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)

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 12, 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) [3]. 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" [13]. 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 2.1.3

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. 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 [3], those are:

- JPEG2000

- TIFF

- Deep Zoom Images (DZI)

- DICOM (supplement 145), without reference implementation as of yet [3]

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.

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 the steps (a) - (c) of the process chain from section 1.2. 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 Image Formats

2.1.1 Open WSI Formats

Deep Zoom Images

2.2 Short Introduction to Neural Networks

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 [12] 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 [17]. 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 [17]. 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 [12].

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 [15].

Even though MS are more a concept than a specific architectural style, certain traits are usually shared between them [15]. According to [15] and [12], 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, [2] 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. [2] 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 [17].

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 [15].

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[1], interprocess communication becomes a necessity. This can lead to communicational overhead which has a negative impact on the overall performance of the system [17].

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 [15].

---

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

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:

- Interprocess communication doesn't arise between the single stages of the process chain, since they have a set order[2]

- 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 (2.2.1 - 2.2.3) are dedicated to illustrate the process chain necessary to accomplish the research objective stated in chapter 1.2. The process chain consists of the following steps:

---

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

(a) convert WSIs of different[3] formats to dzi format

(b) annotate dzi images with the annotation tool made for this purpose

(c) persist made annotations in a file

(d) seperate annotated images into tiles of custom size

(e) keep correspondence between tiles of an image and its annotations

To fulfill those steps, 3 Microservices will be introduced in the following subsections. Those are:

- Conversion Service (see chap. 2.2.1)
  This service will be responsible for converting WSIs into the dzi format (a).

- Annotation Service (see chap. 2.2.2)
  This service will offer a tool to annotate an image (b) and persist made annotations (c).

- Tessellation Service (see chap. 2.2.3)
  This service will be responsible for seperating an image into tiles (d) and keep the correspondence between tiles and annotations (e).

2.4.1 Conversion Service

The devices which create an WSI, so called whole slide scanners, create images in various formats, depending on the producer. The conversion service has the goal of converting those formats into the dzi format, not only for the purpose of unification, but also to add the deep zoom feature[4] to the images (see fig. 2.3). This is of special importance, since an average WSI with 1,600 megapixels has a size of approximately 4.6 GB [6].

set of WHIs of different formats   convert   set of WHIs in .dzi format
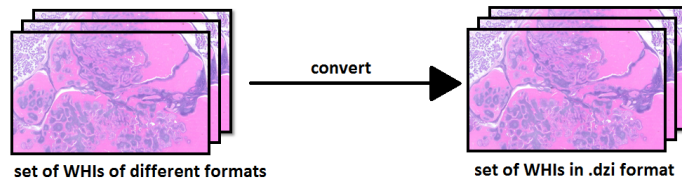
Figure 2.1: Visualization of the Conversion Service

The Conversion Service will be implemented as a python script, which upon calling takes every single WSI of a certain format inside a given folder and

---

[3]See chap. 2.2.1 for a listing of valid input formats.
[4]See chap. 2.1.1

converts it to the dzi format. Each converted WSI will be saved in another specified folder. Valid image formats (and their corresponding producers) for conversion are:

- .bif (Ventana)

- .mrxs (Mirax)

- .ndpi (Hamamatsu)

- .scn (Leica)

- .svs (Aperio)

- .svslide (Sakura)

- .tif (Aperio, Trestle, Ventana)

- .tiff (Philips)

- .vms (Hamamatsu)

- .vmu (Hamamatsu)

2.4.2 Annotation Service

The first step to create a valid training sample for a NN is to annotate the WSIs which will later serve as that. To do so, a GUI must be deployed which enables a pathologist to make annotations to an WSI. Additionally, the Annotation Service also needs to be capable to persist made annotations (see fig. 2.4). This will happen by saving the annotations into a file which will be placed next to the image.
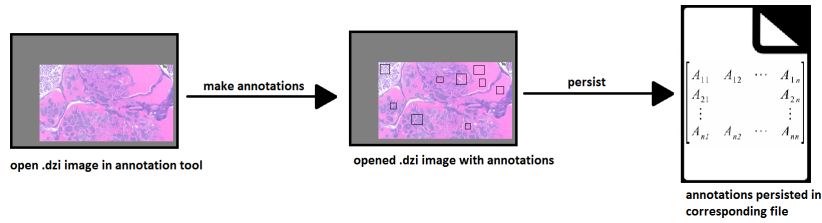

