

Masaryk University
Faculty of Informatics



Visualization of Digital Pathology Images and Results of Their Analyses Using Deep Neural Networks

Master's Thesis
Nikoleta Češková

Brno, Spring 2020

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Nikoleta Češková

Advisor: doc. RNDr. Petr Holub, Ph.D.

Acknowledgements

I would like to thank my supervisor doc. RNDr. Petr Holub, Ph.D. for his support and guidance during the running of this project. Furthermore, I would like to thank MUDr. Rudolf Nenutil, CSc. from Masaryk Memorial Cancer Institute for his feedback on design and implementation of the system, and finally, I would also like to thank the rest of the research team for their collaborative effort.

Abstract

The aim of this thesis is to compare high-resolution image viewers suitable for visualization of large-scale pathology images and to modify one of these tools for the specific needs of the digital pathology research project between Masaryk University and Masaryk Memorial Cancer Institute.

Keywords

Digital pathology, WSI, visualization, tiled image pyramid, high-resolution viewers, OpenSeadragon

Table of Contents

List of tables	vi
List of figures	vii
Introduction	1
1. Digital Pathology	3
1.1 AI in digital pathology	4
1.2 WSI	5
1.2.1 Optimization	6
1.2.2 File formats	8
2. Project description	10
2.1 Workflow	10
2.2 Viewer requirements	12
3. Viewers	15
3.1 WSI viewers	15
3.1.1 Cytomine	15
3.1.2 HistomicsTK	17
3.2 Image viewers	17
3.2.1 IIMPooViewer	18
3.2.2 OpenSeadragon	18
4. Methodology	19
5. Analysis	22
5.1 Analysis conclusion	33
6. Design and implementation	34
6.1 Design	34
6.2 Implementation	37
Conclusion	44
References	45
Appendix A – Installation instructions	46
Appendix B – User manual	48

List of tables

Table 1 Overlay summary.....	23
Table 2 Annotation summary.....	24
Table 3 Performance summary	26
Table 4 Interactiveness summary.....	28
Table 5 Modifiability summary.....	32

List of figures

Figure 1 Pyramid tile structure	7
Figure 2 DZI description file.....	9
Figure 3 Grayscale probability mask	11
Figure 4 Simplified process overview.....	12
Figure 5 Cytomine workflow	16
Figure 6 IIPMooViewer mid-zooming	27
Figure 7 Stackoverflow search results	30
Figure 8 Contributors on GitHub	30
Figure 9 Issues on GitHub.....	31
Figure 10 Issues on GitHub (February 2020)	32
Figure 13 Tiles in OpenSeadragon	35
Figure 14 getTileUrl function	35
Figure 15 Opacity control for the annotation layer	37
Figure 16 Thresholding computation for probabilities mask	38
Figure 17 Annotation input form	40
Figure 18 Web page with results	41
Figure 19 Implementation of Deepzoom protocol	42
Figure 20 Final design of OpenSeadragon.....	43

Introduction

Artificial intelligence (AI) is nowadays being incorporated into several different fields, such as finance, education, robotics, military, transportation, or healthcare. Apart from large corporations investing in the development of these technologies, many smaller independent research groups around the world attempt to employ the AI to simplify laborious tasks and improve the quality of life. The critical element of these projects is an efficient method to present the results of AI algorithms to the final users who are supposed to benefit from them.

The goal of this thesis is to analyze the state of the art of visualization of AI systems in the domain of digital pathology, to enable users to visualize results and to interact with them. The aims of the thesis are twofold: (a) design methodology to compare existing tools for visualization of large-scale digital pathology imaging and to perform the actual comparison, (b) to develop a system for a visualization based on the most suitable existing tool specifically for the digital pathology project between Masaryk University and Masaryk Memorial Cancer Institute.

The thesis is divided into the following parts:

The first chapter introduces the digital pathology. After a brief introduction to this domain, we will focus on the application of artificial intelligence and its current shortcomings. The second part will explore techniques of digital image acquisition and its subsequent optimization, which form the necessary background to understand this thesis.

The second chapter describes the project which this thesis is part of, and which uses deep neural networks to identify cancerous regions in the high-resolution scans of optical microscopy slides tissue. Apart from the project workflow, requirements regarding an image viewer will be specified.

The third chapter provides a brief overview of the functionalities of visualization tools selected for further analysis.

The fourth chapter defines the methodology on how to compare selected viewers, and the fifth chapter details the results of the analysis.

The last part of the thesis focuses on my own implementation of the visualization system and how it was developed based on modifications of the selected viewer.

My contribution to the thesis includes:

- Collection and summary of project requirements concerning the viewer.
- Research of existing solutions.
- Analysis of selected solutions.
- Implementation of a custom viewer specifically for the needs of the project and implementation of missing functionalities, based on the outcomes of the analysis of the existing solutions.

1. Digital Pathology

One of the main challenges for today's pathologist is managing vast amounts of data. Even though glass slides are an efficient way for the initial diagnosis, they are expensive and time-consuming when it comes to consultation, education, research, or physical management. Possibly more efficient and cost-effective means of presenting pathology information is provided by digital pathology, one of the newest fields in medicine, which only started to be mentioned in published scientific journals in the year 2000. [1]

In the beginning, digital pathology was perceived merely as the conversion of the optical image of a pathologic slide into a digital image that can be viewed via a computer. However, with an advance in technology, the concept and functionality of digital pathology have grown exponentially over the past few years. Nowadays, digital pathology can be described as the field of the anatomic and microanatomic pathology information system, which allows not only the visualization but also electronic management, real-time evaluation, comparison, consultation, clinical diagnosis and research, and development of artificial intelligence tools. [2]

Essential advantages over standard pathology among others include the relative permanence of digital slides versus glass ones; the ability to easily access all specimens from the same patient; the possibility to view one or many slides with relative ease on a monitor with mouse-click control; the capacity to annotate and comment region of interest which have immense potential for research and teaching; and, of course, the ability to share digital slides with other off-site pathologists who could collaborate and enhance patient care. [2]

In the following chapter, we will present one of the most recent but rapidly developing enhancements of digital pathology, which is artificial intelligence.

1.1 AI in digital pathology

The potential of Artificial Intelligence (AI) in digital pathology varies from routine tasks, currently undertaken by pathologists, to provide new insights that are not possible by a human observer alone, or to assist pathologists to perform routine tasks in a more effective manner. Although AI tools may assist pathologists to reduce subjectivity and workload in general, the notion that AI would replace them even in simple interpretative judgments is still a source of struggle in the community. [3]

The focus of AI application is currently based around image analysis, such as computer-aided image processing techniques (e.g., filtering, color correction) and user-driven feature extraction (e.g., edge detection, segmentation, intensity thresholding). Newer methods are based on machine-learning algorithms that use statistical methods to model the output data to 'learn' some defined outcome of interest. An example could be the probability that a specific diagnosis is present in the image. An algorithm can be trained with annotated images (supervised learning) or can discover key features by itself (unsupervised learning). [3]

An example of supervised learning, which is being increasingly used in the context of digital pathology, since it does not rely on engineered features and can learn directly from primary data, is a deep neural network-based approach. The process consists of learning on a set of images with associated labels and subsequent interrogation of new input images without pre-existing assumptions. These algorithms are inevitably dependent on precisely annotated digital slide images for classification training. This dependency requires constant collaboration with pathologists who provide domain-specific knowledge and direct scientists towards key attributes that should be examined. Pathologists also represent the standard against which the algorithm are compared. [4]

One of the main challenges when it comes to applying AI, particularly deep learning systems, in medicine is the lack of "explainability" over its decisions, which raises many legal and regulatory issues. At the time, there is active research to make algorithms more comprehensible by humans by providing insights on how they work (e.g., dividing execution into steps or marking key features that the algorithm is focusing on when making a decision). These methods would provide more interpretability and transparency but might come at the expense of performance. [5]

In the next chapter, we will explore WSI (whole slide imaging) as the primary mean of capturing digital pathology images. Other means that will not be mentioned in this thesis but might be preferred for specific use cases include a digital camera on a microscope or streaming video.

