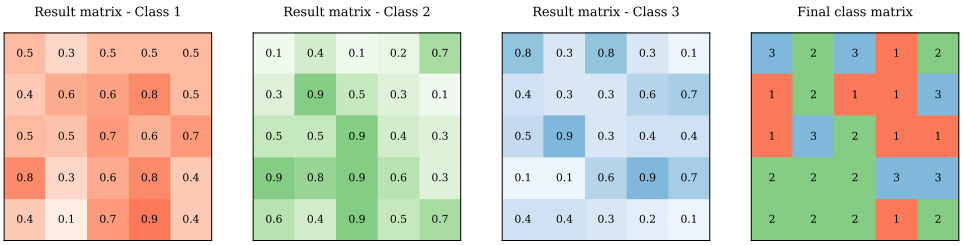
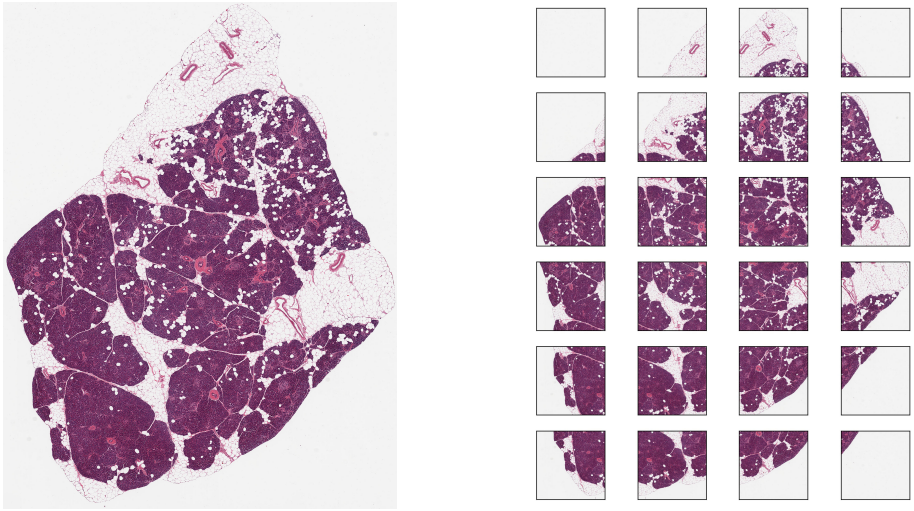


Figure 5.12. Highest likelihood class selecting.

5.4.3 Demonstration on histological data

Now, after illustrating the process of tiling image, predicting and merging it back into a single predicted segmentation mask, I'll show the process on the real output from QuPath. In following Figure 5.13 we can see the input cropped tissue sample to be tiled and (not precise) indication of how this one image could be tiled in with a certain overlap.



a) b)

Figure 5.13. Illustration of the tiling function. Image **a)** on left shows the input JPEG image, obtained from QuPath. Image **b)** illustrates the tiling with 30% overlaps. Tiles in this illustration are not of dimension 512×512 pixels as it would be while processing.

For example if the original whole-slide scan had a full resolution of 40000×40000 pixels and we detected a tissue sample at position $x = 5000$ px and $y = 5000$ px and with height $h = 16000$ px and width $w = 8000$ px in the original resolution, we can recalculate the position of the 2.0 times downscaled version to be $(x_{2.0} = 2500, y_{2.0} = 2500, w_{2.0} = 4000, h_{2.0} = 8000)$ px. We preform this downscale, because we also used a downscaled tiles in the training process from 1024 to 512 px ($downscale = 2.0$). Now we obtained a single crop sample of resolution 4000×8000 px. Therefore if we now use a tiling operation with 30% overlap and tile size of 512×512 px, we will get $\lfloor 4000/512 \cdot 1.30 \rfloor = 10$ tiles in horizontal direction and $\lfloor 8000/512 \cdot 1.30 \rfloor = 20$ tiles in vertical direction, so in total $10 \cdot 20 = 200$ tiles to process with segmentation algorithm. If we would not use the tissue sample detection and crop, we would have to process all tiles, in this case $\lfloor 20000/512 \cdot 1.30 \rfloor \cdot \lfloor 20000/512 \cdot 1.30 \rfloor = 2500$ tiles. This could save a significant amount of time, when there is a lot of background pixels and small area of tissue.

In the Figure 5.14 bellow we can see the heatmap segmentation output for each class for the input image 5.13. The likelihood values ranges from 0 (less likely the given pixel is of the given class) to 1 (more likely the given pixel is of the given class).