Figure 2.2: Visualization of the Annotation Service

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[5] GUI (see fig. 2.5).
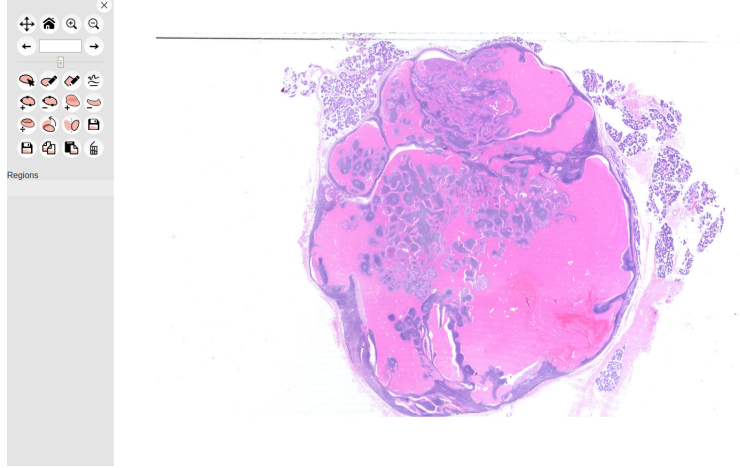

Figure 2.3: Microdraw GUI with opened WSI

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." [16]

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.

2.4.3 Tessellation Service

The task of the Tessellation Service is to tessellate a given WSI into multiple tiles while keeping up the correspondence between image areas and annotations (see fig. 2.6). Tessellation describes the process of seperating a geometric space (e.g. a plane) into multiple tiles of one or more shapes. No matter if the tiles are unifrom or of different shapes, there must be no gaps or overlapping areas in the resulting tiles.

---

[5]See `https://github.com/r03ert0/microdraw` for more information on the Microdraw project
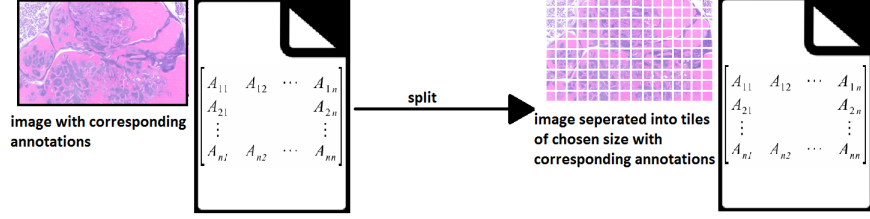
Figure 2.4: Visualization of the Tessellation Service

The tessellation of the Tessellation Service differs from the given definition in two aspects. First, all tiles will be of a unifrom shape (square). Second, the service will also offer the possibility of seperating only the annotated areas of an WSI into tiles. Therefore, the "no gaps" rule is invalid, when this option is chosen. The rule of no overlapping areas holds true in either case, however.

As mentioned before, the second task of the Tessellation Service is to ensure the correspondence between tiles and annotations of the original WSI. This means that, if the original WSI has an annotation in the area of a tile, this tile needs to have the same annotation.

2.5   Definition of terms

To prevent missunderstandings and confusion, the following subsections 2.1.1 - 2.1.3 will define some terminoligy which will be mandatory for the understanding of certain areas of this thesis.
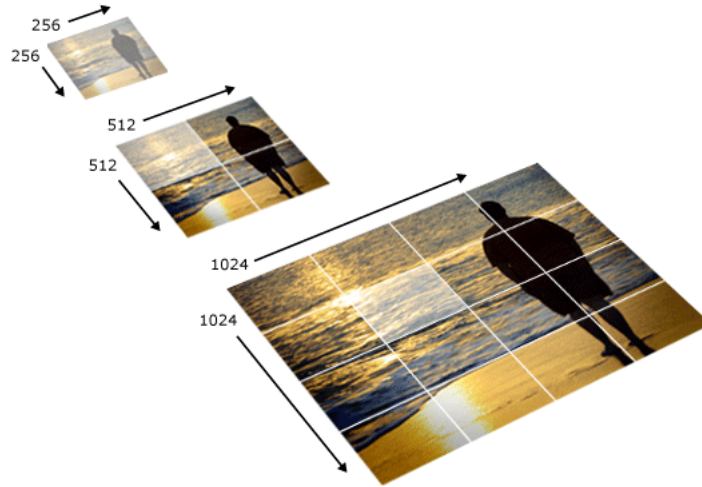
2.5.1 Deep Zoom Image Format

