Development of a guided tagging tool for Whole Slide Images

Master Thesis

Submitted in partial fulfillment of the requirements for the degree of

Master of Science (M.Sc.)
in Applied Computer Science

at the

Berlin University of Applied Sciences (HTW)

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

First Supervisor: Prof. Dr. Peter Hufnagl

Second Supervisor: Diplom Informatiker Benjaming Voigt

Submitted by:

Sascha Nawrot (B.Sc.)

Berlin, August 22, 2016

Preface

Hello, this is the preface

Abstract

This is the abstract.

Contents

Chapter 1

Introduction

1.1 Motivation

The medical discipline of pathology is in a digital transformation. Instead of looking at tissue samples through the means of traditional light microscopy, it now is possible to digitalize those samples. This digitalization is done with the help of a so called slide scanner. The result of such an operation is a*whole slide image* (WSI) [5]. The digital nature of WSIs opens the door to the realm of image processesing and analysis which yields certain benefits, such as the use of image segmentation and registration methods to support the pathologist in his/her work.

A very promising novel approach to image analysis is the use of *neural networks*[1]. These are a group of models inspired by our current understanding of biological NN. In them, regardless if artificial or biological, many neurons are interconnected with each other. The construct of interconnected neurons is what we consider a NN. Each single one of those neurons has input values and an output value. Once the input reaches a certain trigger point, the cell in the neuron sends a signal as output. The connections between the neurons are weighted and can dampen or strengthen a signal. Because of this, old pathways can be blocked and new ones created. In other words, a NN is capable of "learning" [35]. This is a huge advantage compared to other software models. While certain problems are "easier" to solve in a sequential, algorithmic fashion (say an equation or the towers of hanoi), certain problems (e.g. image segmentation or object recognition) are very complex, so that new approaches are needed, while other problems can't be solved algorithmic at all. With the use of adequate training samples, a NN can learn to solve a problem, much like a human.

Their use, especially in the area of image classification and object recognition, enabled big breakthroughs in the recent past. Karpathy and Fei-Fei, for example, created a NN that is capable of describing an image or a scene with

[1]See chapter 2.2

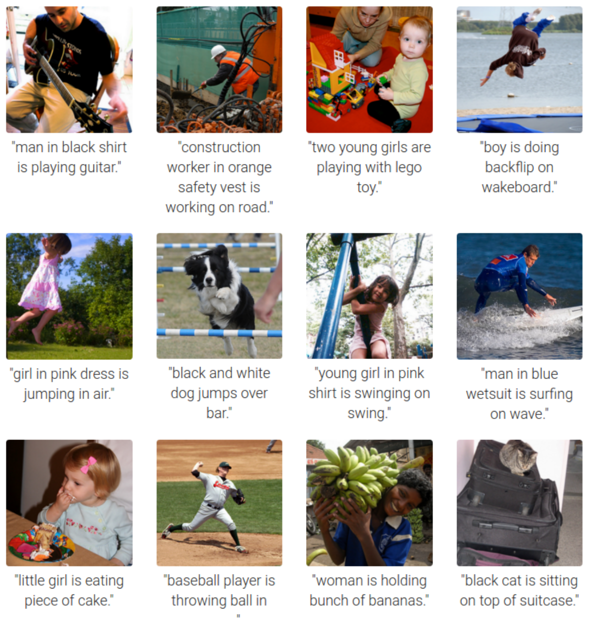
"man in black shirt is playing guitar."
"construction worker in orange safety vest is working on road."
"two young girls are playing with lego toy."
"boy is doing backflip on wakeboard."
"girl in pink dress is jumping in air."
"black and white dog jumps over bar."
"young girl in pink shirt is swinging on swing."
"man in blue wetsuit is surfing on wave."
"little girl is eating piece of cake."
"baseball player is throwing ball in "
"woman is holding bunch of bananas."
"black cat is sitting on top of suitcase."

Figure 1.1: Example results of the in [1] introduced model (source: `http://cs.stanford.edu/people/karpathy/deepimagesent/`)

written text [1] (see fig. 1.1 for a selection of examples).

There is enormous potential in the use of NN in the digital pathology as well, but to transfer these models and technologies, certain hindrances must be overcome. One of those hindrances is the need for proper training samples. While generally there are large amounts of WSIs (e.g. publicly available at the Cancer Genome Atlas[2]), most of them won't be usable without further preparation as a training sample. One way of preparing them are annotations. These can be added to the WSIs, stored and later used for training. The result of such an approach could be similar to the one of [1], but with a medical context instead of daily situations.

Therefore the goal of this thesis is to give tools into the hands of pathologists and data scientists to annotate WSIs and save those annotations in such a way that they will be usable later in combination with NN.

1.2 Research Objective

The objective of this thesis is the conceptualization and implementation of tools to prepare WSIs for the further use as training samples in NN. To achieve this,

[2]`https://gdc-portal.nci.nih.gov/`

a process chain with all the necessary steps needs to be established. The chain consists of the following tasks:

(A) open WSI with a viewer tool

(B) make that viewer tool capable of creating, managing and saving annotations

(C) make made annotations usable as training samples for NN

There is no standardized WSI file format [5]. Hence, slide scanner vendors developed their own proprietary solutions. This either leads to

(i) locking-in on a specific vendor or

(ii) separate handling of each proprietary format

(i) would render the whole process chain vendor specific, limiting it's use drastically. (ii) would not render the process chain vendor specific but call for a lot of extra work, due to the separate handling of different formats. Luckily, open file formats have been specified. According to [5], those are:

- JPEG2000
- TIFF
- Deep Zoom Images (DZI)
- DICOM (supplement 145), without reference implementation as of yet [5]

Therefore, to achieve (A), the first step of the process chain is to establish a tool with which WSIs of various formats can be turned into one of those open ones. This way, neither (i) nor (ii) will arise as a problem.

To achieve (A) and (B), it is also necessary to deploy a graphical user interface (GUI), that not only makes it possible to open and view a WSI (A), but also enables the user to annotate the WSI, as well as manage made annotations (B).

To achieve (C), another tool needs to be established, that is capable of turning saved annotations into training samples which are prepared for a further use in NN.

In summary: to reach the research objective of this thesis, tools to achieve the following tasks need to be established:

(a) conversion of various WSI formats into an open format

(b) annotation of WSIs and management thereof

(c) extracting and preparing annotations as training samples for later use in NN

1.3 About this thesis

This thesis contains 6 chapters. *Chapter 1 - Introduction* and *2 - Background* address the scope, background and vocabulary of this thesis. The chapters 3 - 5 are directly concerning themselves with 1.2(a) - 1.2(c). They correspond as follows:

(a): *chapter 3 - Conversion Service*

(b): *chapter 4 - Annotation Service*

(c): *chapter 5 - Tesselation Service*

Chapter 6 - Conclusion will discuss and conclude the findings of the aforementioned chapters.

Chapter 2

Background

2.1 Whole Slide Image Formats

As the size of an acquired raw, uncompressed WSI is massive[1], file formatting is required to mitigate the enormous amounts of information. Since there is no standardized format for WSIs, vendors came up with their own, proprietary solutions, which vary greatly [5]. Efforts of standardization are being made through the *Digital Imaging and Communications in Medicine* (DICOM) Standard [10].

Usually, WSI files are stored as a multitude of single images, spanning multiple folders and different resolutions. Those files are used to construct a so called *image pyramid* [18] (see fig. 2.1 and subsection 2.1.1).

2.1.1 DICOM Supplement 145

[19] describes DICOM as follows:

> "Digital Imaging and Communications in Medicine (DICOM), synonymous with ISO (International Organization for Standardization) standard 12052, is the global standard for medical imaging and is used in all electronic medical record systems that include imaging as part of the patient record."

Before *Supplement 145: Whole Slide Microscopic Image IOD and SOP Classes*, the DICOM Standard did not address standardization of WSI. Among others, the College of American Pathologist's Diagnostic Intelligence and Health Information Technology Committee is responsible for the creation and further advancement of this supplement [19].

It addresses every step involved in creating WSIs: image creation, acquisition, processing, analyzing, distribution, visualization and data management

[1] A typical 1,600 megapixel slide requires about 4.6 GB of memory on average [18]. The size of a H&E (hematoxylin and eosin) stained slide ranges typically from 4 to 20 GB [19].

[10]. It impacted the way how data is stored greatly [19], due to the introduction of a pyramid image model [10] (see fig. 2.1).

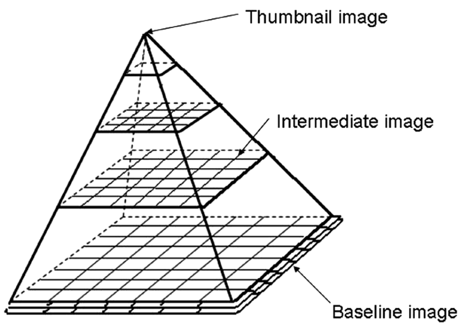

Figure 2.1: DICOMs image pyramid (source: [19])

The image pyramid model facilitates rapid zooming and reduces the computational burden of randomly accessing and traversing a WSI [19], [20]. This is made possible by storing an image in several precomputed resolutions, with the highest resolution sitting at the bottom (called the *baseline image*) and a thumbnail or low power image at the top (compare fig. 2.1) [10]. This creates a pyramid like stack of images, hence the name "pyramid model". The different resolutions are referred to as *layers* [10] or *levels* [19] respectively.

Each level is tessellated into square or rectangular fragments, called tiles, and stored in a two dimensional array [18].

Because of this internal organization, the tiles of each level can be retrieved and put together separately, to either form a subregion of the image or show it entirely. This makes it easy to randomly access any subregion of the image without loading large amounts of data [19].

2.1.2 Proprietary Formats

Vendors of whole slide scanners implement their own file formats, libraries and viewers (see tab. 2.1 for a list of vendors and their formats). Because of this, they can focus on the key features and abilities of their product. This generally leads to a higher usability, ease-of-use and enables highly tailored customer support. Furthermore, in comparison to open source projects, the longevity of proprietary software is often higher [29].

vendor	**formats**
Aperio	SVS, TIF
Hamamatsu	VMS, VMU, NDPI
Leica	SCN
3DHistech/Mirax	MRXS
Philips	TIFF
Sakura	SVSLIDE
Trestle	TIF
Ventana	BIF, TIF

Table 2.1: File formats by vendor

Since the proprietary formats have little to no documentation, most of the information presented here was reverse engineered in [15] and [39]. All proprietary formats listed here implement a modified version of the pyramid model introduced in 2.1.1.