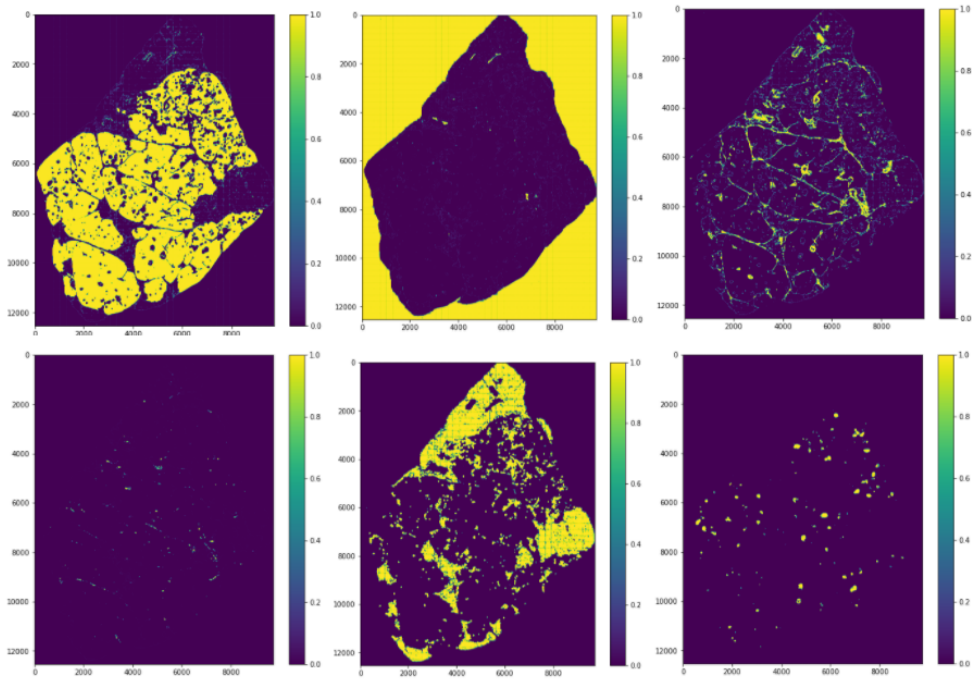


Figure 5.14. Preview of prediction masks for each class. From Top Left to Right: Acinar, Other, Tissue, Vessels, Fat, Islets. (Ducts not shown here)

Chapter 6
Inference

6.1 Inference Pipeline

The Interface process consists of the following steps:

(i) Selected scans are automatically loaded by QuPath software in the background, using the `Export All.groovy` script, which opens the scan, automatically detects the tissues samples within the scan, using the QuPath's tissue detection plugin, and these regions are exported in high resolution (downsampled by factor of 2.0) for future segmentation. The crops are made to save time by not tiling and predicting the background areas, where there is no tissue sample.

(ii) Next the UNet model is built with the same parameters as the best model during the training stage. Afterwards its weights are loaded from `models/best-model.h5`

(iii) The scan crops are tiled to 512×512 tiles with, by default, 30% overlap. These tiles are then passed to model for segmentation prediction, and then reassembled to their original position as before tiling, averaged (on the overlapping areas) and merged into the same shape as the input crop.

(iv) Thanks to saving the location, sizes and downscale factor of each crop when exporting from QuPath, we can re-position them to one big scan/image. This is the final segmentation result of the given scan.

(v) For the purpose of loading the segmentations to a web application and to be able to set opacity for each layer/class individually, the image is "exploded" into 7 images, where each image only contains pixels from a single class.

(vi) Lastly a file called `areas.json` is generated and it contains the total count of pixels for each of the class layers. From this area coverage ratios by classes can be calculated, and this is shown in the pie graph in the web application.

6.1.1 Areas JSON file structure

The generated `areas.json` file consist of keys with the same name as the respective scan name. It holds an inner object of separate key for every segmented layer referring to a number of total pixels in this scan image belonging to the given class. Preview of the file is shown in the following listing:

```
{
    "GTEX-1128S-1926": {
        "acinar": 10315714,
        "other": 68660945,
        "tissue": 3503715,
        "ducts": 3,
        "vessels": 60388,
```

```
        "fat": 14349497,
        "islets": 1786868
    },
    "GTEX-11LCK-0226": {
        "acinar": 10292623,
        "other": 58581740,
        "tissue": 2149272,
        "ducts": 2,
        "vessels": 115841,
        "fat": 9320912,
        "islets": 303166
    },
    ...
}
```

6.2 Web Application

I wanted this tool for segmentating histology image scans to be accessible even for non computer experts. First, I thought about creating a desktop application, but it is hard to distribute it, make it cross-platform and keep it updated. Therefore I decide to incorporate all the functionality into a single web application, which can be accessed on every PC, or even a mobile phone.