Figure 2.5: 3 consecutive levels of a dzi image (source: https://i-msdn.sec.s-msft.com/dynimg/IC141135.png)

The Deep Zoom Image Format (dzi) is an xml-based file format maintained by Microsoft to improve performance and quality in the handling of large image files. For this purpose an image is represented in a tiled pyramid (see fig. 2.1).

As seen in fig. 2.1 there are multiple versions of a single image in different resolutions. Each resolution in the pyramid is called a *level*. At each level the image is scaled down by the factor 4 (2 in each dimension). Furthermore, the image gets tiled up into 256^2 tiles (256 in each dimension) [14].

If a viewer wants to view a certain area of the image (e.g. the highlighted tile in the last image in fig. 2.1), only the corresponding tiles need to be loaded. This saves large amounts of bandwidth and memory. The same goes for a viewer, who is zoomed out very far. In such a view the full level of detail isn't needed, so that a version from a lower level can be loaded.

A dzi file consists of two parts: a describing .xml file[6] and a folder with more subfolder. Each subfolder describes a level and as such contains all the tiles for that particular level.

2.5.2 Neural Network

Artificial neural networks (NN) are a group of models inspired by biological neural networks[7]. In a NN, regardless if artificial or biological, many neurons

---

[6]Frameworks like *OpenSeaDragon* also support further formats, such as .json.

[7]For the remainder of this thesis, neural network will always represent the artificial one, unless explicitly stated otherwise.

are interconnected with each other. The construct of interconnected neurons can be seperated into layers, of which there are three kinds:

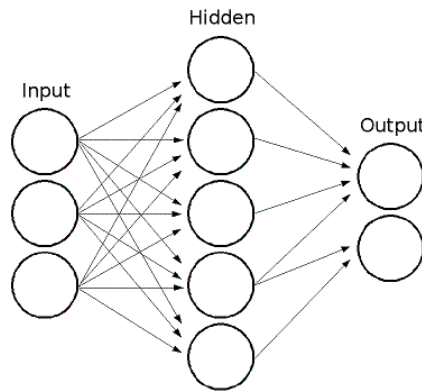- input layer

- hidden layer

- output layer

Figure 2.6: 3 layer NN (source: http://docs.opencv.org/2.4/_images/mlp.png)

Basically, the input layer, as the name suggests, is the layer where the NN gets its input data from. After that, there are a number of hidden layers[8], which are responsible for further computation of the input values. At the end is the output layer which is responsible for communicating the results of the prior operations (compare fig. 2.2). Each single neuron 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.

A huge benefit of NN, over other software models, is their ability to learn. While certain problems are easier to solve in a sequential, algorithmic fashion (say an equation or the towers of hanoi), certain problems are so complex 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 train to solve a problem, not unlike a human, by learning. Since this topic alone is enough for a number of theses, the author refers to [13] for further detailed information.

---

[8]A NN doesn't necessarily need to have any hidden layers. For non trivial problems however, it becomes mandatory.

Chapter 3

Conversion Service

3.1 Methodology

The objective of the Conversion Service is to convert a given set of input WHIs into dzi files. Since a dzi is layered into a pyramid scheme, it is necessary to calculate the needed number of levels, as well as the dimensions of each level (see fig. 3.1 for an example).
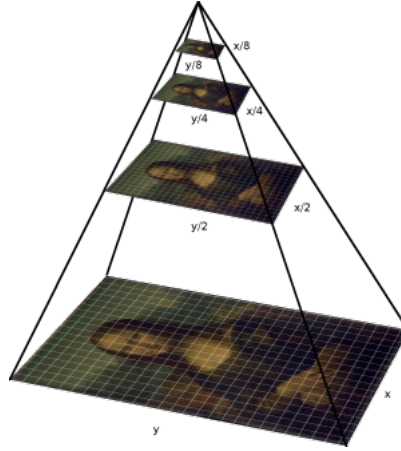