Aperio

The SVS format by Aperio is a TIFF-based format, which comes in a single file [15]. It has a specific internal organization in which the first image is the baseline image, which is always tiled (usually with 240x240 pixels). This is followed by a thumbnail, typically with dimensions of about 1024x768 pixels. The thumbnail is followed by at least one intermediate pyramid image (compare fig. 2.1), with the same compression and tile organization as the baseline image [39]. Optionally, there may be a slide label and macro camera image at the end of each file [39].

Hamamatsu

Hamamatsu WSIs come in 3 variants:

(1) VMS

(2) VMU

(3) NDPI

(1) and (2) consist of an index file ((1) - [file name].vms, (2) - [file name].vmu) and 2 or more image files. In the case of (2), there is also an additional optimization file. (3) consists of a single TIFF-like file with custom TIFF tags. While (1) and (3) contain jpeg images, (2) contains a custom, uncompressed image format called *NGR*[2] [39].

The random access support for decoding parts of jpeg files is poor. To get around this, so called *restart markers*[3] are used to create virtual slides [15]. The

[2]For more information on NGR, consult `http://openslide.org/formats/hamamatsu/`

[3]Restart markers were originally designed for error recovery. The markers allow the decoder to resynchronize at set intervals throughout the image [15].

markers are placed at regular intervals. The offset of every marker is specified in different manner. In the case of (1), it can be found in the index file. In the case of (2), the optimization file holds the information and in the case of (3), a TIFF tag contains the offset [39].

Leica

SCN is a single file format based on BigTIFF, additionally .scn provides a pyramidal thumbnail image. [15].

The first TIFF directory has a tag called "ImageDescription" which contains an XML document that defines the internal structure of the WSI [39].

Leica WSIs are structured as a collection of images, each of which has multiple pyramid levels. While the collection only has a size, images have a size and position, all measured in nanometers. Each dimension has a size in pixels, an optional focal plane number, and a TIFF directory containing the image data. Fluorescence images have different dimensions (and thus different TIFF directories) for each channel [39].

Brightfield slides have at least two images: a low-resolution macro image and one or more main images corresponding to regions of the macro image. Fluorescence slides can have two macro images: one brightfield and one fluorescence [39].

3DHistech/Mirax

MRXS is a multi-file format with very complicated metadata in a mixture of text and binary formats. Images are stored as either JPEG, PNG or BMP [15]. The poor handling of random access is also applicable to PNG. Because of this, multiple images are needed to encode a single slide image. To avoid having many individual files, images are packed into a small number of data files. An index file provides offsets into the data files for each required piece of data. [39].

A 3DHistech/Mirax scanner take images with an overlap. Each picture taken is then tessellated without an overlap. Therefore, they only occur between taken pictures [39].

The generation of the image pyramid differs from the process described in 2.1.1. To create the n^{th} level, each image of the $n^{th} - 1$ level is divided by 2 in each dimension and then concatenated into a new image. Where the $n^{th} - 1$ level had 4 images in 2x2 neighborhood, the n^{th} level will only have 1 image. This process has no regards for overlaps. Thus, overlaps may occur in the higher levels of the image pyramid [39].

Philips

Philips' TIFF is an export from the native iSyntax format. An XML document with the hierarchical structure of the WSI can be found over the *ImageDescription* tag of the first TIFF directory. It contains key-value pairs based on DICOM tags [39].

Slides with multiple regions of interest are structured as a single image pyramid enclosing all regions. Slides may omit pixel data for TIFF tiles not in an ROI. When such tiles are downsampled into a tile that does contain pixel data, their contents are rendered as white pixels [39].

Label and macro images are stored either as JPEG or as stripped TIFF directories.

Sakura

WSIs in the SVSLIDE format are SQLite 3 database files. Their tables contain the metadata, associated images and tiles in the JPEG format. The tiles are addressed as (focal plane, downsample, level-0 X coordinate, level-0 Y coordinate, color channel). Additionally, each color channel has a separate grayscale image [39].

Trestle

Trestles TIF is a single-file TIFF. The WSI has the standard pyramdic scheme and tessellation. It contains non-standard metadata and overlaps, which are specified in additional files. The first image in the TIFF file is the baseline image. Subsequent images are assumed to be consecutive levels of the image pyramid with decreasing resolution [39].

Ventana

Ventanas WSIs are single-file BigTIFF images, organized in the typical pyramidical scheme. The images are tiled and have non-standard metadata, as well as overlaps. They come with a macro and a thumbnail image [39].

2.1.3 Open Formats

As mentioned in 2.1.2, proprietary formats typically come without much or any documentation. Furthermore, a vendors viewer is usually the only way of viewing WSIs of a particular format. This creates a vendor lock-in, where users can't take advantage of new improvements offered by other vendors. Furthermore, most viewers only provide support for Windows platforms. While, in a clinical setting, Windows may dominate the market, a significant amount of users in medical research prefer Linux or Mac OS X [15]. The use of mobile platforms, such as iOS or Android tablets may also have a great influence of the work flow in the future. Some vendors try to compensate for this fact with a server-based approach, which hurts performance by adding a network round-trip delay on every digital slide operation [15].

To compensate those issues, open image formats have been suggested, which will be discussed further in the following subsections.

Deep Zoom Images

The Deep Zoom Image (DZI) format is an XML-based file format, developed and maintained by Microsoft [37]. A DZI is a pyramidcal, tiled image (see fig. 2.2), similar to the one described in 2.1.1 (compare fig. 2.1 and 2.2), with two exceptions:

1. the baseline image is referred to as the highest level, instead of the lowest; this either turns the image pyramid or its labeling upside down
2. tiles are always square, with the exception of the last column/row

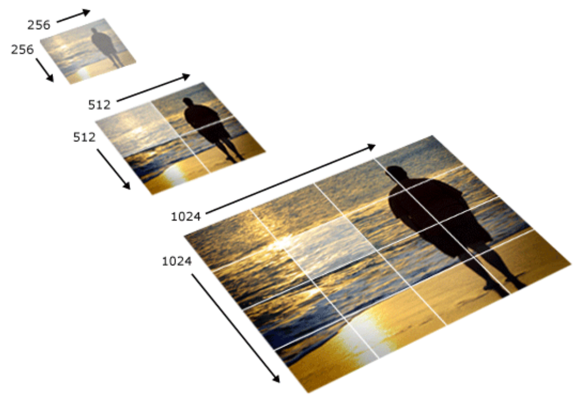

Figure 2.2: DZI pyramid model example (source: [37])

A DZI consists of two main parts [37]:

(1) a describing XML file ([file name].dzi) with informations about:

- the format of single tiles (e.g. JPEG or PNG)
- overlap between tiles
- size of tiles
- height and width of baseline image

(2) a directory ([file name]_files) with image tiles of the specified format

(1) and (2) are stored "next" to each other, so that there are 2 spearate files. (2) contains sub directories, one for each level of the image pyramid. The baseline image of a DZI is in the highest level. Each level is tessellated into as many tiles necessary to go over the whole image, with each tile having the size specified in the XML file. If the image size is no multiple of the specified tile size, the width of the n^{th} column of tiles will be ($width \mod tile\ size$) pixels. Equally, the height of the m^{th} row will be ($height \mod tile\ size$) pixels. Thus, the outermost right bottom tile $t_{n,m}$ will be of ($width \mod tile\ size$) x ($height \mod tile\ size$) pixels.

International Image Interoperability Framework

The International Image Interoperability Framework (IIIF) is the result of a cooperation between The British Library, Stanford University, the Bodleian Libraries[4], the Bibliothèque Nationale de France, Nasjonalbiblioteket[5]], Los Alamos National Laboratory Research Library and Cornell University [6]. Version 1.0 was published in 2012.

IIIFs goal is to collaboratively produce an interoperable technology and community framework for image delivery [27]. To achieve this, IIIF tries to:

(1) give scholars access to image-based resources around the world

(2) define a set of common APIs to support interoperability between image repositories

(3) develop and document shared technologies (such as image servers and web clients), that enable scholars to view, compare, manipulate and annotate images

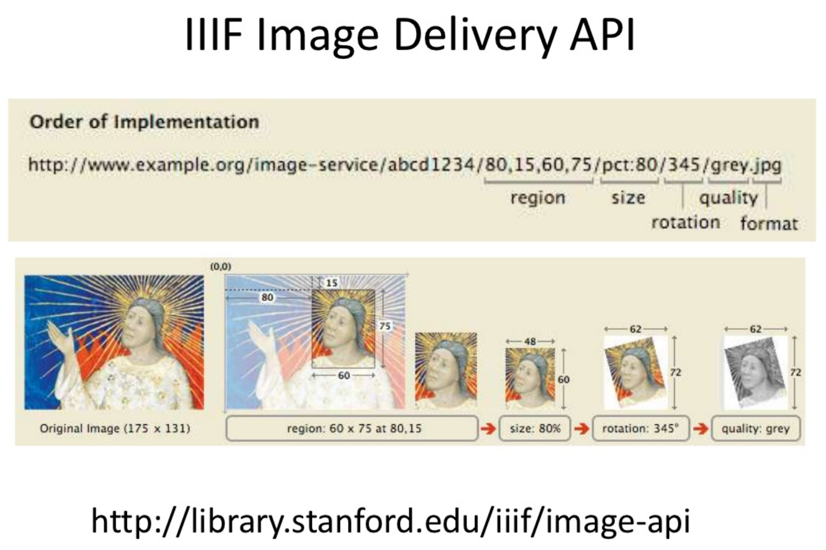

Figure 2.3: Example of iiif request (source:`http://www.slideshare.net/Tom-Cramer/iiif-international-image-interoperability-framework-dlf2012?ref=https://www.diglib.org/forums/2012forum/transcending-silos-leveraging-linked-data-and-open-image-apis-for-collaborative-access-to-digital-facsimiles/`)

[4]Oxford University
[5]National Library of Norway

The relevant part for this thesis is (2), especially the image API [16]. It specifies a web service that returns an image in response to a standard web request. The URL can specify the region, size, rotation, quality and format of the requested image (see fig. 2.3). Originally intended for resources in digital image repositories maintained by cultural heritage organizations, the API can be used to retrieve static images in response to a properly constructed URL [26]. The URL scheme looks like this [16]:

```
{scheme}://{server}{/prefix}/{identifier}/{region}/{size}/{rotation
    }/{quality}.{format}
```

The *region* and *size* parameters are of special interest for this thesis[6]. With them, it is possible to request only a certain region of an image in a specified size.