1.2 WSI

Whole slide imaging (WSI), also known as "virtual microscopy", refers to generating digital slides from conventional glass ones for diagnostic, educational, or research purposes. From a practical perspective, WSI consists of two phases. The first phase incorporates specialized hardware (scanner) and software (image mosaic assembly/stitching) to digitize glass slides and to produce a large digital image ("digital slide"). The second phase utilizes specialized software to view and analyze these images, which are the main focus of this thesis. [6]

In the following sections, we will examine the acquisition of digital slides and challenges that accompany the processing of these sizable files.

1.2.1 Image acquisition

The first step in image acquisition from the glass slide is a scanner. WSI scanners have a wide range of appearances and functionality to satisfy different market needs. Some devices are designed to scan only a limited number of slides; some can accommodate hundreds. There are scanners designed for smaller slides and also for whole-mount glass slides (e.g., complete prostate gland sections).

The essential components of a WSI scanner include:

- Microscope with lens objective
- Light source
- Robotics to load and move glass slides
- One or more digital cameras
- Computer
- Software to manage digital slides

Slides might be scanned manually or automatically, and several scanners support continuous or random-access processing, where while one slide is being scanned, another can be uploaded. Actual scanning speed varies from under 1 minute to around 1-3 minutes per slide depending on objective magnification (e.g., 20x, 40x) and a number of planes of focus acquired. [6]

Software is also an essential part of image acquisition as it is responsible for focus calibration, compression, stitching, or automatic tissue detection. Some scanners offer their users interactive software to select regions of a slide to be scanned, perform focusing, and adjust white balancing. After a glass slide is scanned, file formatting is required to diminish the size of a raw, uncompressed WSI file. A typical virtual slide, which we worked with in the project, has 20,000 megapixels (native resolution of 100,000 px * 200,000 px) and would require about 60 GB of memory (assuming 24-bit depth, i.e., 3B/px); therefore, several optimization techniques need to be implemented as described in the next section.

1.2.1 Optimization

Since digital slides depict high-resolution microscopic images, it usually has around hundredths thousands of pixels per dimension with several GBs or tens of GBs in size. To process such an image in full size would require considerable computation power. However, pathologists do not always need to see the image in its full resolution, and even in this case, they are concentrating on a specific area. In practice, several techniques are being implemented to accelerate the processing of digital slide images.

Pyramidal format

The image pyramid is a data structure designed to support efficient scaling through reduced image representation. It consists of copies of an original image, which have gradually decreased resolution. The bottom of the pyramid (level zero) is equal to the original image, the next level is filtered and subsampled by a factor of two, and the rest of the pyramid levels are generated by the same algorithm. [7]

By storing pre-computed scaled-down copies of the large high-resolution image, the software can efficiently render a version of the image by reading only the data from the pyramid level that is closest to the currently requested zoom. Storing these levels increase performance at the cost of storage. [8]

Significant area

To reduce the required storage capacity for a digital image, many software products attempt to reduce the actual scan area by spotting significant areas while scanning and only store these in high-resolution. Consequently, the final digital slide contains many sparse high-resolution areas, which may follow an independent pyramidal format. [8]

Tiles

To further improve performance, digital slides are split into smaller areas known as tiles. Each level in the pyramid format mentioned above is de facto, a grid of tiles of the same size (Figure 1). By zooming or moving through the image, only tiles that intersect with the current viewport are being read and displayed, which therefore minimize the number of I/O operations. [8]

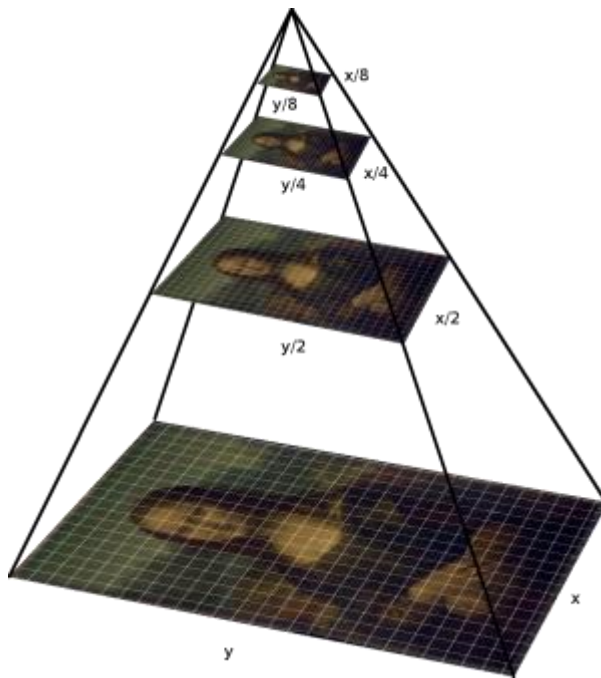


Figure 1 Pyramid tile structure

(Source: <https://iipimage.sourceforge.io/documentation/images/>)

Compression

As was mentioned above, digital pathology images are quite substantial, which leads to the employment of various compression and image representation schemes. Some of the implemented schemes are lossless; some may lead to color information loss. Typical compressions being used are LZW, PNG, Raw, JPEG, and JPEG2000, which will be discussed in the next chapter with other possible file formats currently employed in virtual microscopy. [8]

1.2.2 File formats

Digital pathology currently lacks a universally accepted image format. Although some vendors might support their proprietary file formats for the best performance, commonly used open image standards for digital images are MRXS, TIFF or pyramid TIFF, DZI, and JPEG2000.

MRXS

MRXS stands for the MIRAX scan, and it is a proprietary file format with metadata and indexes. After the slide is scanned by the MIRAX scanner, it is divided into multiple PNG, BMP, or JPEG files and packed into several data files. The slide is stored in the pyramidal format with the highest resolution, usually having 16 tiles per camera position, and the next levels are scaled down by a factor of 2. The index file is used to retrieve the requested piece of data from the data files by providing offsets. [9]

Pyramid TIFF

TIFF is a tag-based file format for storing and interchanging raster images, which can hold multiple images in a single file. The term "Pyramid TIFF" refers to a TIFF file that wraps a sequence of bitmaps that each represent the same image at increasingly coarse resolutions. The individual images may be compressed or tiled. Most image manipulation programs that can handle TIFFs can read the primary (full-size) image of the TIFF pyramid, but some might ignore the smaller images. [10]

DZI

Deep Zoom Image is represented by a tiled image pyramid. This format allows the Deep Zoom rendering engine to grab only the data that is necessary for a particular view. Each resolution of the pyramid is called a level. At each level of the pyramid, the image is scaled down by 4 (a factor of 2 in each dimension¹) until the last level (1x1 pix) is reached. Tiles for each level are stored in a separate folder and are named column_row.jpeg/png, representing the row and column number of the tile, respectively. Apart from the tiles folder, there is also an XML file containing metadata. (Figure 2). [11]

¹ Max Level is calculated as $\lceil \log_2 \max \{image\ height, image\ width\} \rceil$ [12]

```

<?xml version="1.0" encoding="UTF-8"?>
<Image xmlns="http://schemas.microsoft.com/deepzoom/2008"
  Format="jpeg"
  Overlap="1"
  TileSize="254"
  >
  <Size
    Height="221772"
    Width="105185"
  />
</Image>

```

Figure 2 DZI description file

An overlap attribute might be added in addition to the standard tile size. Tile overlaps are on both sides of the join between tiles, so a one-pixel overlap would mean that tiles 0_0 and 1_0 have two columns of pixels in common. The tile size is usually chosen so that non-edge tiles have a size of power of two. With overlap one and tile size 254, corner tile will be 255x255, and all inner tiles will be 256x256 as they are overlapping on both sides. [12]

JPEG2000

JPEG2000 is an image encoding system that was created by the Joint Photographic Experts Group in 2000. It uses state-of-the-art compression techniques based on discrete wavelet transform, enabling lossless compressions and reducing the image size to 2x -2.5x. JPEG2000 supports progressive decoding, which allows a viewer to see a lower quality version of a requested image while the whole file has not been fully downloaded. It also supports transparency and multiple color spaces. On the negative side, JPEG2000 format is not universally supported in browsers, and its encoding is CPU intensive. [13]

2. Project description

The project for which this thesis was conceived applies deep learning into digital pathology. The goal of the project is to develop an algorithm for detecting potentially cancerous areas in the prostate gland section. As a result, a pathologist, who currently must explore the whole slide manually using an optical or virtual microscope, can focus primarily on suspect regions. Consequently, the time necessary for diagnosis should be considerably decreased.