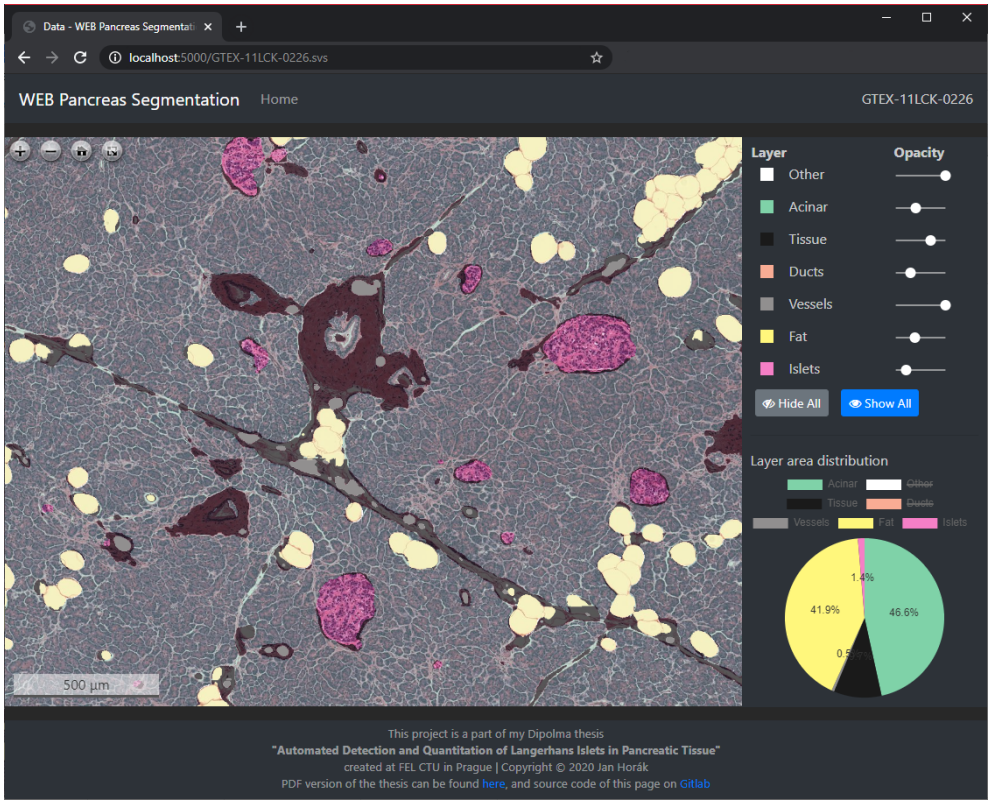


Figure 6.1. Web application screenshot.

The final web application consists of three separate pages.

- Home page
- Scan page

- Segment page

Home page is the root page displayed when a user visits the web. There is an information about this project, instructions for using the application and links for displaying processed scan images.