The region parameter defines the rectangular portion of the full image to be returned. It can be specified by pixel coordinates, percentage or by the value "full" (see tab. 2.2 and fig. 2.4).

Form	**Description**
full	The complete image is returned, without any cropping.
x,y,w,h	The region of the full image to be returned is defined in terms of absolute pixel values. The value of x represents the number of pixels from the 0 position on the horizontal axis. The value of y represents the number of pixels from the 0 position on the vertical axis. Thus the x,y position 0,0 is the upper left-most pixel of the image. w represents the width of the region and h represents the height of the region in pixels.
pct:x,y,w,h	The region to be returned is specified as a sequence of percentages of the full image's dimensions, as reported in the Image Information document. Thus, x represents the number of pixels from the 0 position on the horizontal axis, calculated as a percentage of the reported width. w represents the width of the region, also calculated as a percentage of the reported width. The same applies to y and h respectively. These may be floating point numbers.

Table 2.2: Valid values for *region* parameter (source: [16])

If the request specifies a regions whose size extends beyond the actual size of the image, the response should be a cropped image, instead of an image with added empty space. If the region is completely outside of the image, the response should be a 404 http status code [16].

[6]For detailed information on all parameters see the official API: `http://iiif.io/api/image/2.0`

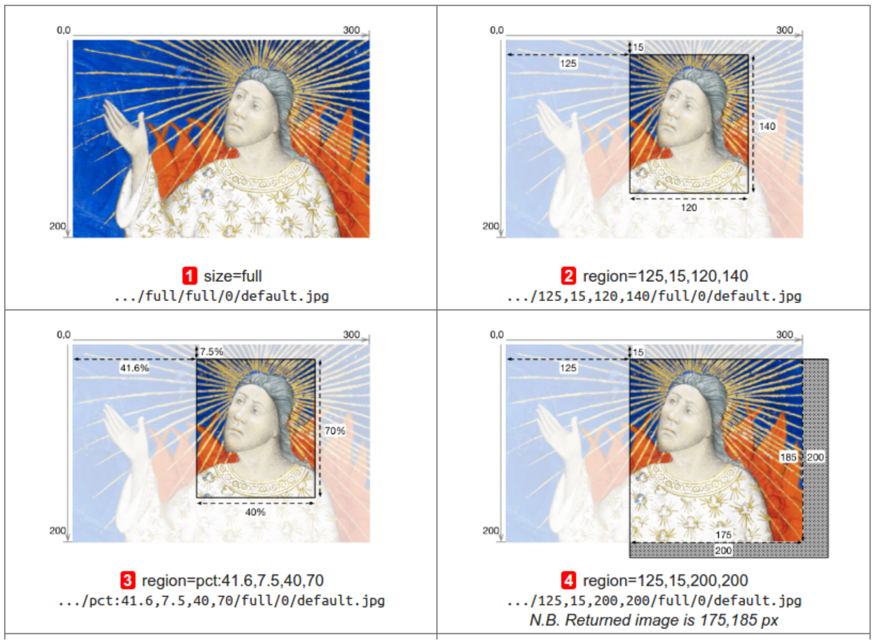
0,0
300
200
1 size=full
.../full/full/0/default.jpg
15
125
140
120
2 region=125,15,120,140
.../125,15,120,140/full/0/default.jpg
7.5%
41.6%
70%
40%
3 region=pct:41.6,7.5,40,70
.../pct:41.6,7.5,40,70/full/0/default.jpg
185
200
175
4 region=125,15,200,200
.../125,15,200,200/full/0/default.jpg
N.B. Returned image is 175,185 px

Figure 2.4: Results of IIIF request with different values for region parameter (source: [16])

If a region was extracted, it is scaled to the dimensions specified by the size parameter (see tab. 2.3 and fig. 2.5).

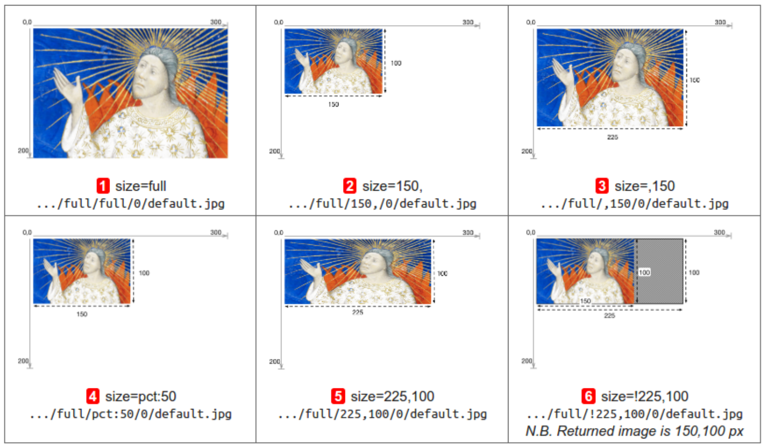
1 size=full
.../full/full/0/default.jpg
2 size=150,
.../full/150,/0/default.jpg
3 size=,150
.../full/,150/0/default.jpg
4 size=pct:50
.../full/pct:50/0/default.jpg
5 size=225,100
.../full/225,100/0/default.jpg
6 size=!225,100
.../full/!225,100/0/default.jpg
N.B. Returned image is 150,100 px

Figure 2.5: Results of IIIF request with different values for size parameter (source: [16])

If the resulting height or width equals 0, then the server should return a 400 http status code. Depending on the image server, scaling above the full size of the extracted region may be supported [16].

Form	Description
full	The extracted region is not scaled, and is returned at its full size.
w,	The extracted region should be scaled so that its width is exactly equal to w, and the height will be a calculated value that maintains the aspect ratio of the extracted region.
,h	The extracted region should be scaled so that its height is exactly equal to h, and the width will be a calculated value that maintains the aspect ratio of the extracted region.
pct:n	The width and height of the returned image is scaled to n% of the width and height of the extracted region. The aspect ratio of the returned image is the same as that of the extracted region.
w,h	The width and height of the returned image are exactly w and h. The aspect ratio of the returned image may be different than the extracted region, resulting in a distorted image.
!w,h	The image content is scaled for the best fit such that the resulting width and height are less than or equal to the requested width and height. The exact scaling may be determined by the service provider, based on characteristics including image quality and system performance. The dimensions of the returned image content are calculated to maintain the aspect ratio of the extracted region.

Table 2.3: Valid values for *size* parameter (source: [16])

To use the IIIF API, a compliant web server must be deployed. There are 2 alternatives to set up a new one [26], [27]:

- **Loris**, an open source image server based on python that supports the IIIF API versions 2.0, 1.1 and 1.0. Supported image formats are JPEG, JPEG2000 and TIFF.

- **IIPImage Server**, an open source Fast CGI module written in C++, that is designed to be embedded within a hosting web server such as Apache, Lighttpd, MyServer or Nginx. Supported image formats are JPEG2000 and TIFF.

OpenStreetMap/Tiled Map Service

OpenStreetMap (OSM) is a popular tile source used in many online geographic mapping specifications [26]. It's a community driven alternative to services such as Google Maps. Information is added by users via aerial images, GPS devices and field maps. All OSM data is classified as *open data*, meaning that it can be used anywhere, as long as the OSM Foundation is credited [23].

Tiled Map Service (TMS) is a tile scheme developed by the Open Source Geospatial Foundation (OSGF) [26] and specified in [22]. The OSGF is a non-profit organization whose goal it is to support the needs of the open source geospatial community. TMS provides access to cartographic maps of geo-referenced data. Access to these resources is provided via a "REST" interface, starting with a root resource describing available layers, then map resources with a set of scales, then scales holding sets of tiles [22].

Both, OSM and TMS, offer zooming images, which in general, have the functionality necessary, to be of use for this thesis. Unfortunately, the are also highly specialized on the needs of the mapping community, which makes them unattractive choices for the research objective of this thesis respectively.

JPEG 2000

[34] describes the image compression standard JPEG 2000 as follows:

> "JPEG 2000 is an image coding system that uses state-of-the-art compression techniques based on wavelet technology. Its architecture lends itself to a wide range of uses from portable digital cameras through to advanced pre-press, medical imaging and other key sectors."

It incorporates a mathematically lossless compression mode, in which the storage requirement of images can be reduced by an average of 2:1. On top of that, there is a visually lossless compression rate[7] of between 10:1 to 20:1 [30]. JPEG 2000 code streams offer mechanisms to support random access at varying degrees of granularity. It is possible to store different parts of the same picture using different quality [9].

In the compression process, JPEG 2000 partitions an image into rectangular and non-overlapping tiles of equal size (except for tiles at the image borders). The tile size is arbitrary and can be as large as the original image itself (resulting in only one tile) or as small as a single pixel. Furthermore, the image gets decomposed into a multiple resolution representation [36].

This creates a tiled image pyramid, similar to the one described in 2.1.1.

The encoding-decoding process of JPEG 2000 is beyond the scope of this thesis. Therefore, it is recommended to consult either [30] for a quick overview or [36] for an in depth guide.

[7]At *At visually lossless compression rates, even a trained a can't see the difference between original and compressed version [30].*

TIFF/BigTIFF

The Tagged Image File Format (TIFF) consists of a number of corresponding key-value pairs (e.g. *ImageWidth* and *ImageLength*, who describe the width and length of contained the image), called *tags*. One of the core features of this format is that it allows for the image data to be stored in tiles [14].

Each tiles offset is saved in an image header, so that efficient random access to any tile is granted. The original specification demands a use of 32 bit file offset values, limiting the maximum offset to 2^{32}. This constraint limits the file size to be below 4 GB [14].

This constraint lead to the development of BigTIFF. The offset values were raised to a 64 bit base, limiting the maximum offset to 2^{64}. This results in an image size of up to 18,000 peta bytes [11].

TIFF and BigTIFF are capable of saving images in multiple resolutions. Together with the feature of saving tiles, the image pyramid model described in 2.1.1 can be taken advantage of [12].

2.2 Short Introduction to Neural Networks

The objective of this thesis ultimately is to create training samples for NN[8]. Before going into other details, it is necessary to clarify what NN are, how they work, why they need training samples and what they use them for[9].

Artificial Neural Networks (NN) are a group of models inspired by Biological Neural Networks (BNN) . BNNs can be described as an interconnected web of neurons (see fig. 2.6), whose purpose it is to transmit information in the form of electrical signals. A neuron receives input via dendrites and send output via axons [43]. An average human adult brain contains about 10^{11} neurons. Each of those receives input from about 10^4 other neurons. If their combined input is strong enough, the receiving neuron will send an output signal to other neurons [8].