2.1 Workflow

The first step in the project is to obtain scanned slides. Glass slides are scanned by a scanner operated by pathologists; if the image is used for training or evaluating the AI system, the image is subsequently annotated using a specialized program (ASAP - Automated Slide Analysis Platform²). The creation of annotations is a lengthy process as the pathologist must inspect the whole slide and mark regions that have specific features suggesting the presence of cancer in tissue. Annotated images are then sent to scientists who will use them to "teach" the neural network to recognize these regions. Apart from annotated images, the pathologist also sends unannotated slides that are used in the resulting AI-supported diagnostic process.

The outcome from the neural network is a list of regions that were detected as positive for cancer. More precisely, the image is split into tiles, which are classified by the network; the classification is a probability that the tile belongs to the cancerous region. To illustrate, the network can produce a table with probabilities for each tile, which is converted to a PNG image in grayscale (Figure 3), where darker areas correspond to higher probabilities and lighter to lower ones. Regions marked by the pathologist and network are then compared to evaluate the outcome.

² <https://computationalpathologygroup.github.io/ASAP/>

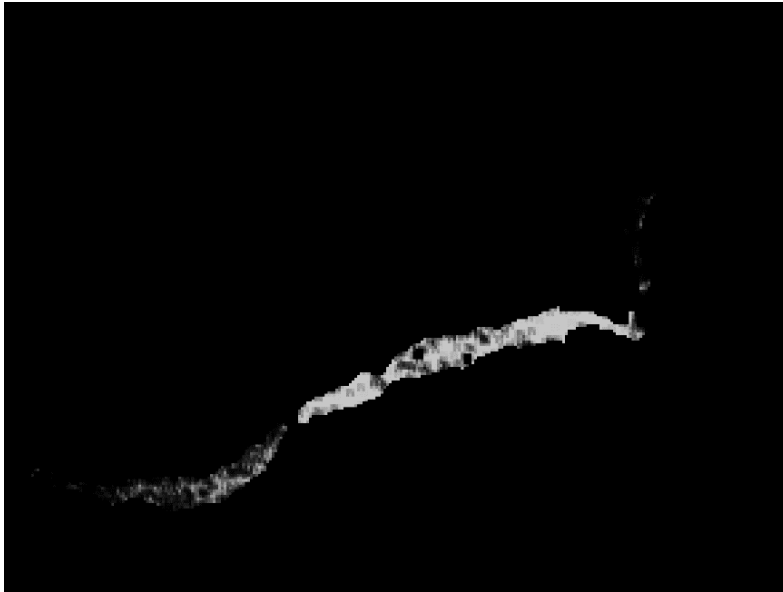


Figure 3 Grayscale probability mask

Another vital part of the process is the interpretability of the results. As was mentioned in the first chapter (1.1), the decision making of neural networks is still considered a black box that causes trust issues, especially in the healthcare domain. That is why different methods are being applied to explain network behavior. The goal is to see which structures of the image influence the decision the most and whether they can be attributed to features recognized by the pathologist.

The last part of the process is the feedback. After the pathologist compares results with his annotations, he might notice some areas, which were marked by the network and not by him. Either these areas are correct and were overlooked by the pathologist in the initial annotations, or they represent false positives (the area was classified as cancerous even though it is healthy). In either case, the pathologist should be able to add the comment to this area and annotate it accordingly so that the network can be retrained. As annotation of precise cancerous regions is rarely pixel-perfect, and even pathologists' opinions might vary for some borderline cases, collaborative annotations are desirable to give the network the best results possible and reinforce its learning.

In the next chapter, we will focus on the requirements for image viewers, which will combine all phases of the project.

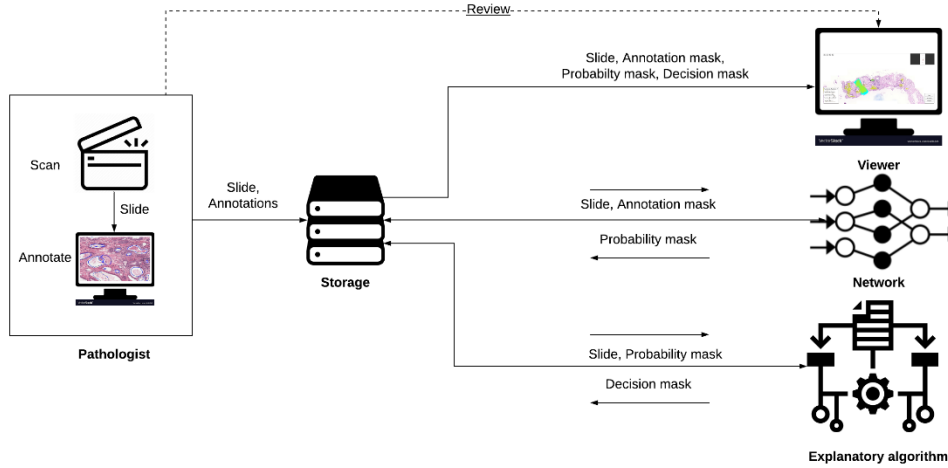


Figure 4 Simplified process overview

2.2 Viewer requirements

The process described above requires besides the neural network also a visualization tool to present final results and to support additional functionality regarding interoperability and collaboration. In practice, the tool must fulfill several requirements listed below.

Handle large high-resolution images

As we mentioned in the first part (1.2), digital slides usually have several GBs or tens of GBs in size; thus, the prerequisite for the image viewer is the ability to work with these files.

Overlays

Besides the original slide, it is also important to visualize annotations from the pathologist and the results of the network for comparison. Both of these layers with the original image should be accessible at the same time as the pathologist needs to focus on the difference between annotations and results and then explore the tissue beneath them. Furthermore, it should be possible to change the opacity of the respective layers.

Another required functionality regarding overlays is thresholding. As was mentioned in the description of the workflow, regions from the network have different probabilities, and the pathologist might choose to visualize only the relevant one (e.g., more significant than 90%). In our case, that means to hide the pixels whose color intensity does not pass the threshold.

Another necessary layer is the one with explainability, which needs to be shown over the other layers with its proper opacity and thresholding control so the pathologist might evaluate the decisive structures determined by the network.

In conclusion, the tool should, besides the original slide, support three overlay images (annotation mask, probability mask, and explainability mask) with the ability to change the opacity for respective layers and characteristics of specific pixels.

Annotation support

As for annotations, pathologists already have the software for the initial annotation; however, they still require adding additional ones to adjust network results. The viewer should allow them to create an annotation over questionable regions and to edit it afterward if needed. The selected tool should also allow adding comments and reviews so multiple pathologists can express their opinion on the particular region.