Figure 3.1: Example of a pyramid scheme in image processing (source: `http://iipimage.sourceforge.net/images/pyramid.png`)

Therefore, the Conversion Service must be able to open an WHI img_{input} of any of the in 2.2.1 defined formats. Based on the size of img_{input} the number of necessary levels lvl must be calculated. Once lvl has been determined, img_{input} must be resized into an appropriate scale for each lvl_i in lvl. The resized image will be called img_i, with i representing the corresponding level. In the next

step, every img_i will be tessellated into $x * y$ tiles. Each tile will be referrenced via $t_{c,r}^i$, with r being the row and c being the column of the tile in whi_i. To complete the conversion, the Conversion Service must create a describing xml file for each converted image img_{dzi}.

3.1.1 Creating a Deep Zoom Image

To create a dzi, the Conversion Service must be capable of all the afore mentioned tasks. According to [9], there is a number of frameworks, that can achieve this task (see tab. 3.1).

Since the conversion to dzi is required, two frameworks are not usable from the start. Those are MapTiler, who creates tms[1] images and Kakadu, which creates iiif[2] images. Furthermore, the various desktop applications are not usable either, since the Conversion Service should operate as a microservice.

For a python script, deepzoom.py would be the natural choice, however at a second glance it can be seen that VIPS offers "[...] for a number of languages" (see tab. 3.1). One of those languages happens to be python. Therefore, deepzoom.py and VIPS will be further investigated in the following two subsections.

---

[1]Tile Map Service (tms) is a tile scheme developed and maintained by the Open Source Geospatial Foundation [8]

[2]International Image Interoperability Framework (iiif), specified by the International Image Interoperability Framework group, is an image delivery API which responds to requests via HTTP and HTTPS [5]

option	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 (via dzsave feature)
Sharp	Node.js, uses VIPS	dzi
MagickSlicer	shell script (Linux/Max)	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 3.1: Overview of conversion options for zooming image formats (source: [9])

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:

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

The various options and their default values can be seen in tab. 3.2. If called without a designated output destination, deepzoom.py will save the converted dzi right next to the original 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 3.2: 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:

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

---

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

The options are analogous with the terminal version (compare tab. 3.2). To start the conversion process, the following call must be made within the python script:

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

Upon calling, the ImageCreator opens the input image img_{input} and creates a description with all the needed information for the dzis describing xml file[5]. After that, the number of levels is calculated. For this, the bigger value of height and width of img_{input} is chosen (see eq. 3.1) and then used to determine the number of levels lvl (see eq. 3.2).

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

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

Once lvl has been determined, img_{input} will be resized in the chosen quality (-q/image_quality) for every level i, with $i \in (0, lvl - 1)$. The new resolution will be calculated for both dimensions dim with a function $scale$ (see eq. 3.3) analogously. Furthermore, the image will be interpolated with the specified filter (-r/resize_filter). The resized image will be called img_i.

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

Once img_i has been created, it will be tessellated into as many tiles of the specified size (-s/tile_size) and with the specified overlap (-o/tile_overlap) as possible. Since not every image will be of the size $2^n, n \in$ in either dimension, it is highly likely that the set of tiles for the last column/row will be smaller then specified in either dimension.

Every tile will be saved as [column]_[row].[format] ([format] depending on -f/file_format) in a folder called according to the corresponding level i. This folder will be located inside another folder, called [filename]_files. The describing xml file will be persisted as [filename].dzi on the same level.

3.1.3 VIPS

VIPS (VASARI Image Processing System) describes itself as "[...] a free image processing system [...]" [4]. 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 for the Conversion Service[6] [4].

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

---

[5]Compare chap. 2.1.1
[6]See chap. 2.2.1

line. The GUI will not be further discussed, since it is of no interest for the implementation of the Conversion Service.

One of the strongest traits of VIPS is its speed and little data usage compared to other imaging libraries [11]. Given the task of the Conversion Service to convert WSIs of various formats, which tend to be quite large, both of those perks are especially welcome.

