Preparation of Whole Slide Images for usage in Neural Networks

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, April 13, 2016

Abstract

This is the abstract.

Preface

Hello, this is the preface

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 microscopes 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* [2]. The digital nature of whole slide images 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 area of image analysis are the so called *neural networks*[1], which follow a machine learning approach. Their use, especially in the area of image classification and object recognition, made big breakthroughs possible in the recent past, e.g. Karpathy and Fei-Fei [1], who created a neural network which is capable of describing an image or a scene with written text (see fig. 1.1 for some examples).

There is enormous potential in the use of neural networks in the digital pathology as well, but to transfer these algorithms and technologies, certain hindrances must be overcome. One of those hindrances is the need for proper training samples and an established ground truth[2]. While generally there are huge amounts of whole slide images, 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 whole slide images, stored and later used for training.

Therefore the goal of this thesis is to give tools into the hands of pathologists to annotate whole slide images and save those annotations in such a way that they will be usable later in the combination with neural networks.

[1] See chapter 2.1.3

[2] *Ground truth* refers to a training sample whose outcome is known and validated as true.

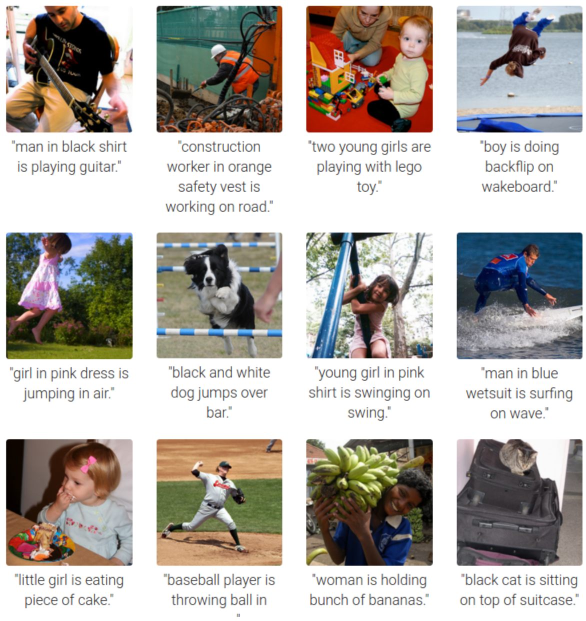

Figure 1.1: Example results of the in [1] introduced model (source: http://cs.stanford.edu/people/karpathy/deepimagesent/

1.2 Research Objective

The objective of this thesis is the conceptualization and implementation of tools for whole slide images, which allow for their annotation and a further usage in neural networks. As a requirement, the tools have to be implemented in the form of microservices[3], each one with their own short documentation, including instructions for installation, usage and some examplary use cases. To achieve this goal, the implementation of 3 microservices is necessary.

The first microservice needs to be capable of converting a given set of image formats into the so called *Deep Zoom Image Format*[4]. The supported image formats are *.bif, .mrxs, .ndpi, .scn, .svs, .svslide, .tif, .tiff, .vms* and *.vmu*, in accordance with the capabilities of the Openslide framework [3].

The second microservices task is to give a tool at hand with which a pathologist will be able to annotate all those whole slide images which were converted using the first microservice. Furthermore, the made annotations need to be persisted together with the highest resolution of the corresponding image.

The third microservice will be responsible for preparing the annotated whole slide images for the further usage in neural networks. For that purpose the ser-

[3]See chapter 2.1.2
[4]See chapter 2.1.1

vice needs to be capable of dividing a single annotated whole slide image into multiple tiles, with the choice of either using the whole image or just the annotated areas. Furthermore, each tile needs enough information to reconstruct the whole image again afterwards.

1.3 About this thesis

Apart from the *Introduction*, there are 5 more chapters in this thesis.

Chapter 2 - Background defines some terminoligy and the general, required process chain which are all necessary to understand further chapters of this thesis. Furthermore, 3 microservices will be introduced in short.

Chapter 3 - Methodology gives an overview over the current state of research for each microservice, as well as best practices.

Chapter 4 - Implementation goes into further details about how each microservice is implemented and which software and frameworks were used for that.

Chapter 5 - Discussion will introduce a measurement for each microservice to measure its success. It will discuss the test setup as well as list the results.

Chapter 6 - Conclusion will interpret the Results from Chapter 5 and analyze them closer. Furthermore, it will give an idea of what steps are to be taken next in the future.

Chapter 2

Background

2.1 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.

2.1.1 Deep Zoom Image Format

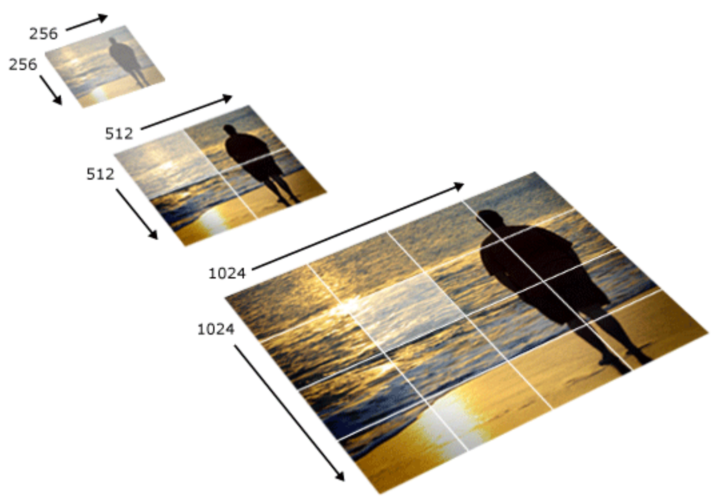

Figure 2.1: 3 consecutive levels of a .dzi image (source: https://i-msdn.sec.s-msft.com/dynimg/IC141135.png)

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) [6].