To sum up, the tool should support the creation of annotations, edit, and collaborative comments. Moreover, more precisely, the annotation will describe the region; better, the network can learn. For that reason, a freehand tool should be available or a type of watershed algorithm which would automatically cover the selected area.

Reasonable performance and storage needs

Since the viewer might be used on slower machines, performance is also an essential factor. As digital images are quite sizable, we need to consider file format, which will be employed. Some viewers require specialized formats for more efficient work; however, these formats might occupy larger storage space and take up some time to be created. The required viewer should adequately balance performance with storage needs.

Interactiveness

Another essential requirement is interactiveness. If the pathologist should use the tool regularly, he needs to understand its functionalities and be satisfied with the performance. For example, the speed of zooming and panning is vital as larger delays are noticeable by the user and disturb his workflow. Another factor is file browsing capability; in other words, how effortless it is for the user to load and browse new images.

Modifiability

Lastly, since all requirements probably will not be present in one tool, some modifications will be unavoidable. As all of our solutions are open-source, it is possible to access the source code; however, it is also necessary to consider how difficult it will be to add or modify any functionality. In conclusion, a well-documented tool with a strong community is preferable.

The next chapter will introduce several viewers, that should meet the requirements mentioned above.

3. Viewers

As was mentioned in the first chapter, due to the large size of the digital images, it is not possible to use conventional image viewing software, which requires the whole image to be loaded in RAM. Image viewers used for digital images work with pyramidal file formats, which allows them to display only the requested area of the image.

Based on the project description and user requirements, four tools have been selected to undergo a detailed examination. In the next part, we will divide these viewers into two categories: WSI viewers which apart of basic functionality (manipulating images) often offers more sophisticated functions (analysis, annotations...); and light-weighted image viewers adjusted for work with high-resolution images which mostly rely on the community for additional functionality and which were not primarily developed for digital pathology. We will also focus only on open-source software in both categories.

3.1 WSI viewers

The advance in digital pathology and a growing number of images to manage, visualize and analyze created a need for mature viewing tools. In recent years, several developers were attempting to fill this gap with their specialized software offering a wide range of functionalities required by pathologists.

3.1.1 Cytomine³

Cytomine software is developed by a group of researchers from Montefiore Institute (University of Liège, Belgium), and it is an open-source internet platform that allows collaborative analysis for large (multi-gigapixel) images and their semi-automatic processing using machine learning algorithm. The software can be employed in education, research, or diagnostic domain where collaborative analysis is required.

³ Source: doc.cytomine.org

In the following image, we can observe possible workflow for image manipulation from capturing through annotating and reviewing to the final report.

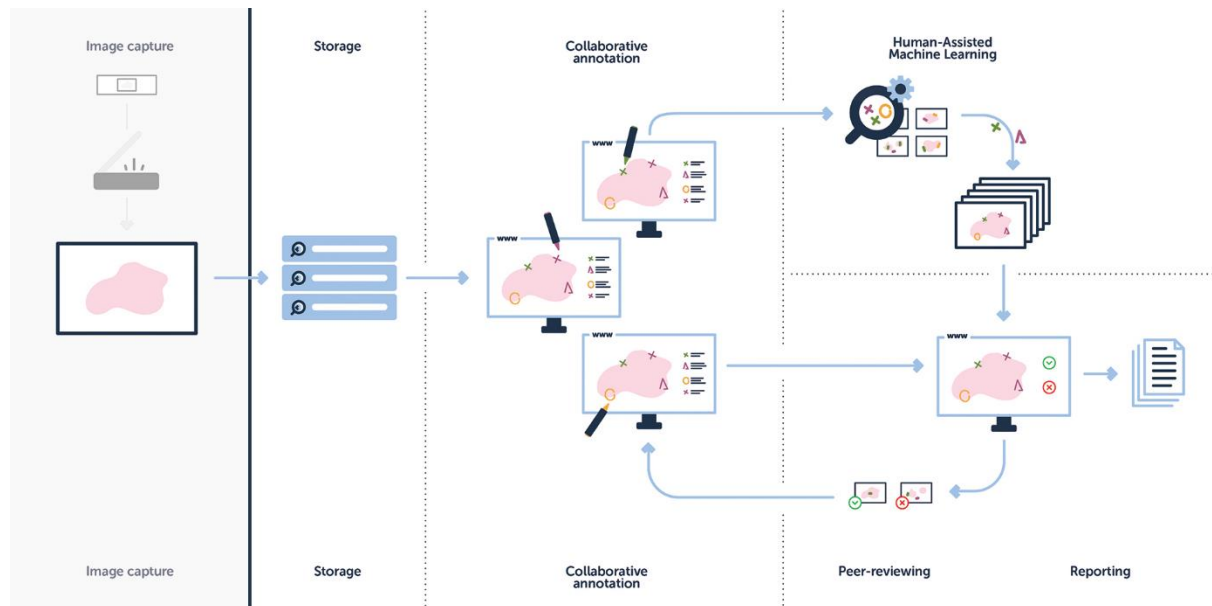


Figure 5 Cytomine workflow

(Source: <https://doc.cytomine.org/HomePage>)

Typical functionalities offer by Cytomine are:

- Storage: distributed and scalable whole-slide image storage provides immediate access to images and allows them to organize them into projects. It supports several file formats (JP(E)G, PNG, BMP, TIF, TIFF, SVS, SVS, SCN, NDPI, VMS, MRXS, VSI.)
- Collaborative Annotations: allows real-time collaborations on WSI images with semantic annotation tools and user-specific layers
- Human-assisted machine learning: custom-made tools might assist with the quantification of histological features or biomarker expression.
- Review and report: allow reviewing annotations and results by pathologists all over the world and generating automatic reports.

Open-source software is also commercialized by The Cytomine company. The company proposes services related to open source Cytomine software such as hosting, installation, and configuration or enhancing its functionalities and on-demand machine learning algorithms. This solution might be beneficial for hospitals or well-established companies with routine diagnostics.

3.1.2 HistomicsTK⁴

HistomicsTK was created in collaboration with Kitware, Inc. and Emory University as an open-source platform for the management, visualization, and analysis of large images focused on digital histology. It is a Python and REST API for the analysis of Histopathology images.

HistomicsTK is developed in coordination with Digital Slide archive (open-source platform for the visualization and management of whole-slide digital pathology image data) and `large_image` (Python module designed to manipulate large multi-resolution images).

HistomicsTK can be used in two ways:

- Python package: employed mainly for its analytical functionality. It provides algorithms for color normalization, color deconvolution, feature extraction, cell-nuclei segmentation, and tissue detection.
- Server-side Girder plugin: As Girder plugin HistomicsTK gains its capabilities (e.g., storing data, authentication, access control, data upload/download). HistomicsTK depends on other three Girder plugins which are girder worker (task execution and monitoring), `large_image` (work with multi-resolution image) and `slicer_cli_web` (RESTful access to image analysis pipelines)

Girder is an open-source Data Management Toolkit built-in Python by Kitware. It can be used as a platform for new web applications that require data organization (storing data from heterogeneous backend storage through a single web API), user management and authentication (securely store credentials), authorization management (user-based and role-based access control).

3.2 Image viewers

Open source light-weighted image viewers were developed to satisfy the needs of a high-resolution viewer in several domains as art, geography, medicine, or astronomy. The main goal is to comply with the most popular image standards (e.g., IIF) being flexible, customizable, and easily employed. The initial functionality of these viewers is usually limited, and plugins are developed, as the new requirements arise, either by the original development team or by the community.

⁴ Source: <https://digitalslidearchive.github.io/HistomicsTK/>

3.2.1 IIPMooViewer⁵

IIPMooViewer is a light-weight HTML-5 based JavaScript high-resolution image streaming and zooming client compatible with the majority of web browsers. It offers multi-layers images and support for annotations of regions within the image. It was designed to work with an IIPImage server; however, it can also work with Zoomify, Deepzoom, Djabatoka, and IIF protocols.