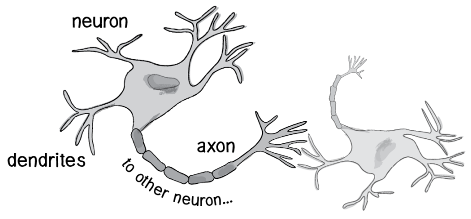

Figure 2.6: Neuron in a BNN (source: [43])

[8]compare chap. 1.2

[9]It should be mentioned that the scope of NN is huge and worth a whole thesis by themselves. This is nothing more than a short introduction and further consultation of literature (e.g. [4], [8], [17], [35], [43]) is highly recommended.

NN are much simpler in comparison[10], but generally work in the same fashion.

One of the biggest strengths of a NN, much like a BNN, is the ability to adapt by learning[11]. This adaption is based on *weights* that are assigned to the connections between single neurons. Fig 2.7 shows an exemplary NN with neurons and the connections between them.

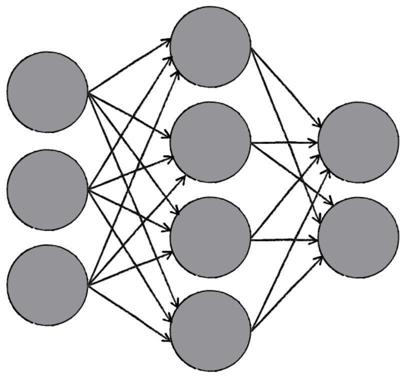

Figure 2.7: Exemplary NN (source: [43])

Each line in fig, 2.7 represents a connection between 2 neurons. Those connections are a one-directional flow of information, each assigned with a specific weight. This weight is a simple number that is multiplied with the incoming/outgoing signal and therefore weakens or enhances it. They are the defining factor of the behavior of a NN. Determining those values is the purpose of training a NN [8].

According to [43], some of the standard use cases for NN are:

- Pattern Recognition
- Time Series Prediction
- Signal Processing Perceptron
- Control
- Soft Sensors
- Anomaly Detection

2.2.1 Methods of Learning

There are 3 general strategies when it comes to the training of a NN [8]. Those are:

[10]Usually, they don't have much more than a few dozen neurons [8].

[11]As humans, NN learn by training [43].

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning (a variant of Unsupervised Learning [41])

Supervised Learning is a strategy that involves a training set to which the correct output is known, as well as an observing teacher. The NN is provided with the training data and computes its output. This output is compared to the expected output and the difference is measured. According to the error made, the weights of the NN are corrected. The magnitude of the correction is determined by the used learning algorithm [41].

Unsupervised Learning is a strategy that is required when the correct output is unknown and no teacher is available. Because of this, he NN must organize itself [43]. [41] makes a distinction between 2 different classes of unsupervised learning:

- reinforced learning
- competitive learning

Reinforced learning adjusts the weights in such a way, that desired output is reproduced. For example, a robot in a maze. If the robot can drive straight without any hindrances, it can associate this sensory input with driving straight (desired outcome). As soon as it approaches a turn, the robot will hit a wall (non-desired outcome). To prevent it from hitting the wall it must turn, therefore the weights of turning must be adjusted to the sensory input of being at a turn. Another example is *Hebbian learning*[12] [41].

In competitive learning, the single neurons compete against each other for the right to give a certain output for an associated input. Only one element in the NN is allowed to answer, so that other, competing neurons are inhibited [41].

2.2.2 The Perceptron

The perceptron was invented by Rosenblatt at the Cornell Aeronautical Laboratory in 1957 [42]. It is the computational model of a single neuron and as such, the simplest NN possible [43]. A perceptron consists of one or more inputs, a processor and a single output (see fig. 2.8) [42].

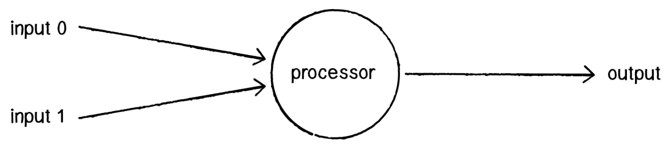

Figure 2.8: Perceptron by Rosenblatt (source: [43])

[12]see [41] for further information

This can be directly compared to the neuron in fig. 2.7, where:

- input = dendrites
- processor = cell
- output = axon

A perceptron is only capable of solving *linearly separable* problems, such as logical *AND* and *OR* problems. To solve non-linearly separable problems, more then one perceptron is required [42]. Simply put, a problem is linearly separable, if it can be solved with a straight line (see fig. 2.9), otherwise it is considered a non-linearly separable problem (see fig. 2.10).

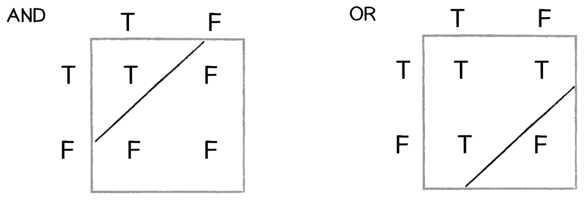

Figure 2.9: Examples for linearly separable problems (source: [43])

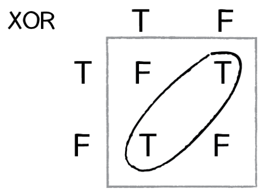

Figure 2.10: Examples for non-linearly separable problems (source: [43])

2.2.3 Multi-layered Neural Networks

To solve more complex problems, multiple perceptrons can be connected to from a more powerful NN. A single perceptron might not be able to solve *XOR*, but one perceptron can solve *OR*, while the other can solve $\neg AND$. Those two perceptrons combined can solve *XOR* [43].

If multiple perceptrons get combined, they create layers. Those layers can be separated into 3 distinct types [4]:

- input layer
- hidden layer
- output layer

A typical NN will have an input layer, which is connected to a number of hidden layers, which either connect to more hidden layers or, eventually, an output layer (see fig. 2.11 for a NN with one hidden layer).

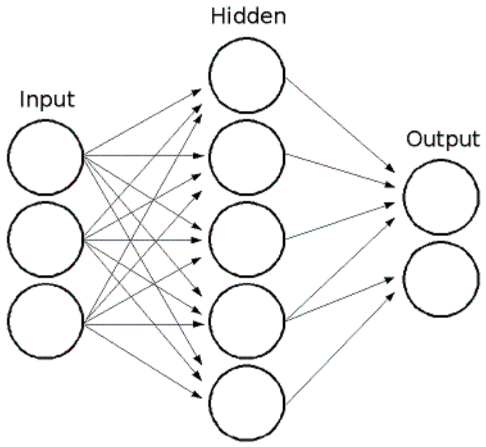

Figure 2.11: NN with multiple layers (source: `http://docs.opencv.org/2.4/_images/mlp.png`)

As the name suggests, the input layer gets provided with the raw information input. Depending on the internal weights and connections inside the hidden layer, a representation of the input information gets formed. At last, the output layer generates output, again based on the connections and weights of the hidden and output layer [4].

Training this kind of NN is much more complicated than training a simple perceptron, since weights are scattered all over the NN and its layers. A solution to this problem is called *backpropagation* [43].

Backpropagation

Training is an optimization process. To optimize something, a metric to measure has to be established. In the case of backpropagation, this metric is the accumulated output error of the NN to a given input[13]. There are several ways to calculate this error, with the *mean square error*[14] being the most common one [8].

Finding the optimal weights is an iterative process of the following steps:

1. start with training set of data with known output

[13]To do so, it is necessary to know the right answer. Therefore, backpropagation is part of the supervised learning process.

[14]Mean square error is the average of the square of the differences of two variables, in this case the expected and the actual output.

2. initialize weights in NN
3. for each set of input, feed the NN and compute the output
4. compare calculated with known output
5. adjust weights to reduce error

There are 2 possibilities in how to proceed. The first one is to compare results and adjust weights after each input/output-cycle. The second one is to calculate the accumulated error over a whole iteration of the input/output-cycle. Each of those iterations is known as an *epoch* [8].

2.3 Microservices

The following section elaborates on the concept of *Microservices* (MS), defining what they are, listing their pros and cons, as well as explaining why this approach was chosen over a monolithic approach. A monolithic software solution is described by [33] as follows:

> "[...] a monolithic application [is] built as a single unit. Enterprise Applications are often built in three main parts: a client-side user interface (consisting of HTML pages and javascript running in a browser on the user's machine) a database (consisting of many tables inserted into a common, and usually relational, database management system), and a server-side application. The server-side application will handle HTTP requests, execute domain logic, retrieve and update data from the database, and select and populate HTML views to be sent to the browser. This server-side application is a monolith - a single logical executable. Any changes to the system involve building and deploying a new version of the server-side application."

2.3.1 Definition

MS are an interpretation of the Service Oriented Architecture. The concept is to separate one monolithic software construct into several smaller, modular pieces of software [45]. As such, MS are a modularization concept. However, they differ from other such concepts, since MS are independent from each other. This is a trait, other modularization concepts usually lack [45]. As a result, changes in one MS don't bring up the necessity of deploying the whole product cycle again, but just the one service. This can be achieved by turning each MS into an independent process with its own runtime [33].

This modularization creates an information barrier between different MS. Therefore, if MS need to share data or communicate with each other, light weight communication mechanisms must be established, such as a RESTful API [40].

Even though MS are more a concept than a specific architectural style, certain traits are usually shared between them [40]. According to [40] and [33], those are:

(a) **Componentization as a Service:** bringing chosen components (e.g. external libraries) together to make a customized service

(b) **Organized Around Business Capabilities:** cross-functional teams, including the full range of skills required to achieve the MS goal

(c) **Products instead of Projects:** teams own a product over its full lifetime, not just for the remainder of a project

(d) **Smart Endpoints and Dumb Pipes:** each microservice is as decoupled as possible with its own domain logic

(e) **Decentralized Governance:** enabling developer choice to build on preferred languages for each component.

(f) **Decentralized Data Management:** having each microservice label and handle data differently

(g) **Infrastructure Automation:** including automated deployment up the pipeline

(h) **Design for Failure:** a consequence of using services as components, is that applications need to be designed so that they can tolerate the failure of single or multiple services

Furthermore, [3] defined 5 architectural constraints, which should help to develop a MS:

(1.) **Elastic**
The elasticity constraint describes the ability of a MS to scale up or down, without affecting the rest of the system. This can be realized in different ways. [3] suggests to architect the system in such a fashion, that multiple stateless instances of each microservice can run, together with a mechanism for Service naming, registration, and discovery along with routing and load-balancing of requests.