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[1] and a folder with more subfolder. Each subfolder describes a level and as such contains all the tiles for that particular level.

2.1.2 Microservice

The concept of microservices is to seperate one monolithic software construct into several smaller, modular pieces of software. According to [7], the idea of microservices is not new, but can be found in the UNIX philosophy. Three basic ideas are stated in [7]:

- A program should fulfill only one task, and it should do it well.
- Programs should be able to work together.
- Besides, the programs should use a universal interface.

As such, microservices are a modularization concept. However, they differ from other concepts, since they are independet from each other. This is a trait, other modularization concepts usually lack. As a result, changes in one microservice don't bring up the necessity of deploying the whole product again but just the one service.

Because of their inherent traits, microservices need to be their own processes in one way or another, may it be as an actual operating system process or as e.g. a docker container[2].

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, like data bases. It is imperative for the concept of modularization, however, that each microservice has its own storage of which it is in charge of.

A disadvantage of this modularization is, that inter process communication becomes a necessity. However, there are different approaches with which microservices can communicate. [7] suggests the following:

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

[2] Docker is a tool, which enables software to be wrapped up in so called "'containers"'. Those containers are a complete, but stripped down, filesystem containing everything the software needs to run (e.g. source code, runtime environment, system tools and libraries, ...). See `https://www.docker.com/what-docker`

- communication via protocols like REST[3]
- an HTML user interfaces with links to other microservices
- data replication

It is important to define how and with which technology to communicate with, when adressing each microservices to ensure that this particular one can actually be reached with the defined method.

2.1.3 Neural Network

Artificial neural networks (NN) are a group of models inspiried by biological neural networks[4]. In a NN, regardless if artificial or biological, many neurons 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

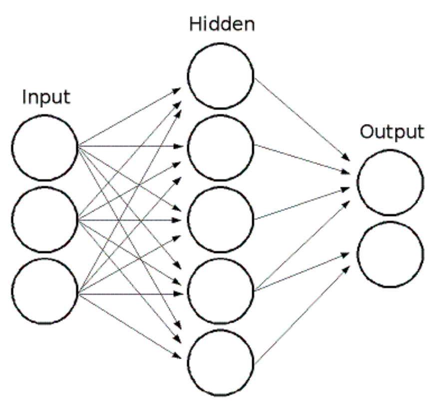

Figure 2.2: 3 layer NN (source: http://docs.opencv.org/2.4/_images/mlp.png)

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[5],

[3]Representational state transfer (REST) is architectural style for distributed hypermedia systems, see [4].

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

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

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 [5] for further detailed information.

2.2 Process Chain

This section and its following subsections (2.2.1 - 2.2.3) are dedicated to illustrating the defined process chain necessary to accomplish the research objective stated in chapter 1.2.

The process chain has the following basic steps in the given order:

- convert WHIs to .dzi format
- annotate converted images with annotation tool
- persist annotations
- seperate annotated image into tiles of custom size and keep correspondence between tile and annotation

To fulfill those steps, 3 Microservices will be introduced. Those are:

- Conversion Service (see chap. 2.2.1)
- Annotation Service (see chap. 2.2.2)
- Tesselation Service (see chap. 2.2.3)

The conversion service will be responsible for converting WHIs into the .dzi format. The anntation service will offer a user interface to make annotations and be responsible for persisting them. Last, the tesselation service will be responsible for creating tiles with the annotated informations from the corresponding picture. Furthermore, this service must also be able to restructure the tiles into the initial image again.

In the following, each mircoservice will have a section to define it more closey.

2.2.1 Definition of Conversion Service

2.2.2 Definition of Annotation Service

2.2.3 Definition of Tesselation Service

Chapter 3

Methodology

3.1 Conversion Service

3.1.1 Literature review

3.1.2 Chosen methods

3.2 Annotation Service

3.2.1 Literature review

3.2.2 Chosen methods

3.3 Tessellation Service

3.3.1 Literature review

3.3.2 Chosen methods

Chapter 4

Implementation

4.1 Implementation of Conversion Service

4.1.1 Used technologies and frameworks

4.1.2 Documentation

4.2 Implementation of Annotation Service

4.2.1 Used technologies and frameworks

4.2.2 Documentation

4.3 Implementation of Tessellation Service

4.3.1 Used technologies and frameworks

4.3.2 Documentation

Chapter 5

Discussion

5.1 Conversion Service Test

5.1.1 Setup

5.1.2 Results

5.2 Annotation Service Test

5.2.1 Setup

5.2.2 Results

5.3 Tessellation Service Test

5.3.1 Setup

5.3.2 Results

Chapter 6

Conclusion

6.1 Results

6.2 Conclusion

6.3 Future tasks

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] 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.

[3] 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.

[4] 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.

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

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

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

List of Figures

List of Tables