IIPImage is an advanced image server system for web-based viewing high-resolution images. The server can handle gigapixel size images and advanced image features, and it is still bandwidth-efficient with low processor and memory requirements. Streaming is tiled-based, which means that the whole image does not need to be downloaded at once, and it makes the server very scalable. Supported image formats are TIFF or JPEG2000. IIPImage is used in a variety of domains, including art, astronomy, science, medicine, geographical information systems, and many others.

3.2.2 OpenSeadragon⁶

OpenSeadragon is an open-source JavaScript-based web viewer for high-resolution zoomable images. It supports several server protocols, and it can work with a variety of zooming images consisting of tiles such as DZI, TMS, Zoomify, IIF. It is an implementation of Seadragon technology developed in 2003. Seadragon technology is a web visualization technology for smooth browsing of extremely large files without any latency, which was supporting former Microsoft systems Silverlight, Pivot, or Photosynth.

Even though without modification, OpenSeadragon offers only essential functionalities (zoom and pan), some optional controls can be easily enabled (toolbar, rotation, overlays, multi-image) or added as a plugin (annotations, magnifier, navigator, screenshot...) developed by contributors.

⁵ Source: <https://iipimage.sourceforge.io/>

⁶ Source : <http://openseadragon.github.io/>

4. Methodology

We have already established the requirements for the viewer and selected several tools that should comply with them. The following step is to set criteria for each of the requirements to objectively compare these tools and choose the best fitting for the described project. We will not be focusing on the first requirement as all tools were chosen due to compatibility with high-resolution images. For the rest of the requirements, we will specify criteria and corresponding measurements.

Overlays

For overlays, we will be concentrating on four criteria:

- Does the selected tool support overlays?
- If so, does it support at least four layers?
- Is it possible to set the opacity for respective layers?
- Is it possible to set thresholding?

All of the requirements listed above are the "must" for our viewer, and if they are not already incorporated in a viewer, they must be added programmatically afterward.

This category will have only nominal results for each tool. We can respond either yes, no, or partially (some tools support overlays in a different format that is required).

Annotation support

For annotation, we will have five criteria:

- Does the selected tool allow the creation of annotations?
- Is there a freehand draw available?
- Can existing annotations be edited afterward?
- Is it possible to add a comment or a review to an existing annotation?
- Is the watershed algorithm available?

As for the overlay category, we will assign nominal values for each criterion (yes, no, partially). The "must" functionality is the creation and editing of annotations and a freehand tool. User review belongs in "should have" category, and finally, the watershed algorithm is "nice to have" for our program.

Reasonable performance and storage needs

For storage and performance, we will measure two criteria:

- Overall file size
- Convert time (how much time does it take to convert image from original to desirable format)

We will use identical images for the testing; however, different formats might be employed as not all of them are supported by each viewer. For the conversion, we will be using libvis (image processing library) to change the original MRXS image (TP-2019_2824-01-1.mrxs) to a desirable format (if MRXS are not already supported). We will run the same command five times on the same machine and measure the execution time. Finally, we will use the median to compare the respective results.

For the file size, we will compare the size of the final folders, which will contain the original image, annotations from the pathologist, and probabilities exported by the network.

The viewer, which will require the least conversion time and space, is preferred.

Interactiveness

For interactiveness, we will focus on two criteria:

- Is the zoom/pan continuous?
- Can the user easily browse files?

An important factor for work with high-resolution images is an implemented method of zoom/pan. Some viewers implement continuous zooming, where a new set of requested tiles are, although blurred, immediately visible. This method is preferred from the one where new tiles are preceded by the black area until fully loaded.

As for file browsing capability, we will answer the question of whether the user can browse new files intuitively via existing GUI.

Modifiability

For modifiability, we will look at three criteria:

- Comments in code
- Community
- Documentation

Comments will be evaluated on a nominal scale with yes (each function is fully commented in the code, with expected behavior and input and output arguments) partially (comments are present in the code but gives an only high-level view on the functionality), no (there are no comments to help understand the code).

For community evaluation, we will examine the number of contributors on GitHub, number of questions on StackOverflow, and since the response from the development team is also a key factor, we will inspect the number of issues on GitHub with focus on the number of closed ones and the timeline. The desirable tool should have several contributors outside of original developers, should have strong representation on the selected forum, and an active response on issues within the last year.

Lastly, for the documentation, we will again choose a nominal scale with yes, no, and partially. For an easier modification, the tool should have accessible documentation with the list of functions and an explanation of architecture (especially if it is a more sophisticated solution).

5. Analysis

As the criteria for comparison are set, we will now analyze each category and evaluate respective viewers. For nominal values, we will use tables to arrange results, for ordinal results, graphs and charts will be incorporated to visually support differences.

Overlays

IIPMooViewer

IIPMooViewer supports overlays by default. It is possible to run the viewer with several images; however, only two layers are supported to be shown at once; the rest of the images can be chosen from the drop-down menu. The viewer supports blending between images as well, which in practice means altering the opacity of the top layer. To conclude, the viewer lacks two additional overlays and thresholding functionality.

OpenSeadragon

OpenSeadragon supports overlays (several images or geometrical shapes set in CSS). We can load many images at once; however, we cannot change their opacity by default. The advantage of OpenSeadragon is that it supports transparent images (otherwise background needs to be manually filtered out). In conclusion, the viewer requires the implementation of an opacity control and thresholding functionality.

HistomicsTK

HistomicsTK does not support several images overlay. However, if the images can be converted to vector coordinates, they can be visualized as annotations. Another obstacle is opacity control, which is the same for all layers, so we cannot set different opacity for annotation and probability layer. Implementation of thresholding would require redrawing of annotations vectors or storing several annotations files with a fixed threshold (e.g., for 50%, 80%, 90%). To sum up, overlay images must be converted to vector coordinates, and opacity control needs to be modified.

Cytomine

Cytomine does not have a default option to show multiple images over each other (although it supports side by side viewing of two images). Similar to HistomicsTK, images can be loaded as annotations and be shown in separate layers with shared opacity control, which would need to be adapted to our use case regarding thresholding.

Summary

Viewer	Overlay support	Four layers	Separate opacity	Thresholding
OpenSeadragon	YES	YES	NO	NO
IIPMooViewer	YES	NO	YES	NO
HistomicsTK	PARTIALLY	PARTIALLY	NO	NO
Cytomine	PARTIALLY	PARTIALLY	NO	NO

Table 1 Overlay summary

From the table above, it is already apparent that none of our viewers comply with all overlay requirements. The rest of the functionalities needs to be further developed. From the default state, only OpenSeadragon and IIPMooViewer completely satisfy two of the "must" criteria.

Annotation support

IIPMooViewer

IIPMooViewer, by default, supports square annotations with title, category, and a text field. All of these fields, together with annotation size and position, can be altered or deleted. Even though comments can be added, there is no notification or user management.

OpenSeadragon

OpenSeadragon does not offer annotation tools by default, but it has two plugins that provide overlays for complex annotation libraries: Fabric.js⁷ and Paper.js⁸. Both of these libraries enable the creation and modification of canvas elements (from points and rectangles to complex paths) and provide multiple examples and guidelines. If the viewer is selected for implementation, one of these plugins will be modified to fulfill annotations requirements.

⁷ <http://fabricjs.com/>

⁸ <http://paperjs.org/>

HistomicsTK

HistomicsTK supports the addition of annotations (point, rectangle, polygon, line) and their classification into groups. However, after annotation has been created, only information (name, description, color, line width), not annotation itself (shape, size) can be changed. Also, no review or comments control is available, meaning only the option to evaluate annotation is to add the comment into a description field.