(2.) **Resilient**
This constraint is referring to the before mentioned trait (h) - *Design for Failure*. The failure of or an error in the execution of a MS must not impact other services in the system.

(3.) **Composable**
To avoid confusion, different MS in a system should have the same way of identifying, representing, and manipulating resources, describing the API schema and supported API operations.

(4.) **Minimal**
A MS should only perform one single business function, in which only semantically closely related components are needed.

(5.) **Complete**
A MS must offer a complete functionality, with minimal dependencies to other services. Without this constraint, services would be interconnected again, making it impossible to upgrade or scale individual services.

2.3.2 Advantages and Disadvantages

One big advantage of this modularization is that each service can be written in a different programming language, using different frameworks and tools. Furthermore, each microservice can bring along its own support services and data storages. It is imperative for the concept of modularization, that each microservice has its own storage of which it is in charge of [45].

The small and focused nature of MS makes scaling, updates, general changes and the deploying process easier. Furthermore, smaller teams can work on smaller code bases, making the distribution of know how easier [40].

Another advantage is how well MS plays into the hands of agile, scrum and continuous software development processes, due to their previously discussed inherent traits.

The modularization of MS doesn't only yield advantages. Since each MS has its own, closed off data management[15], interprocess communication becomes a necessity. This can lead to communicational overhead which has a negative impact on the overall performance of the system [45].

2.3.1(e) (*Decentralized Governance*) can lead to compatibility issues, if different developer teams chose to use different technologies. Thus, more communication and social compatibility between teams is required. This can lead to an unstable system which makes the deployment of extensive workarounds necessary [40].

It often makes sense to share code inside a system to not replicate functionality which is already there and therefore increase the maintenance burden. The independent nature of MS can make that very difficult, since shared libraries must be build carefully and with the fact in mind, that different MS may use different technologies, possibly creating dependency conflicts.

2.3.3 Conclusion

After consideration of the advantages and disadvantages of MS, the author decided in favor of using them. This is mainly due to the fact of working alone on the project, negating some of their inherent disadvantages:

[15]See 2.3.1(f) (*Decentralized Data Management*)

- Interprocess communication doesn't arise between the single stages of the process chain, since they have a set order[16]
- Different technologies may be chosen for the single steps of the process chain, however, working alone on the project makes technological incompatibilities instantly visible
- The services shouldn't share functionality, therefore there should be no need for shared libraries

This makes the advantages outweight the disadvantages clearly:

- different languages and technologies can be used for every single step of the process chain, making the choice of the most fitting tool possible
- WSIs take a heavy toll on memory and disk space due to their size; the use of MS allows each step of the chain to handle those issues in the most suitable way for each given step
- separating the steps of the process chain into multiple MS makes for a smaller and easier maintainable code base
- other bachelor/master students may continue to use or work on this project in the future, making the benefit of a small, easily maintainable code base twice as important
- the implementation of the project will happen in an iterative, continuous manner, which is easily doable with the use of MS

2.4 Process Chain

This section and its following subsections are dedicated to establish the process chain necessary to accomplish the research objectives stated in 1.2(a) - 1.2(c). The usual procedure would look as follows:

(1.) convert chosen WSI img_i^{wsi} to open format img_i^{cvrt}

(2.) open img_i^{cvrt} in a viewer V

(3.) annotate img_i^{cvrt} in V

(4.) persist annotations A_i on img_i^{cvrt} in a file $f_{(A_i)}$

(5.) create training sample ts_i by extracting the information of A_i in correspondence to img_i^{cvrt}

[16]E.g. it wouldn't make sense trying extract a training sample without converting or annotating a WSI first.

While it only makes sense to run (1.) once per img_i^{wsi} to create img_i^{cvrt}, steps (2.) - (4.) can be repeated multiple times, so that there is no need to finish the annotation of an image in one session. That makes it necessary to not only save but also load annotations. Therefore, the loading of already made annotations can be added as step (2.5). This also enables the user of editing and deleting already made annotations. Because of this, step (5.) also needs to be repeatable.

The single steps of the process chain will be sorted into semantic groups. Each group will be realized by its own MS, as stated in 2.3. The semantic groups are: conversion (1.), extraction (5.) and viewing and annotation (2. - 4.).

A MS will be introduced for each group in the following subsections(2.4.1 - 2.4.3). Those are:

- **Conversion Service**
  This service will be responsible of the conversion from img_i^{wsi} to img_i^{cvrt} (1.).

- **Annotation Service**
  This service will offer a GUI to view a img_i^{cvrt}, as well as make and manage annotations (2. - 4.)

- **Tessellation Service**
  This service will be responsible for extracting a ts_i from a given A_i and img_i^{cvrt} (5.).

2.4.1 Conversion Service

The devices which create WSIs, so called *whole slide scanners*, create images in various formats, depending on the producer and a lack of a defined standard [5]. The Conversion Service (CS) has the goal of converting those formats to an open format[17] (see fig. 2.12).

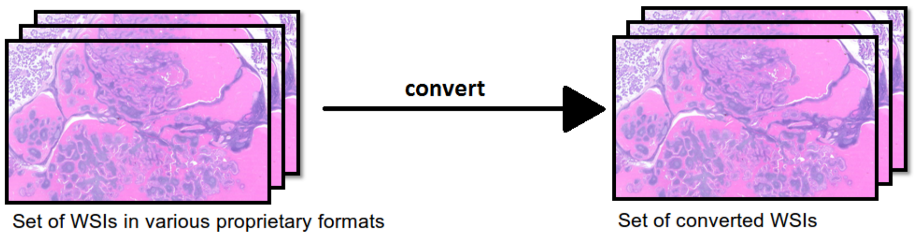

Figure 2.12: Visualization of the Conversion Service

[17]see chap. 2.1.2 for the reasons

Upon invocation, the CS will take every single WSI inside a given directory and convert it to a chosen open format. The output of each conversion will be saved in another specified folder. Valid image formats for conversion are:

- .bif
- .mrxs
- .ndpi
- .scn
- .svs
- .svslide
- .tif
- .tiff
- .vms
- .vmu

2.4.2 Annotation Service

As mentioned in 2.4, the Annotation Service (AS) will provide a graphical user interface (GUI) to view a WSI, make annotations and manage those annotations. This also includes persisting made annotations in a file (see fig. 2.13).

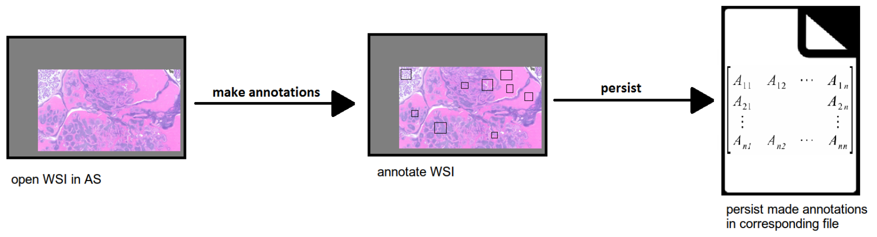

Figure 2.13: Visualization of the Annotation Service

The supplied GUI will offer different tools to help the user annotate the WSI, e.g. a ruler to measure the distance between 2 points. The annotations themselves will be made via drawing a contour around an object of interest and putting a specified label to that region. To ensure uniformity of annotations, labels will not be added in free text. Instead they will be selected from a predefined dictionary.

2.4.3 Tessellation Service

The task of the Tessellation Service (TS) is to extract annotations and their corresponding image data in such a fashion that they will become usable as training samples for NN.

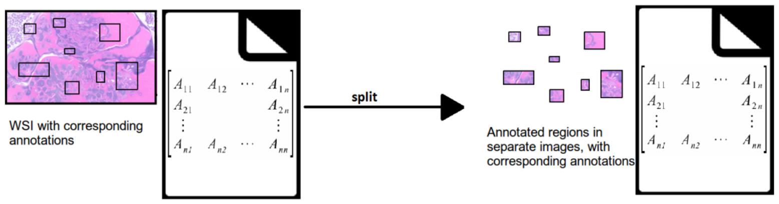
WSI with corresponding annotations
split
Annotated regions in separate images, with corresponding annotations

Figure 2.14: Visualization of the Tessellation Service

Let there be a WSI Img and a corresponding set of annotations A. The TS will achieve the extraction by iterating over every $a_i \in A$, creating a sub-image img_i which is the smallest bounding box around the region described by a_i (see fig. 2.14). To be used as training sample, the TS must keep up the correspondence between img_i and a_i.

Chapter 3

Conversion Service

3.1 Methodology

As stated in 2.1, there is no standardized format for WSIs. Supplement 145 of the DICOM standard tries to unify the whole process around WSIs, but vendors still push their proprietary formats. For the reasons mentioned in 2.1.3, it is necessary to establish a common format for all the WSIs which are subject to the process chain established in 2.4. Therefore, the goal of the CS is to convert WSIs of proprietary formats into a common open format.

To make the conversion as convenient and fast as possible, the CS should only have brief user interaction. For this purpose it will not have a GUI. Instead the CS will be implemented as a console script. Furthermore, the CS should be capable of converting multiple WSIs after one another, so that no restart is necessary between conversions. Therefore, the CS will take an input directory as parameter and convert all WSIs of valid format inside that directory. Another parameter will be the output folder, in which the converted DZIs are stored.

vendor	**formats**
Aperio	SVS, TIF
Hamamatsu	VMS, VMU, NDPI
Leica	SCN
3DHistech/Mirax	MRXS
Philips	TIFF
Sakura	SVSLIDE
Trestle	TIF
Ventana	BIF, TIF

Table 3.1: File formats by vendor

Tab. 3.1 gives an overview of file formats, sorted by vendor, which are viable as input for the conversion.