VIPS superior 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 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 [10].

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 will look like the following:

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

When called, VIPS will take the image [input], convert it into the dzi file format and then save it to [output]. The various options and their default values can be seen in tab. 3.3.

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 3.3: 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 into the specified output location.

3.2 Implementation

As stated before, the Conversion Service is implemented as a python script. The first iteration of the script used deepzoom.py for the conversion. This caused severe performance issues though. Out of all the image files in the test set[7], only one could be converted[8]. Every other file was either too big, so the process would eventually be killed by the operating system or exit with an IOError concerning the input file from the PIL imaging library. The second iteration of the Conversion Service uses VIPS, which is faster then deepzoom.py and also capable of converting all the given test images.

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

```
1    $ 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 up to conversion and where to save the resulting dzi files.

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

```
1  def main():
2      path = checkParams()
3      files = os.listdir(path)
4      for file in files:
5          print("——————————————————————————————————————")
6          extLen = getFileExt(file)
7          if(extLen != 0):
8              print("converting " + file + "...")
9              convert(path, file, extLen)
10             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
2  # param path: directory of param file
3  # param file: file to be converted
4  # param extLen: length of file extension
5  def convert(path, file, extLen):
6      dzi = OUTPUT + file[:extLen] + "dzi"
7      im = Vips.Image.new_from_file(path + file)
8      im.dzsave(dzi, overlap=OVERLAP, tile_size=TILESIZE)
```

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 6). *OUTPUT*

---

[7]See chap. 3.3
[8]CMU-3.svs from Aperio, see Appendix A

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*]/".

3.3   Test

To test the correct functionality of the Conversion Service, test data is needed. OpenSlide offers a selection of freely distributable data[9], which can be used for that purpose.

Since the WSIs are of enormous size, they are not delivered via the repository. 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)

Because of their structure, (4), (6) and (7) will be placed in directories titled with their file extension. Fig. 3.2 shows the content of the input folder.

---

[9]See OpenSlides Homepage: `http://openslide.cs.cmu.edu`, or directly for the test data: `http://openslide.cs.cmu.edu/download/openslide-testdata/`
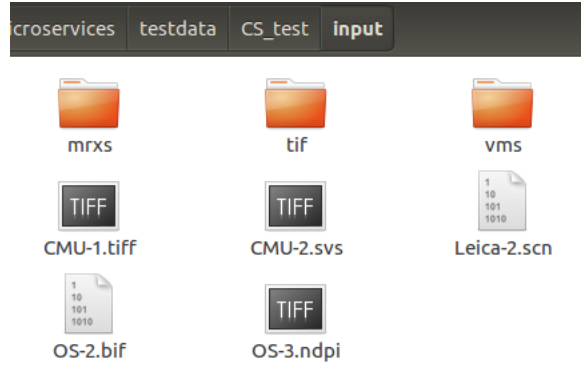
Figure 3.2: Content of input directory

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

```
1   $ python ConversionService.py input/ out_1/
2   $ python ConversionService.py input/mrxs out_2/
3   $ python ConversionService.py input/tif out_3/
4   $ python ConversionService.py input/vms out_4/
```

3.3.2   Result

All runs of ConversionService.py were successful. Tab. 3.3 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 3.4: 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).

25

Chapter 4

Annotation Service

4.1   Methodology

4.1.1   DICOM

4.2   Implementation

4.3   Test

4.3.1   Setup

4.3.2   Result

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

[3] 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.

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

[5] M. Appleby et al. Iiif image api 2.0. `http://iiif.io/api/image/2.0/#status-of-this-document`. Accessed: 26.04.2016.

[6] 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`.

[7] R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of Carolina, 2000. `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`.

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

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

[10] 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`.

[11] 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/`.

[12] 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.

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

[14] Microsoft. Deep zoom file format overview. `https://msdn.microsoft.com/en-us/library/cc645077(v=vs.95).aspx`. Accessed: 11.04.2016.

[15] 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.

[16] Roberto Toro. Microdraw. `https://github.com/r03ert0/microdraw/wiki`. Accessed: 18.04.2016.

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

List of Figures

List of Tables