Cytomine

Cytomine is known for its sophisticated annotation tools. New annotations can be created by different shapes, points, lines, or by freehand. Each annotation attribute can be modified, e.g., size, position, name, ontology, author, date, and set of custom properties with key and values. Another attribute of annotations are comments, which can be added with notification to all contributors or just to selected users. Finally, the whole image can be reviewed (each annotation for the image can be accepted or rejected). In practice, after a pathologist marks some arguable region, other collaborators get notifications to review the given annotation, which will lower the subjectivity.

Summary

Viewer	Annotation support	Freehand draw	Edit	Groups	Comments/ Reviews	Water shed
OpenSeadragon	PARTIALLY	YES	YES	NO	NO	NO
IIPMooViewer	YES	NO	YES	NO	PARTIALLY	NO
HistomicsTK	YES	PARTIALLY	PARTIALLY	YES	PARTIALLY	NO
Cytomine	YES	YES	YES	YES	YES	NO

Table 2 Annotation summary

As for annotation, Cytomine is far ahead from its competitors as it complies with all of the project requirements except for the watershed algorithm, which, similarly to thresholding (overlay section), is not a commonly requested functionality. For the rest of the viewers, annotation functionality needs to be heavily modified.

Reasonable performance and storage needs

IIPMooViewer

IIPMooViewer handles JPEG200 and pyramidal TIF files thanks to IIPImage, which provides requested tiles. In practice, source image (.mrxs), annotation mask (.png), and probabilities heatmap (.png) needs to be converted into pyramidal TIF.

OpenSeadragon

OpenSeadragon, by default, supports DZI images (1.2.2) or simple images without pyramidal format such as PNG and JPEG. In practice, source image needs to be converted to DZI, and annotation and probability masks can be displayed as simple PNG files. OpenSeadragon also supports several image serving protocols (Zoomify, Deepzoom, IIF...) that can provide DZI zooming tiles from different pyramidal formats.

HistomicsTK

Thanks to large_image and OpenSlide libraries, HistomicsTK should handle multiple file formats, including MRXS. As for annotation and probabilities, correct XML files with vector description needs to be created as original annotation from pathologist uses different coordinates order.

Cytomine

Cytomine also supports several file formats, as was mentioned in chapter 3.1.1. The source image does not need to be converted (MRXS is supported); however, annotations and probabilities XML files need to be adjusted.

Summary

Viewer	Conversion time	Storage needs
OpenSeadragon	109min (MRXS to DZI ⁹)	0,79GB (2x PNG, 1xDZI)
IIPMooViewer	4min 40s (MRXS to pyramidal TIF ¹⁰)	0.65GB (3x pyramidal TIF)
HistomicsTK	0	2.2GB (1 MRSX, 2x XML)
Cytomine	0	2.2GB (1 MRXS, 2x XML)

Table 3 Performance summary

Several factors need to be considered before the evaluation of the result. Although the XML file does not occupy storage space, it would be inefficient for thresholding calculation and would need to be converted to a different pyramidal file format. Furthermore, even though OpenSeadragon supports PNG images, operate (e.g., recalculate pixel attributes) with a high-resolution image without pyramidal structure would notably increase manipulation time. In conclusion, the summary table serves only as an initial overview, and image formats for the chosen viewer will be adjusted to gain the best performance.

Interactiveness

IIPMooViewer

Although IIPMooViewer was designed for the visualization of high-resolution images, manipulation with images themselves is rather inefficient. When zooming the image, a new level of tiles must be downloaded and displayed, which is noticeable by the user as they are preceded by the black area (Figure 6). It is similar for the panning when tiles in a new viewport position must be downloaded.

IIPMooViewer also does not offer any file browsing capability, and a path to images needs to be set in the indexing file.

⁹ time vips dzsave ".../TP-2019_2824-01-1.mrxs" ".../TP-2019_2824-01-1" --compression = 9

¹⁰ time vips tiffsave ".../TP-2019_2824-01-1.mrxs" ".../TP-2019_2824-01-1.tif" --tile --pyramid --compression=jpeg --Q=80 --tile-width 256 --tile-height 256 --bigtiff



Figure 6 IIPMooViewer mid-zooming

OpenSeadragon

Zooming and panning in OpenSeadragon is smooth without any noticeable delay; however, as the viewer is just a web application for visualization (as IIPMooViewer), it does not offer any file browsing capability. New images must be configured in the initial HTML file, which is not convenient for ordinary users.

HistomicsTK

HistomicsTK also offers zooming and panning without noticeable delays. Since we used the viewer as a girder plugin, file browsing capability is also covered. Each user has its profile with storage where he can upload WSI. After the initiation of HistomicsTK, files from the girder repository can be selected. The only difficulty is with annotations, as HistomicsTK does not support overlays, images (annotation and probability mask) must be converted to specific vector coordinates and load via API. This approach seems impractical for end-users, and some shortcuts should be implemented.

Cytomine

Cytomine also provides smooth image manipulation as tiles are efficiently requested by Zoomify protocol and displayed. Another advantage is incorporated file browsing capability. Users can create a project with its proper description, terminology, and sets of images and contributors. Images can be added by drag and drop option or by a classical folder browse. The drawback is the same as in HistomicsTK, and it is a conversion of an annotation and probability mask image to vector coordinates and subsequent load by API. Not only is it impractical for the user, but it is also an obstacle for the thresholding computation.

Summary

Viewer	Continuous zooming/panning	File browsing capability
OpenSeadragon	YES	NO
IIPMooViewer	NO	NO
HistomicsTK	YES	YES
Cytomine	YES	YES

Table 4 Interactiveness summary

Regarding continuous movement around the image, all the viewer except for IIMPooviewer provides satisfactory experience. Furthermore, HistomicsTK and Cytomine are equipped with user-friendly file browsing capability.

Modifiability

IIPMooViewer

The first version of IIPMooViewer was developed in 2007 as a compatibility tool for the IIPImage server. Although the development of IIPImage is still undergoing, IIPMooViewer seems rather obsolete, and no new features should be expected in the near future, as the last active issue was at the beginning of 2019. When it comes to community, there is but one topic on StackOverflow regarding this viewer and only a small number of GitHub contributors (responsible for less than 10% of commits in total).

On the other hand, even though documentation is mostly concerned about initial parameters and options to properly set the viewer, after some examination, code is fairly simple to understand with several comments to each important functionality. Additionally, the main developer is likely to respond to any issue that might arise.

OpenSeadragon

As can be seen on charts below OpenSeadragon community is still active. Several contributors provide useful plugins (annotations, overlays, magnifiers, screenshots...). Moreover, main developers are actively responding to any issue and even illustrate how to implement some common functionalities on the CodePen page.

On the issue of code comprehension, source code is thoroughly commented, and there are also well-documented API functions that should be called if any modifications are required. Apart from developed documentation, the main web page contains examples of how to correctly set the viewer with different file formats, additional features, and customization.

HistomicsTK

Although from Figures 8 and 9, it is apparent that HistomicsTK also has its community and active issues responds, questions concern mostly user perspective. Focus on users also applies to the documentation. HistomicsTK provides a full user guide on how to operate the viewer and run analysis; however, any other documentation about architecture or available functions for web client modifications is not reachable. As for the code, several comments are presented throughout the scripts.

Cytomine

Cytomine is composed of multiple components written in different programming languages. We will be examining Cytomine-Web-UI component written in VueJS (although modifications might also need to be implemented in the Image management server developed in Grails). As the charts suggest, there is no strong community outside of the leading developers for this platform. Some comments are present in the code; however, no functionality is adequately explained.

On the matter of documentation, a simple architecture graph on how different components communicate is available on the web site. Analogous to HistomicsTK, guides are concentrated on the usability of the product rather than possible modifications.