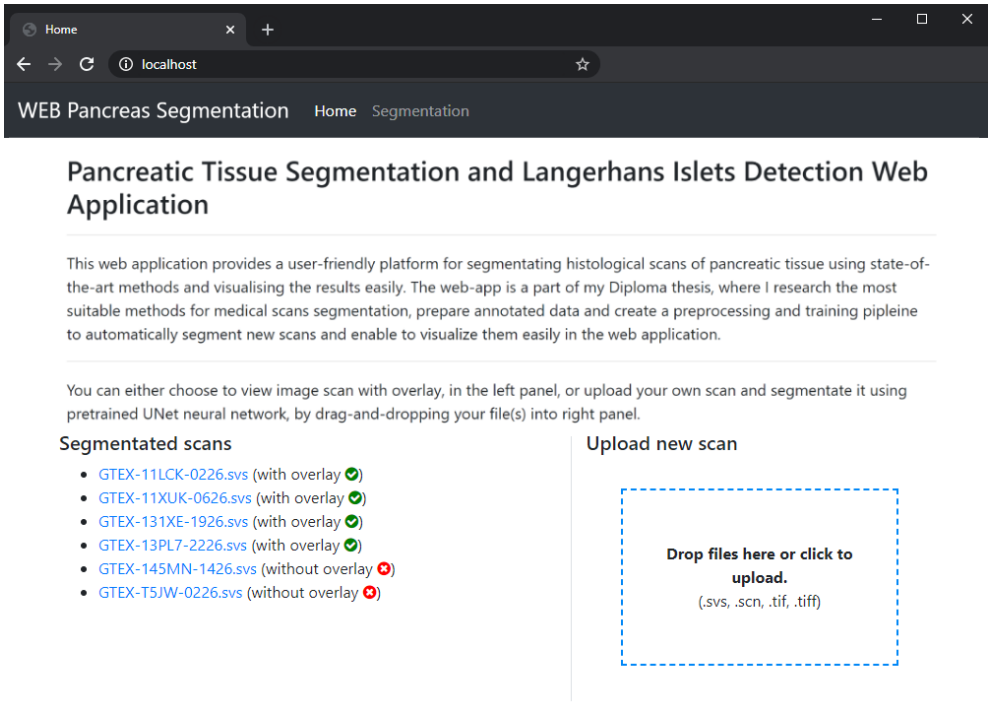


Figure 6.2. At the home page, the user can view all uploaded scans (scans in his scan folder), see for which of the scans the segmentations were already generated, and by using Drag-and-Drop on the right side panel, the user can upload new scans to process very easily.

Scan page is a page consisting of tree main elements: viewer window, sliders for changing opacity of individual segmentation layers and layer distribution pie chart. The sliders and graph are placed in a right-side panel. All the segmentation layers can be also hidden out with a single click of the *Hide all* button - so the scan sample can be seen without any overlay, or the overlay layers can be set to non-transparent mode with a button *Show all*, so user can easily see the segmentation regions. There is also one more preset mode called *Islets*, which is activated by clicking a button of the same name. This mode will show all layers to pre-defined opacity values, except the Islets layer, which is fully transparent. This will cause Langerhans islets to pop out and be easily visible among other tissue sections.

The layer distribution graph displays the ratio of coverage by each segmentation layer of the total scan area. By clicking on labels next to the pie graph, user can hide this segment and the graph is automatically re-adjusted to only include and quantify the non-hidden layers. The layer *Other* is initially hidden by default, as it mainly contains the background of the sample and thus interfere with ratios of objects of interest within the sample.

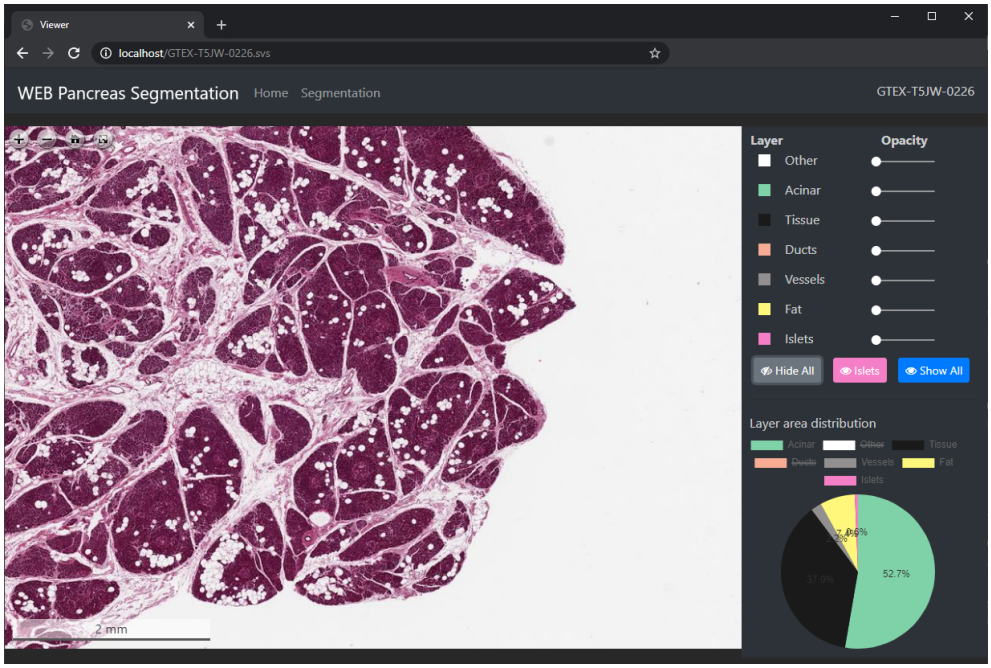


Figure 6.3. Source image without overlay. Each of the layers has its own slider in the right side panel. By sliding it, the user can adjust the opacity of that specific layer. Below the sliders there are 3 buttons: to hide all overlaid segmentation completely, display them at 100%, and one custom preset, to highlight the Langerhans Islets in the scan (as shown in this image).