3.1.1 Selection of Image Format

tool	**description**	**image format**
Deep Zoom Composer	dekstop app for Windows	DZI
Image Composite Editor	panoramic image stitcher from Microsoft Research for the Windows desktop	DZI
DeepZoomTools.dll	.NET-library, comes with Deep Zoom Composer	DZI
deepzoom.py	Python	DZI
deepzoom	Perl utility	DZI
PHP Deep Zoom Tools	PHP	DZI
Deepzoom	PHP	DZI
DZT	an image slicing library and tool written in Ruby	DZI
MapTiler	desktop app for Windows, Mac, Linux	TMS
VIPS	command line tool and library for a number of languages	DZI
Sharp	Node.js, uses VIPS	DZI
MagickSlicer	shell script (Linux/Mac)	DZI
Gmap Uploader Tiler	C++	DZI
Node.js Deep Zoom Tools	Node.js, under construction	DZI
OpenSeaDragon DZI Online Composer	Web app (and PERL and PHP scripts)	DZI
Zoomable	service, offers embeds; no explicit API	DZI
ZoomHub	service, under construction	DZI
Kakadu	C++ library to encode or decode JPEG 2000 images	IIIF
PyramidIO	Java (command line and library)	DZI

Table 3.2: Overview of conversion options for zooming image formats (source: [26])

A format or service must be chosen as conversion target for the CS. Choices have been established in 2.1.3. These are:

(1) BigTIFF

(2) DZI

(3) IFFF

(4) JPEG 2000

(5) TMS/OMS

To convert a WSI, a conversion tool is needed. Tab. 3.2 shows a listing of possibilities for that purpose. Listed are the name of the tool, used technology and output format. It shows clearly, that DZI has a great variety of options, while the alternatives have little to none (Map Tiler for TMS, Kakadu for IFFF and none for the others).

Since the CS should only consist of brief user interaction and be as automated as possible, desktop and web applications are not valid as tools for conversion. This excludes *Deep Zoom Composer*, *MapTiler*, *OpenSeaDragon DZI Online Composer* and *Zoomable* as possible choices (therefore also excluding, next to other reasons[1], TMS as possible format).

One of the reasons not to use proprietary formats was the support of only certain operating systems, eliminating Windows-only tools. Those are *Image Composite Editor* and *DeepZoomTools.dll*.

Furthermore, reading the proprietary formats is a highly specialized task, eliminating most of the leftover choices: *deepzoom* [2], *DZT* [13], *sharp* [24], *MagickSlicer*, *Node.js Deep Zoom Tools*[2], *Gmap Uploader Tiler* [38], Zoomhub [25] and *PyramidIO* [21].

Kakadu can only encode and decode JPEG 2000 images [26], making it no valid choice either.

This leaves *deepzoom.py* and *VIPS*, both creating DZI as output. Through the use of OpenSlide, they are both capable of reading all proprietary formats stated in tab. 3.1 [39].

3.1.2 Deepzoom.py

Deepzoom.py[3] is a python script and part of Open Zoom[4]. It can either be called directly over a terminal or imported as a module in another python script. The conversion procedure itself is analogous for both methods.

If run in a terminal the call looks like the following:

```
$ python deepzoom.py [options] [input file]
```

[1]compare chap 2.1.3

[2]MagickSlicer and Node.js Deep Zoom Tools use ImageMagick to read images, which doesn't support any of the proprietary WSI formats [28].

[3]See `https://github.com/openzoom/deepzoom.py` for further details

[4]See `https://github.com/openzoom` for further details

The various options and their default values can be seen in tab. 3.3. If called without a designated output destination, deepzoom.py will save the converted DZI in the same directory as the input file.

option	**description**	**default**
-h	show help dialog	-
-d	output destination	-
-s	size of the tiles in pixels	254
-f	image format of the tiles	jpg
-o	overlap of the tiles in pixels (0 - 10)	1
-q	quality of the output image (0.0 - 1.0)	0.8
-r	type of resize filter	antialias

Table 3.3: Options for deepzoom.py

The resize filter is applied to interpolate the pixels of the image when changing its size for the different levels. Supported filters are:

- cubic
- biliniear
- bicubic
- nearest
- antialias

When used as module in another python script, deepzoom.py can simply be imported via the usual *import* command. To actually use deepzoom.py, a Deep Zoom Image Creator needs to be created. This class will manage the conversion process:

```
# Create Deep Zoom Image Creator
creator = deepzoom.ImageCreator(tile_size=[size],
  tile_overlap=[overlap], tile_format=[format],
  image_quality=[quality], resize_filter=[filter])
```

The options are analogous with the terminal version (compare tab. 3.3). To start the conversion process, the following call must be made within the python script:

```
# Create Deep Zoom image pyramid from source
creator.create([source], [destination])
```

Upon calling, the ImageCreator opens the input image img^{wsi} and accesses the information necessary to create the describing XML file for the DZI[5]. The needed number of levels is calculated next. For this, the bigger value of height or

[5]Compare chap. 2.1.3

width of img^{wsi} is chosen (see eq. 3.1) and then used to determine the number of levels lvl^{max} (see eq. 3.2) necessary.

$$max\_dim = max(height, width) \tag{3.1}$$

$$lvl^{max} = \lceil log_2(max\_dim) + 1 \rceil \tag{3.2}$$

Once lvl^{max} has been determined, a resized version img_i^{dzi} of img^{wsi} will be created for every level $i \in [0, lvl - 1]$. The quality of img_i^{dzi} will be reduced according to the value specified for -q/image_quality (see tab. 3.3). The resolution of img_i^{dzi} will be calculated with the *scale* function (see eq. 3.3) for both, height and width. Furthermore, the image will be interpolated with the specified filter (-r/resize_filter parameter, see tab. 3.3).

$$scale = \lceil dim * 0.5^{lvl^{max} - i} \rceil \tag{3.3}$$

Once img_i^{dzi} has been created, it will be tessellated into as many tiles of the specified size (-s/tile_size parameter, see tab. 3.3) and overlap (-o/tile_overlap parameter, see tab. 3.3) as possible. If the size of img^{wsi} in either dimension isn't a multiple of the tile size, the last row/column of tiles will be smaller by the amount of $(tile\ size - ([height \text{ or } width] \mod tile\ size))$ pixels.

Every tile will be saved as [column]_[row].[format] (depending on the -f/file_format parameter, see tab. 3.3) in a directory called according to the corresponding level i. Each one of those level directories will be contained within a directory called [filename]_files. The describing XML file will be persisted as [filename].dzi in the same directory as [filename]_files.

3.1.3 VIPS

VIPS (VASARI Image Processing System) describes itself as "[...] a free image processing system [...]" [7]. It includes a wide range of different image processing tools, such as various filters, histograms, geometric transformations and color processing algorithms. It also supports various scientific image formats, especially those needed by the CS[6] [7]. One of the strongest traits of VIPS is its speed and little data usage compared to other imaging libraries [32].

VIPS comes in two parts: the actual library (called libvips) and a GUI (called nip2). libvips offers interfaces for C, C++, python and the command line. The GUI will not be further discussed, since it is of no interest for the implementation of the CS.

VIPS speed and little data usage comes from a fully demand-driven image input/output system. While conventional imaging libraries queue their operations and go through them sequentially, VIPS awaits a final write command, before actually manipulating the image. All the queued operations will then be evaluated and molded into a few single operations. Thus, requiring no additional

[6]See chap. 2.2.1

disc space for intermediates and no unnecessary disc in- and output. Furthermore, if more than one CPU is available, VIPS will automatically evaluate in parallel [31].

As mentioned before, VIPS has a command line and python interface. In either case, a function called *dzsave(...)* will manage the conversion from a WSI to a DZI. A call in the terminal looks as follows:

```
$ vips dzsave [input] [output] [options]
```

When called, VIPS will take the image [input], convert it into a DZI and then save it to [output]. The various options and their default values can be seen in tab. 3.4.

option	**description**	**default**
layout	directory layout (allowed: dz, google, zoomify)	dz
overlap	tile overlap in pixels	1
centre	center image in tile	false
depth	pyramid depth	onepixel
angle	rotate image during save	d0
container	pyramid container type	fs
properties	write a properties file to the output directory	false
strip	strip all metadata from image	false

Table 3.4: Options for VIPS

A call in python does have the same parameters and default values. It looks like this:

```
1 image = Vips.Image.new_from_file(input)
2 image.dzsave(output[, options])
```

In line 1 the image gets opened and saved into *image*. While being opened, further operations could be done. The command in line 2 writes the processed image as DZI into the specified output location.

3.2 Implementation

As stated before, the CS should be implemented as a script.

The first iteration was a python script using deepzoom.py for the conversion. This caused severe performance issues. Out of all the image files in the test set[7], only one could be converted[8]. Other files were either too big, so the process would eventually be killed by the operating system, or exited with an IOError concerning the input file from the PIL imaging library.

The second iteration uses VIPS python implementation, which is not only faster than deepzoom.py, but also capable of converting all the given test images.

[7]See chap. 3.3

[8]CMU-3.svs from Aperio, see Appendix A

The script has to be called inside a terminal in the following fashion:

```
$ python ConversionService.py [input dir] [output dir]
```

It is mandatory to specify the input and output directory in the call, so the script knows where to look for images to convert and where to save the resulting DZIs.

Upon calling, the *main()* routine will be started, which orchestrates the whole conversion process. It looks as follows:

```
def main():
  path = checkParams()
  files = os.listdir(path)
  for file in files:
    print("----------------------------------------------")
    extLen = getFileExt(file)
    if(extLen != 0):
      print("converting " + file + "...")
      convert(path, file, extLen)
      print("done!")
```

checkParams() checks if the input parameters are valid and, if so, returns the path to the specified folder or exit otherwise. Furthermore, it will create the specified output folder, if it doesn't exist already. In the next step, the specified input folder will be checked for its content. *getFileExt(file)* looks up the extension of each contained file and will either return the length of the files extension or 0 otherwise. Each valid file will then be converted with the *convert(...)* function:

```
1  # convert image source into .dzi format and copies all header
2  # information into [img]_files dir as metadata.txt
3  # param path: directory of param file
4  # param file: file to be converted
5  # param extLen: length of file extension
6  def convert(path, file, extLen):
7    dzi = OUTPUT + file[:extLen] + "dzi"
8    im = Vips.Image.new_from_file(path + file)
9    # get image header and save to metadata file
10   im.dzsave(dzi, overlap=OVERLAP, tile_size=TILESIZE)
11   # create file for header
12   headerOutput = OUTPUT + file[:extLen-1] + "_files/metadata.txt"
13   bashCommand = "touch " + headerOutput
14   call(bashCommand.split())
15   # get header information
16   bashCommand = "vipsheader -a " + path + file
17   p = subprocess.Popen(bashCommand.split(), stdout=subprocess.PIPE,
        stderr=subprocess.PIPE)
18   out, err = p.communicate()
19   # write header information to file
20   text_file = open(headerOutput, "w")
21   text_file.write(out)
22   text_file.close()
```

The name for the new DZI file will be created from the original file name, however, the former extension will be replaced by "dzi" (see line 7). *OUTPUT* specifies the output directory which the file will be saved to. Next, the image file will be opened with Vips' Image class. Afterwards, *dzsave(...)* will be called, which handles the actual conversion into the dzi file format. *OVERLAP* and *TILESIZE* are global variables which describe the overlap of the tiles and their respective size. Their values are 0 (*OVERLAP*) and 256 (*TILESIZE*). The output will be saved to the same folder as ConversionService.py is operating from, plus "/dzi/[*OUTPUT*]/".