Summary

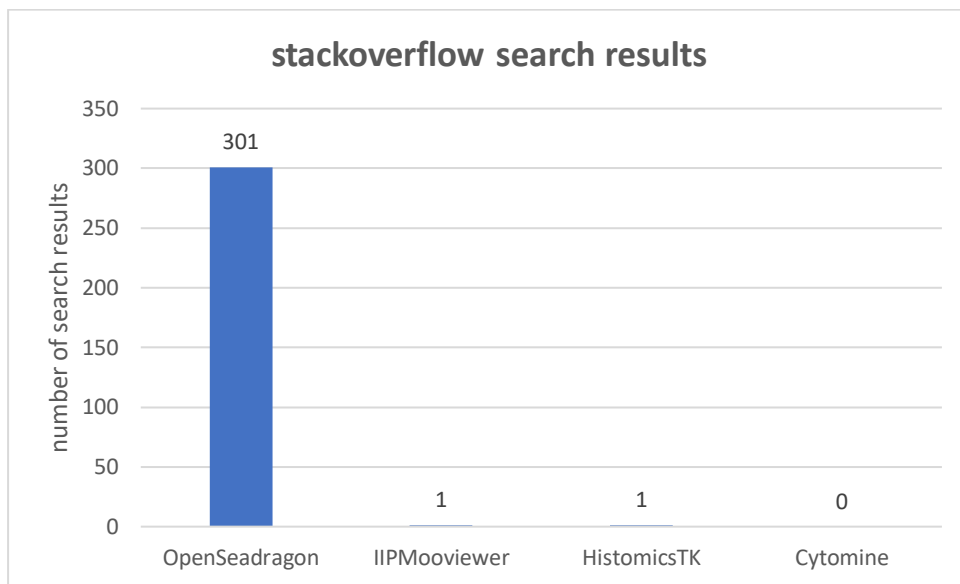


Figure 7 Stackoverflow search results

The chart above illustrates the number of StackOverflow questions regarding each viewer. It is apparent that only OpenSeadragon is represented in the result, which suggests its widespread use among the community.

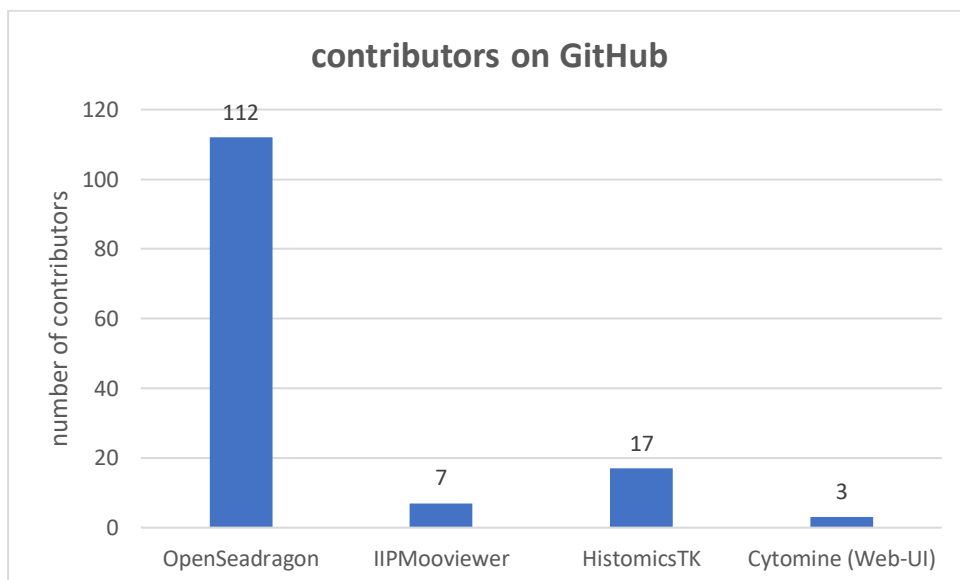


Figure 8 Contributors on GitHub

Regarding contributors, results confirm OpenSeadragon increased popularity in comparison with its competitors. Although the number of HistomicsTK contributors also cannot be neglected.

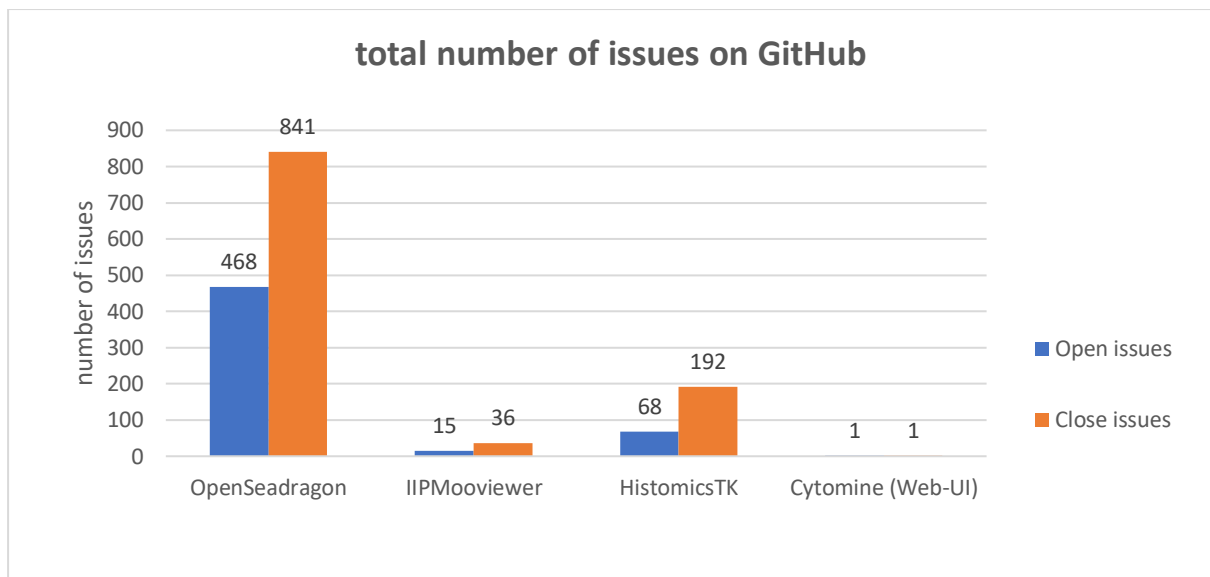


Figure 9 Issues on GitHub

Before we compare the number of issues on GitHub, we need to remark the year when individual projects had been created. OpenSeadragon started on GitHub in 2013, IIPMooviewer in 2012, HistomicsTK in 2016, and finally, Cytomine in 2019. Considering the length of their activity, both OpenSeadragon and HistomicsTK seem to have a substantial number of issues with a similar rate of open/closed issues. Results imply that both of these viewers have an active community, and the development team assigned to issue handling.