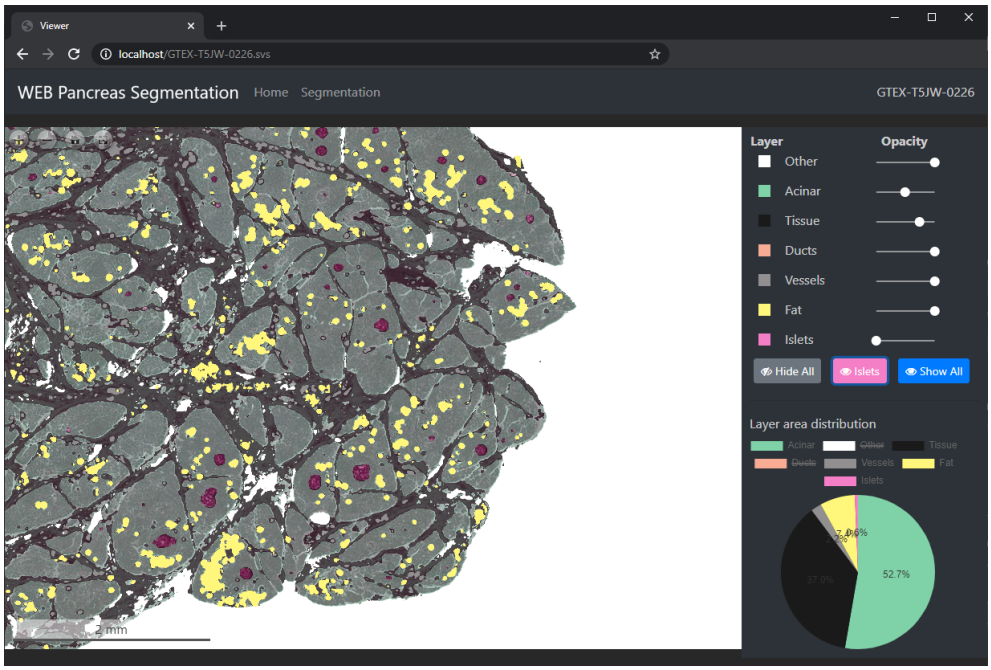


Figure 6.4. Image with *Islets* mode overlay. After the segmentation overlays are generated, an interactive pie graph is shown next to the viewer windows showing the area occupancy by each class. By clicking on the label, this class can be hidden and the ratios of the graph are recalculated accordingly.

Segment page is page to which user is automatically redirected when uploading a new scan for segmentation. On this page a list of not yet segmented

scans is shown. And user can select one or more scans by checking respective checkboxes of each scan. After clicking a confirmation *segment* button, automatic segmentation is started. This will call the function `segment-scans` of the `INFERENCE.py` script is called in the background and user is informed about the segmentation progress in the message log and with progress bar updates.

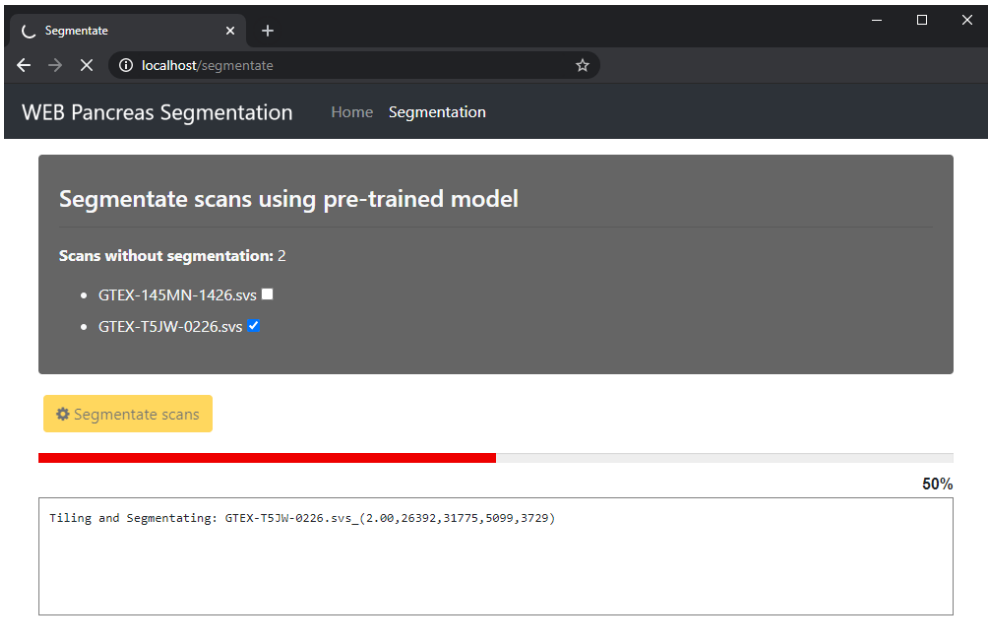


Figure 6.5. In the segmentation tab, users can run new segmentation tasks for selected scans with a single click of a button. The server will process the scans in background and after it's finished, scans can be displayed in the viewer.