When a WSI gets converted into DZI by the CS, most of the image header information is lost. To counteract this, a file *metadata.txt* is created in the [name]_files directory, which serves as container for the header information of the original WSI (see line 12 and 13).

The console command *vipsheader -a* is responsible for extracting the header information (see line 17 - 18). The read information (*out* in line 18) is then written into the *metadata.txt* file (Line 20 - 22).

3.3 Test

Data is needed to test the correct functionality of the CS. OpenSlide offers a selection of freely distributable WSIs[9], which can be used for that purpose.

Since the WSIs are of enormous size, they are not delivered via the CS repository[10]. Instead they need to be downloaded separately from the OpenSlide homepage. For a complete listing of the used test data see Appendix A.

3.3.1 Setup

To create a controlled environment for the test, a new directory will be created, called *CS_test*. A copy of ConversionService.py as well as a directory containing all the test WSIs (called *input*) will be placed in that directory.

Input contains the following slides:

(1) CMU-2 (Aperio, .svs)

(2) CMU-1 (Generic Tiled tiff, .tiff)

(3) OS-3 (Hamamatsu, .ndpi)

(4) CMU-2 (Hamamatsu, .vms)

(5) Leica-2 (Leica, .scn)

(6) Mirax2.2-3 (Mirax, .mrxs)

(7) CMU-2 (Trestle, .tif)

(8) OS-2 (Ventana, .bif)

[9] See OpenSlides Homepage: `http://openslide.cs.cmu.edu`, or directly for the test data: `http://openslide.cs.cmu.edu/download/openslide-testdata/`

[10] see `https://github.com/SasNaw/ConversionService`

Because of their structure, (4), (6) and (7) will be placed in directories titled with their file extension. Fig. 3.1 shows the content of the input folder.

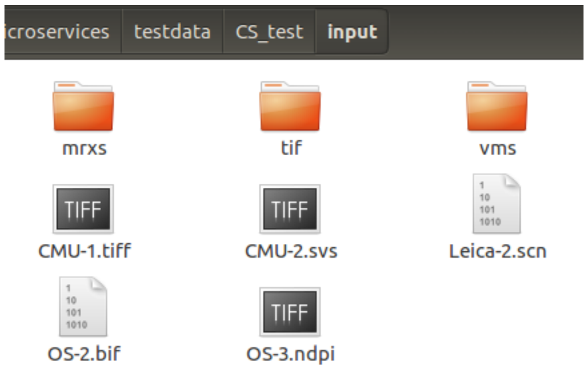

Figure 3.1: Content of input directory

This makes multiple calls of the CS necessary. The calls, in that order, are:

```
$ python ConversionService.py input/ out_1/
$ python ConversionService.py input/mrxs out_2/
$ python ConversionService.py input/tif out_3/
$ python ConversionService.py input/vms out_4/
```

3.3.2 Result

All runs of ConversionService.py were successful. Tab. 3.5 shows an overview of the results:

input	**output**	**time (sec)**
input/	CMU-1.dzi, CMU-2.dzi, Leica-2.dzi, OS-2.dzi, OS-3.dzi	1992
input/mrxs/	Mirax2.2-3.dzi	500
input/tif/	CMU-2.dzi	56
input/vms/	CMU-2-40x - 2010-01-12 13.38.58.dzi	305

Table 3.5: Results of Conversion Service Test

The vast difference in file size of the test data accounts for the different run times of the tests. While the first test converted 5 WSIs (399 sec/WSI), every other test converted a single one. The conversion of (6) was much faster, since the file was smaller in size (304.22 MB) compared to the others (1495.24 MB on average).

Chapter 4

Annotation Service

To enable the pathologist to make annotations in the first place, a GUI needs to be offered by the service. This GUI will be developed in iterations with the help of a number of pathologists to adapt it to their wishes and grant the best possible usability. The basic concept of the first iteration will be based on the Microdraw[1] GUI (see fig. 2.5).

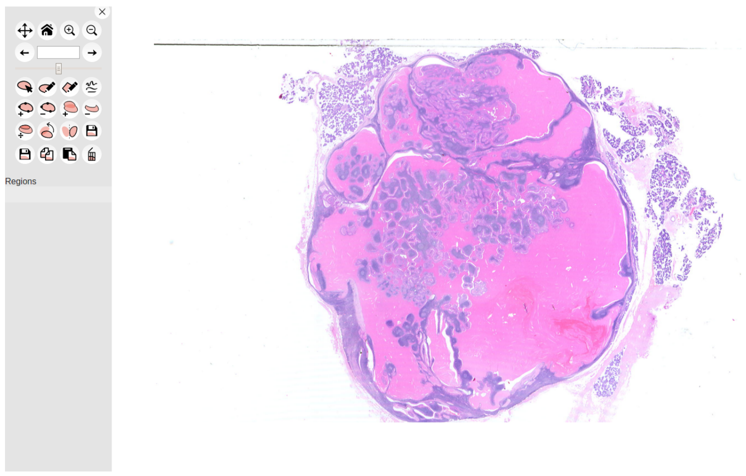

Figure 4.1: Microdraw GUI with opened WSI

Microdraw is a webbased annotation tool, which describes itself as

> ”[...] a collaborative vectorial annotation tool for ultra high resolution data, such as that produced by high-throughput histology.” [44]

[1]See `https://github.com/r03ert0/microdraw` for more information on the Microdraw project

Therefore, the GUI of the Annotation Service, or Annotation Service Viewer (ASV), will run as a web application in a browser. Annotations will be made by selecting a shape or annotation method from the various tools in the toolbar (see the gray bar on the left in fig. 2.5). After selecting the area to be annotated, an actual description of that area can be made via keyboard input.

When the pathologist is done or wants to save the made annotations, the Annotation Service will create a file in which they will be persisted in. Only one WSI can be opened in one ASV at a time.

4.1 Methodology

4.1.1 Parts of the Annotation Service

Annotation Service Viewer

Annotation Service Server

4.1.2 Functionality

4.2 Implementation

4.2.1 Technologies and Frameworks

4.2.2 Local Web Server

4.2.3 Web Application

Chapter 5

Tessellation Service

5.1 Methodology

5.2 Implementation

5.3 Test

5.3.1 Setup

5.3.2 Result

Chapter 6

Conclusion

6.1 Results

6.2 Conclusion

6.3 Future tasks

Appendices

Appendix A

Listing of Conversion Service Test Data

The test data for the Conversion Service can be found at OpenSlides homepage, at the freely distributable test data section[1]. Various slides can be found there. The following subsections (A.1 - A.8) give listings of all used WSIs, sorted by vendor and file format.

A.1 Aperio (.svs)

name	**size (MB)**	**description**
CMU-1-JP2K-33005.svs	126.42	Export of CMU-1.svs, brightfield, JPEG 2000, RGB
CMU-1-Small-Region.svs	1.85	Exported region from CMU-1.svs, brightfield, JPEG, small enough to have a single pyramid level
CMU-1.svs	169.33	Brightfield, JPEG
CMU-2.svs	372.65	Brightfield, JPEG
CMU-3.svs	242.06	Brightfield, JPEG
JP2K-33003-1.svs	60.89	Aorta tissue, brightfield, JPEG 2000, YCbCr
JP2K-33003-2.svs	275.85	Heart tissue, brightfield, JPEG 2000, YCbCr

Table A.1: Aperio data set (source: http://openslide.cs.cmu.edu/download/openslide-testdata/Aperio/)

[1]http://openslide.cs.cmu.edu/download/openslide-testdata/

A.2 Generic Tiled tiff (.tiff)

name	**size (MB)**	**description**
CMU-1.tiff	194.66	Conversion of CMU-1.svs to pyramidal tiled TIFF, brightfield

Table A.2: Generic Tiled tiff data set (source: http://openslide.cs.cmu.edu/download/openslide-testdata/Generic-TIFF/)

A.3 Hamamatsu (.ndpi)

name	**size (MB)**	**description**
CMU-1.ndpi	188.86	Small scan with valid JPEG headers, brightfield, circa 2009
CMU-2.ndpi	382.14	Brightfield, circa 2009
CMU-3.ndpi	270.1	Brightfield, circa 2009
OS-1.ndpi	1,860	H&E stain, brightfield, circa 2012
OS-2.ndpi	931.42	Ki-67 stain, brightfield, circa 2012
OS-3.ndpi	1,370	PTEN stain, brightfield, circa 2012

Table A.3: Hamamatsu data set (.ndpi, source: http://openslide.cs.cmu.edu/download/openslide-testdata/Hamamatsu/)

A.4 Hamamatsu (.vms)

name	**size (GB)**	**description**
CMU-1.zip	0.62	Brightfield
CMU-2.zip	1.13	Brightfield
CMU-3.zip	0.91	Brightfield

Table A.4: Hamamatsu data set (.vms, source: http://openslide.cs.cmu.edu/download/openslide-testdata/Hamamatsu-vms/)

A.5 Leica (.scn)

name	**size (GB)**	**description**
Leica-1.scn	0.28	Brightfield, single ROI, 2010/10/01 schema
Leica-2.scn	2.1	Mouse kidney, H&E stain, brightfield, multiple ROIs with identical resolutions, 2010/10/01 schema
Leica-3.scn	2.79	Mouse kidney, H&E stain, brightfield, multiple ROIs with different resolutions, 2010/10/01 schema
Leica-Fluorescence-1.scn	0.02	Fluorescence, 3 channels, single ROI, 2010/10/01 schema

Table A.5: Leica data set (source: http://openslide.cs.cmu.edu/download/openslide-testdata/Leica/)

A.6 Mirax (.mrxs)

name	size (GB)	description
CMU-1-Exported.zip	2.02	Export of CMU-1.mrxs with overlaps resolved, brightfield, JPEG, CURRENT_SLIDE_VERSION 2.3
CMU-1-Saved-1_16.zip	0.003	Quick save of CMU-1.mrxs at 1/16 resolution (multiple positions per image), brightfield, JPEG, CURRENT_SLIDE_VERSION 1.9
CMU-1-Saved-1_2.zip	0.14	Quick save of CMU-1.mrxs at 1/2 resolution (multiple images per position), brightfield, JPEG, CURRENT_SLIDE_VERSION 1.9
CMU-1.zip	0.54	Brightfield, JPEG, CURRENT_SLIDE_VERSION 1.9
CMU-2.zip	1.22	Brightfield, JPEG, CURRENT_SLIDE_VERSION 1.9
CMU-3.zip	0.65	Brightfield, JPEG, CURRENT_SLIDE_VERSION 1.9
Mirax2-Fluorescence-1.zip	0.06	Fluorescence, 3 channels, JPEG, CURRENT_SLIDE_VERSION 2
Mirax2-Fluorescence-2.zip	0.04	Fluorescence, 3 channels, JPEG, CURRENT_SLIDE_VERSION 2
Mirax2.2-1.zip	2.61	HPS stain, brightfield, JPEG, CURRENT_SLIDE_VERSION 2.2
Mirax2.2-2.zip	2.38	HPS stain, brightfield, JPEG, CURRENT_SLIDE_VERSION 2.2
Mirax2.2-3.zip	2.77	HPS stain, brightfield, JPEG, CURRENT_SLIDE_VERSION 2.2
Mirax2.2-4-BMP.zip	0.95	Brightfield, BMP, CURRENT_SLIDE_VERSION 2.2
Mirax2.2-4-PNG.zip	1.01	Brightfield, PNG, CURRENT_SLIDE_VERSION 2.2

Table A.6: Mirax data set (source: http://openslide.cs.cmu.edu/download/openslide-testdata/Mirax/)

A.7 Trestle (.tiff)

name	**size (MB)**	**description**
CMU-1.zip	158.87	Brightfield
CMU-2.zip	304.22	Brightfield
CMU-3.zip	223.11	Brightfield

Table A.7: Trestle data set (source: http://openslide.cs.cmu.edu/download/openslide-testdata/Trestle/)

A.8 Ventana (.bif)

name	**size (GB)**	**description**
OS-1.bif	3.61	H&E stain, brightfield
OS-2.bif	2.53	Ki-67 stain, brightfield

Table A.8: Trestle data set (source: http://openslide.cs.cmu.edu/download/openslide-testdata/Trestle/)

Bibliography

[1] L. Fei-Fei A. Karpathy. *Deep Visual-Semantic Alignments for Generating Image Descriptions.* Department of Computer Science, Standford University, 2015. `http://cs.stanford.edu/people/karpathy/cvpr2015.pdf`.

[2] R. Berta. deepzoom. `http://search.cpan.org/~drrho/Graphics-DZI-0.05/script/deepzoom`. Accessed: 22.08.2016.

[3] J. Bugwadia. Microservices: Five architectural constraints. `http://www.nirmata.com/2015/02/microservices-five-architectural-constraints/`, February 2015. Accessed: 12.08.2016.

[4] D. Siganos C. Stergiou. Neural networks. Technical report, Imperial College London, 1995. `https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Conclusion`.

[5] T. Cornish. An introduction to digital whole slide imaging and whole slide image analysis. `http://www.hopkinsmedicine.org/mcp/PHENOCORE/CoursePDFs/2013/13%2019%20Cornish%20Digital%20Path.pdf`, July 2013. Accessed: 12.04.2016.

[6] T. Cramer. The international image interoperability framework (iiif): Laying the foundation for common services, integrated resources and a marketplace of tools for scholars worldwide. `https://www.cni.org/topics/information-access-retrieval/international-image-interoperability-framework`, December 2011. Accessed: 18.08.2016.

[7] J. Cupitt. Vips. `http://www.vips.ecs.soton.ac.uk/index.php?title=VIPS`. Accessed: 25.05.2016.

[8] G. Seeman D. Bourg. *AI for Game Developers.* O'Reilly, 2004.

[9] M. Marcellin D. Taubmann. *JPEG 2000: Image compression fundamentals, standards and practice.* Kluwer Academic Publishers, 2001.

[10] Working Group 26. Pathology DICOM Standards Committee. Digital imaging and communications in medicine (dicom), supplement 145: Whole slide microscopic image iod and sop classes, August 2010.

[11] digitalpreservation. Bigtiff. http://www.digitalpreservation.gov/formats/fdd/fdd000328.shtml. Accessed: 20.08.2016.

[12] digitalpreservation. Tiff. http://www.digitalpreservation.gov/formats/fdd/fdd000237.shtml. Accessed: 20.08.2016.

[13] D. Doubrovkine. Dzt. https://github.com/dblock/dzt. Accessed: 22.08.2016.

[14] S. Eddins. Tiff, bigtiff, and blockproc. http://blogs.mathworks.com/steve/2013/08/07/tiff-bigtiff-and-blockproc/, August 2013.

[15] A. Goode et al. Openslide: A vendor-neutral software foundation for digital pathology. *J pathol inform*, 4(1), September 2013. http://download.openslide.org/docs/JPatholInform_2013_4_1_27_119005.pdf.

[16] M. Appleby et al. Iiif image api 2.0. http://iiif.io/api/image/2.0/. Accessed: 18.08.2016.

[17] M. Egmonst-Petersen et al. Image processing with neural networks - a review. *Pattern Regcognition*, 35, October 2002. https://www.researchgate.net/publication/220603536_Image_processing_with_neural_networks_-_a_review_Pattern_Recogn_352279C2301.

[18] N. Farahanil et al. Whole slide imaging in pathology: advantages, limitations, and emerging perspectives. *Pathology and Laboratory Medicine International*, 7, June 2015. https://www.dovepress.com/whole-slide-imaging-in-pathology-advantages-limitations-and-emerging-p-peer-reviewed-fulltext-article-PLMI#ref10.

[19] R. Singh et al. Standardization in digital pathology: Supplement 145 of the dicom standards. *J pathol inform*, 2(23), March 2011. http://www.jpathinformatics.org/temp/JPatholInform2123-3144928_084409.pdf.

[20] S. Park et al. Digital imaging in pathology. *Clin Lab Med*, 4(32), December 2012.

[21] National Institute for Standards and Technology. Pyramidio. https://github.com/usnistgov/pyramidio. Accessed: 22.08.2016.

[22] Open Source Geospatial Foundation. Tile map service specification. http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification. Accessed: 26.04.2016.

[23] OpenStreetMap Foundation. Openstreetmap homepage. http://www.openstreetmap.org/about. Accessed: 18.08.2016.

[24] L. Fuller. sharp homepage. http://sharp.dimens.io/en/stable/. Accessed: 22.08.2016.

[25] D. Gasienica. Vips. `https://github.com/zoomhub/zoomhub`. Accessed: 22.08.2016.

[26] I. Gilman. Openseadragon. `http://openseadragon.github.io/examples/creating-zooming-images/`. Accessed: 26.04.2016.

[27] IIIF. Internation image interoperability framework homepage. `http://iiif.io`. Accessed: 18.08.2016.

[28] ImageMagick. Imagemagick: Formats. `http://www.imagemagick.org/script/formats.php`. Accessed: 22.08.2016.

[29] Optimus Information Inc. Open-source vs. proprietary software, pros and cons. `http://www.optimusinfo.com/downloads/white-paper/open-source-vs-proprietary-software-pros-and-cons.pdf`, 2015. Accessed: 16.08.2016.

[30] intoPix. Everything you always wanted to know about jpeg 2000. `http://www.intopix.com/pdf/JPEG%202000%20Handbook.pdf`, 2008.

[31] K. Martinez J. Cupitt. Vips: an image processing system for large images. *Proc. SPIE*, 2663:19 – 28, 1996. `http://eprints.soton.ac.uk/252227/1/vipsspie96a.pdf`.

[32] K. Martinez J. Cupitt. Vips - a highly tuned image processing software architecture. In *IEEE International Conference on Image Processing*, pages 574 – 577, September 2005. `http://eprints.soton.ac.uk/262371/`.

[33] M. Fowler J.Lewis. Microservices, a definition of this new architectural term. `http://martinfowler.com/articles/microservices.html#footnote-etymology`, March 2014. Accessed: 12.08.2016.

[34] JPEG. Overview of jpeg 2000. `https://jpeg.org/jpeg2000/`. Accessed: 18.08.2016.

[35] D. Kriesel. *A Brief Introduction to Neural Networks*, 2007. `http://www.dkriesel.com/en/science/neural_networks`.

[36] R. Joshi M. Rabbani. An overview of the jpeg2000 still image compression standard. *Signal Processing Image Communication*, 17(3), 2002. `https://www.csd.uoc.gr/~hy471/bibliography/jpeg2000.pdf`.

[37] Microsoft. Deep zoom file format overview. `https://msdn.microsoft.com/en-us/library/cc645077(v=vs.95).aspx`. Accessed: 17.08.2016.

[38] K. Mulka. Gmap uploader tiler. `https://github.com/mulka/tiler`. Accessed: 22.08.2016.

[39] OpenSlide. Openslide format documentation. `http://openslide.org/`. Accessed: 16.08.2016.

[40] J. Riggins. Microservices architecture: The good, the bad, and what you could be doing better. http://nordicapis.com/microservices-architecture-the-good-the-bad-and-what-you-could-be-doing-better/, April 2015. Accessed: 12.08.2016.

[41] R. Rojas. *Neural Networks - A Systematic Introduction.* Springer-Verlag, 1996. https://page.mi.fu-berlin.de/rojas/neural/.

[42] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 1958. http://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf.

[43] D. Shiffman. *The Nature of Code.* D. Shiffman, 2012. http://natureofcode.com/book/chapter-10-neural-networks/.

[44] R. Toro. Microdraw. https://github.com/r03ert0/microdraw/wiki. Accessed: 18.04.2016.

[45] E. Wolff. *Microservices Primer.* innoQ, January 2016. https://leanpub.com/microservices-primer/read.

List of Figures

List of Tables

Nomenclature

AS	Annotation Service
ASV	Annotation Service Viewer
BNN	Biological Neural Networks
CS	Conversion Service
DICOM	Digital Imaging and Communications in Medicine
DZI	Deep Zoom Image
DZI	Deep Zoom Images
GUI	Graphical User Interface
GUI	Graphical User Interface
MS	Microservice
NN	Neural Network
NN	Neural Networks
TS	Tessellation Service
VIPS	VASARI Image Processing System
WSI	Whole Slide Image