Chapter 7
Conclusion

7.1 Testing with domain expert

For the testing of the usability and accuracy I asked a pathology expert from Second Faculty of Medicine, Charles University to test the web application and review it, to better capture an unbiased opinion and results:

"This technology is used primarily to search for suitable donors of pancreatic transplantation (respectively its islets of Langerhans). In this procedure, many sections (sometimes hundreds to thousands) of potential candidates are presented to the pathology laboratory (hence the clinic) and the number and quality of the islets of Langerhans are then evaluated manually. If we compare the technology used here with the usual manual procedure, the web application software was able to recognize almost all the islands of Langerhans in the attached images, altogether with marking of their area with an inaccuracies of a few percent. Regarding the marking of the edges of the islets, this is not a serious problem because the main parameter used in medicine to evaluate the condition of the endocrine component of the pancreas is primarily their number. Even with today's manual procedure, the size is determined only as a guide without an accurate measurement, therefore the results of the area of the Langerhans islets presented by the program correspond to medical standards. Upon close examination of the images, we found that the program shows a really small degree of false negativity (i.e. was able to identify virtually all islets of Langerhans) and a small degree of false positivity, when in cases it identified small vessels and pancreatic ducts such as Langerhans islets. The rate of these false positives in the attached images was in the order of units, and with the increase in the number of learning images, it can certainly be minimized. The attached data show great potential for the future, when with an increased number of learning data could serve as a pre-analysis, when a pathologist would see the already marked Langerhans islets and more or less just check the output of the program, which could lead in a very significant simplification and efficiency of the current process."

7.2 Future Improvements

There is certainly a lot of space for further improvements and extension of this project. Even better segmenting accuracy could be obtained by providing even more manually annotated data. I believe this was nearly the minimum of data for model to be able to generalize across multiple various H&E histology scans. The class balancing could also help a lot, because e.g. the Ducts were almost completely ignored by the neural networks as there was a minority of

this class in the training data for such a diverse tissue section. One could also use the tools such a training pipeline and web server, and completely switch the annotated data for any other tissue domain, such as Stomach or Bone marrow, and still would be able to utilize almost all of the functionalities (data preprocessing for training, whole scan segmentation by tiling, and web server for viewing the scans and segmented overlays in web browser).

The users who tested the web application also mentioned, it would be great if there would be a precise count of the instances detected (not just the percentage of area), and a ruler tool, so that an user could measure the sizes of objects directly in the slide.

7.3 Summary

During the creation of this diploma thesis, I became acquainted with many state-of-the-art techniques in the field of image processing, segmentation and work with biomedical data, specifically histological hematoxylin and eosin scans. I created a procedure for fully automatic segmentation of pancreatic tissue and the search for Langerhans islets by training a deep neural network model using data annotated by me and my colleague. I also created a web application that allows anyone to easily segment a pancreatic tissue scan just by grabbing and dragging it into the web application window, and after automatically processing it viewing that scan with overlays and other analytical tools such as determining the area ratio of each layer.

References

[1] Mark A Atkinson, George S Eisenbarth, and Aaron W Michels. Type 1 diabetes. *The Lancet*. 2014, 383 (9911), 69-82. DOI 10.1016/S0140-6736(13)60591-7.

[2] Rita Bottino, Massimo Trucco, A.N. Balamurugan, and Thomas E. Starzl. Pancreas and islet cell transplantation. *Best Practice & Research Clinical Gastroenterology*. 2002, 16 (3), 457-474. DOI 10.1053/bega.2002.0318.

[3] Lorenzo Piemonti. *Islet Transplantation*. South Dartmouth (MA): MDText.com, Inc., 2000.
`https://www.ncbi.nlm.nih.gov/books/NBK278966/`.

[4] Kevin P. Murphy. *Machine learning*. Cambridge, MA: MIT Press, c2012. ISBN 978-0-262-01802-9.

[5] Christopher M. Bishop. *Pattern recognition and machine learning*. [New York: Springer, c2006. ISBN 0-387-31073-8.

[6] David Fumo. *"A Gentle Introduction To Neural Networks Series — Part 1 (online)"*. 2017.
`https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-series-part-1-2b90b87795bc`. "[Cited 2020-09-21]" .

[7] K.-L Du, and M.N.s Swamy. *Perceptrons.* In: 2014. 67-81. ISBN 978-1-4471-5570-6.

[8] "Stanford University". Convolutional Neural Networks. *CS231n Convolutional Neural Networks for Visual Recognition* . 2017,

[9] Jan Horak. Lip Reading using Deep Convolutional Networks. *Bachelor's Thesis, Czech Technical University in Prague*. 2017,

[10] Adit Deshpande. A Beginner's Guide To Understanding Convolutional Neural Networks. *Personal Github blog (online)*. 2016,

[11] Petar Veličkovic. Deep learning for complete beginners: convolutional neural networks with keras. *Cambridge Spark* . 2016,

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. 2012.
`http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[13] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. *Going Deeper with Convolutions*. 2014.

[14] Karen Simonyan, and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015.

[16] F. Pedregosa. *Underfitting vs. Overfitting*. 2017.
http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html. [cit. 08-27-2020.

[17] William Koehrsen. Overfitting vs. Underfitting: A Conceptual Explanation. *Towards Data Science (online)*. 2018,

[18] Geoff Dougherty. *Digital Image Processing for Medical Applications*. Cambridge: Cambridge University Press, 2018-09-12. ISBN 9780521860857.

[19] V. Grau, A. U. J. Mewes, M. Alcaniz, R. Kikinis, and S. K. Warfield. Improved watershed transform for medical image segmentation using prior information. *IEEE Transactions on Medical Imaging*. 2004, 23 (4), 447-458.

[20] Lei Cai, Jingyang Gao, and Di Zhao. A review of the application of deep learning in medical image classification and segmentation. *Annals of Translational Medicine*. 2020, 8 (11), 713-713. DOI 10.21037/atm.2020.02.44.

[21] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. *Pyramid Scene Parsing Network*. In: 2017. 6230-6239.

[22] Eric Ku. *"ML In Detail 1: PSPNet (online)"*. 2020.
https://medium.com/analytics-vidhya/ml-in-detail-1-pspnet-4527036af33b. "[Cited 2020-09-21]" .

[23] Meet Pragnesh Shah. *Semantic Segmentation using Fully Convolutional Networks over the years (online)*. 2017.
https://meetshah1995.github.io/semantic-segmentation/deep-learning/pytorch/visdom/2017/06/01/semantic-segmentation-over-the-years.html.

[24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015.

[25] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. *Attention U-Net: Learning Where to Look for the Pancreas*. 2018.

[26] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. *UNet++: A Nested U-Net Architecture for Medical Image Segmentation*. 2018.

[27] Jeremy Jordan. *"Evaluating image segmentation models (online)"*. 2018.
https://www.jeremyjordan.me/evaluating-image-segmentation-models/. "[Cited 2020-09-20]" .

[28] Nigel M. Parsad. *"Deep Learning in Medical Imaging V (online)"*. 2018.
https://medium.com/datadriveninvestor/deep-learning-in-medical-imaging-3c1008431aaf. "[Cited 2020-09-20]" .

[29] Mark Polak, Hong Zhang, and Minghong Pi. An evaluation metric for image segmentation of multiple objects. *Image and Vision Computing*. 2009, 27 (8), 1223-1227. DOI 10.1016/j.imavis.2008.09.008.

[30] 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, Jacqueline A. James,

Manuel Salto-Tellez, and Peter W. Hamilton. QuPath. *Scientific Reports*. 2017, 7 (1), DOI 10.1038/s41598-017-17204-5.

[31] Hongyu Chen, Bronwen Martin, Huan Cai, Jennifer L. Fiori, Josephine M. Egan, Sana Siddiqui, and Stuart Maudsley. Pancreas++. *Frontiers in Physiology*. 2013, 3 DOI 10.3389/fphys.2012.00482.

[32] Claire McQuin, Allen Goodman, Vasiliy Chernyshev, Lee Kamentsky, Beth A. Cimini, Kyle W. Karhohs, Minh Doan, Liya Ding, Susanne M. Rafelski, Derek Thirstrup, Winfried Wiegraebe, Shantanu Singh, Tim Becker, Juan C. Caicedo, Anne E. Carpenter, and Tom Misteli. CellProfiler 3.0. *PLOS Biology*. 2018-7-3, 16 (7), DOI 10.1371/journal.pbio.2005970.

[33] Yeh Siang Lau, Li Xu, Yandi Gao, and Renzhi Han. Automated muscle histopathology analysis using CellProfiler. *Skeletal Muscle*. 2018, 8 (1), DOI 10.1186/s13395-018-0178-6.

[34] Martín Abadi, and others. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015.
`https://www.tensorflow.org/`.

[35] François Chollet, and others. *"Keras"*. 2015.
`"https://keras.io"`.

[36] François Chollet, and others. *"Keras: Sequential Model Guide"*. 2015.
`"https://keras.io/getting-started/sequential-model-guide/"`. "[Cited 2020-08-26]".

[37] NIH Common Fund. *The Genotype-Tissue Expression (GTEx) project*. 2013.
`http://www.nature.com/articles/ng.2653`.

[38] Pavel Yakubovskiy. *Segmentation Models*. 2019.
`https://github.com/qubvel/segmentation_models`.

[39] Diederik P. Kingma, and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014.

[40] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. *Focal Loss for Dense Object Detection*. 2017.

[41] Xiaoya Li, Xiaofei Sun, Yuxian Meng, Junjun Liang, Fei Wu, and Jiwei Li. *Dice Loss for Data-imbalanced NLP Tasks*. 2019.

Appendix A
Abbreviations and symbols

A.1 Abbreviations

List of used abbreviations in this text.

RGB	Red, green and blue color channels
NN	Neural (or Neuron) Network
DNN	Deep Neural Network
GPU	Graphic Processing Unit
CPU	Computer Processing Unit
TPU	Tensor Processing Unit - special processor unit developed by Google for hardware acceleration of deep neural networks learning
JPEG	Commonly used method of lossy compression for digital images
LR	Learning rate - step size at each iteration while moving toward a minimum of a loss function
H&E	Hematoxylin and eosin staining - one of the principal tissue stains used in histology
GT	Ground Truth - reference annotation created by a human

A.2 Symbols

px	Picture element - smallest unit of a bitmap image

Appendix B
List of attached files on CD

`models`	Folder containing trained weights of the final model for each epoch.
`export`	Directory which contains all exported files during inference, can be deleted)
`scans`	Directory which contains exported crops and full segmentations
`overlays`	Contains generated masks and classes distribution for web-app
`pancreas-images`	Empty folder, default location for saving/loading .svs scan files
`qupath-program`	Directory which contains binaries for Windows and Linux version of QuPath program, which is used in the segmentation pipeline
`scripts`	Directory which contains useful Groovy scripts for QuPath exporting, the script "Export All.groovy" is used in the segmentation pipeline, do not change it or its location
`webserver`	Contains source-codes of web application) static/overlays (new segmentations from "/export/overlays" can be copied here to be displayed. When a scan is segmented through web-app, the new overlays are copied automatically and the class distribution file is updated.
`data512`	Directory which contains processed 512x512px tiles - training data of images and annotations from "PREPROCESS-DATA.ipynb" notebook
`INFERENCE.py`	Main Python script for segmentating new histology scans, this file is also used as a module for web server
`webserver/pancreas-server.py`	Script to run and configure web server for web-application. Change to "webserver" directory before starting it.
`PREPROCESS-DATA.ipynb`	Jupyter Notebook containing importing and preprocessing tiles for training.
`TRAIN.ipynb`	Jupyter Notebook containing training and evaluation source code.
`INFERENCE.ipynb`	Jupyter Notebook file for model configuration, training and evaluating

`requirements.txt`	Required pip packages to be installed
`utils.py`	Python file containing various Python functions, e.g. script for cutting the image to tiles
`RUN-WEB.bat`	Web-app executable file for Windows
`RUN-WEB.sh`	Web-app executable file for Linux
	Note: if using Python environment, switch to it first, or include it into the script
`horak-thesis-en-2021.tex`, `horak-thesis-en-2021.pdf`	This document in PDF and TEX formats.

Appendix C
Prerequisites Installation

C.1 Windows

- Install git (for installing openslide-python from git repository)
- Install CUDA and cuDNN drivers (for GPU accelerated support in Tensorflow)
- Install OpenSlide - download windows binaries and extract to desired folder
- Add OpenSlide bin and lib to PATH environment variable

```
C:\openslide-win64-20171122\bin
C:\openslide-win64-20171122\bin
```

- Install Python 3.6, 3.7 or 3.8, if not yet installed (Anaconda is recommended)
- If using Anaconda, create new environment and activate it, by:

```
conda create -n pancreas python=3.7
conda activate pancreas
```

- Install Python packages from requirements.txt, using:

```
pip install -r requirements.txt
```

C.2 Linux (Ubuntu)

- Install git (for installing openslide-python from git repository)

```
sudo apt install git
```

- Install CUDA and cuDNN drivers (for GPU accelerated support in Tensorflow)
- Install OpenSlide using:

```
sudo apt install openslide-tools
```

- Install Python 3.6, 3.7 or 3.8, if not yet installed (Anaconda is recommended)
- If using Anaconda, create new environment and activate it, by:

```
conda create -n pancreas python=3.7
conda activate pancreas
```

- Install Python packages from requirements.txt, using:

```
pip install -r requirements.txt
```

Notes When using Python of version 3.8 and higher you may get an error if installing tensorflow-gpu of version $\leq$ 2.0. Newer versions of tensorflow should be compatible as well. Therefore you can replace the row "tensorflow-gpu$\leq$2.0" with just "tensorflow" in the file requirements.txt and run `pip install -r requirements.txt` again