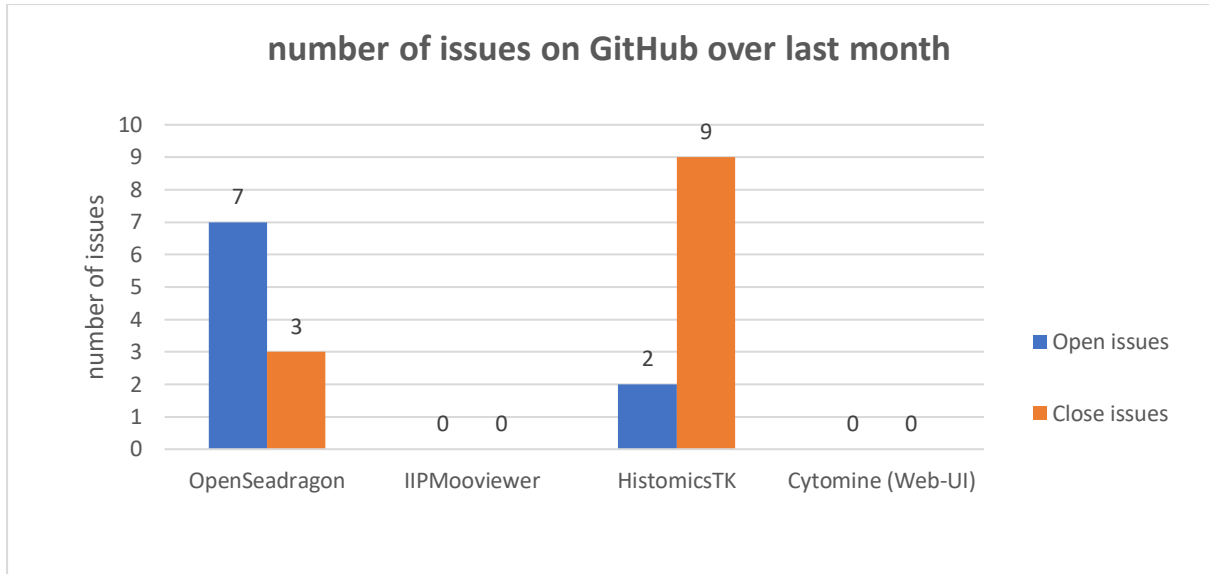


Figure 10 Issues on GitHub (February 2020)

The last chart summarizes issues on GitHub from last month (February 2020). Results suggest the active development of both OpenSeadragon and HistomicsTK.

Viewer	Comments	Community	Documentation
OpenSeadragon	YES	YES	YES
IIPMooViewer	PARTIALLY	PARTIALLY	NO
HistomicsTK	PARTIALLY	PARTIALLY	NO
Cytomine	PARTIALLY	NO	PARTIALLY

Table 5 Modifiability summary

Since both HistomicsTK and Cytomine are complex tools not only for the visualization but also for analysis, any modification would require profound code and architecture knowledge. In our use case, OpenSeadragon has the best conditions for future modification as it is under active development with sufficient documentation.

5.1 Analysis conclusion

Overall results of the analysis clearly state that none of the examined viewers fulfilled all stated requirements. However, each of them presents substantial assets in several categories.

IIMPooviewer – the main advantage of this viewer is its simplicity. The viewer already offers overlays with opacity control, and due to the comprehensible code, implementation of the missing functionalities is feasible. On the other hand, the viewer does not provide any file browsing capability and is not supported by an active community for future updates. Moreover, its primary limitation is ineffective zooming, which would negatively affect user experience.

OpenSeadragon – the main advantage of OpenSeadragon is an active community and its application in several domains. The viewer also provides the documentation for modification functions with several implementation examples. The drawback of the viewer includes missing annotation support, which needs to be correctly implemented from one of the existing plugins and lacking file browsing capability.

HistomicsTK – HistomicsTK offers an efficient zooming, file browsing capability, and essential annotation support. Regarding disadvantages, without proper documentation and architecture description, any modifications would be challenging, and frequent communication with the development team would be necessary.

Cytomine – Cytomine excels primarily in user experience. File browsing capability offers multiple functionalities, and user management facilitates annotations creation and subsequent reviews. However, the project is notably complex, and necessary modifications would require changing multiple modules in different programming languages.

To conclude, since overlays and thresholding represent the crucial requirement for our project and none of the tools satisfy them, deciding criteria will be modification feasibility. Moreover, the whole project is still under development, and new requirements might arise in the future, which supports the reason why the best fitting viewer will be OpenSeadragon.

IIMPooviewer, although used at the begging of the project, is not the final option due to inadequate zooming. HistomicsTK and Cytomine are well-designed tools for digital pathology; however, it would be challenging to modify them to meet our requirements. Both would be an adequate choice for a project requiring their existing functionalities and advanced algorithms.

In the next chapter, we will explore alterations made on the OpenSeadragon to overcome its drawbacks mentioned in the analysis chapter.

6. Design and implementation

As the project is already ongoing for more than a year, some visualization tool was needed to evaluate preliminary results. IIMPooviewer was selected for this purpose as it offered two layers with opacity control by default. However, OpenSeadragon was chosen as the final solution; thus, some of the existing code will be modified for the new viewer, and additional required functionality will be implemented. In chapter 6.1, we will focus on the design of the viewer, and in 6.2, we will present necessary code modifications.

6.1 Design

OpenSeadragon does not have any dependencies, and after downloading necessary folders with JavaScript (openseadragon.min.js) and icon images, the viewer is ready to be deployed on any web page by creating a viewer object in an HTML file (Figure 11). Our instance of the viewer is running on an Apache server with IPIImage as the FCGI module.

```
<div id="osd" style="width: 800px; height: 600px;"></div>

<script src="./openseadragon.min.js"></script>
<script type="text/javascript">

var viewer = OpenSeadragon({
  id: "osd",
  prefixUrl: "images/",
  tileSources: "pathToFile.dzi"
});

</script>
```

Figure 11 OpenSeadragon initiation

The image in OpenSeadragon is represented as a tile pyramid. Initially, minLevel (by default set as 10 but can be altered) of tiles is loaded, and by zooming the image, tiles from lower levels are called. The figure below (Figure 12) depicts how the viewer visualizes initial level 10 and level 18 (max level ¹¹), which represents the native resolution of our image.

¹¹ $\lceil \log_2 \max\{105185, 221772\} \rceil = 18$

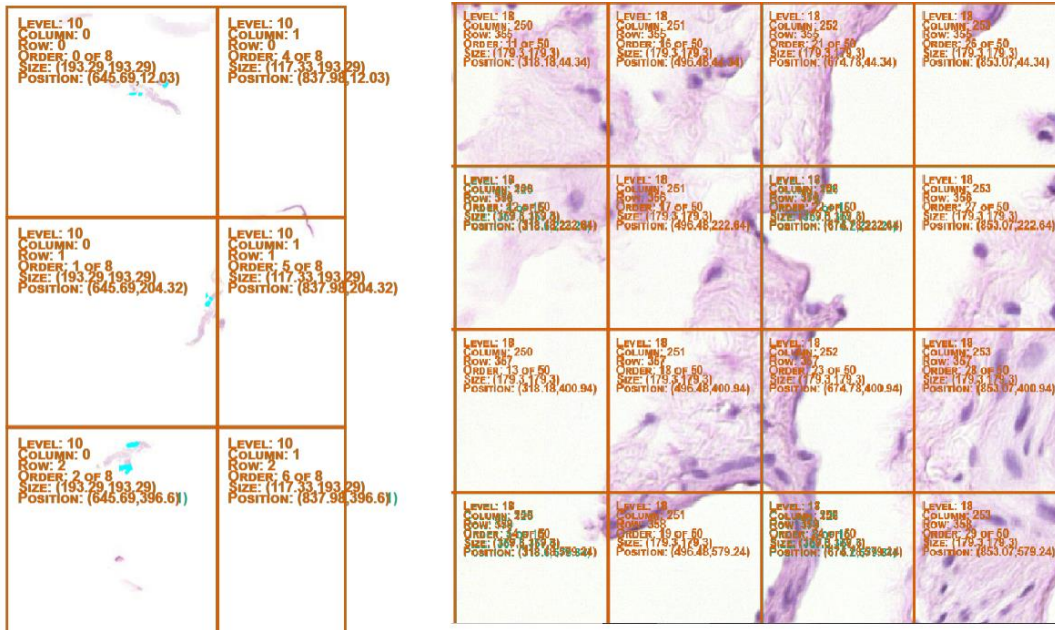


Figure 12 Tiles in OpenSeadragon

In order to visualize the correct tile, the viewer must retrieve its location. Location can be obtained from the `tileSources` attribute in viewer initialization. `PathToFile.dzi` file contains necessary information about the image (Figure 2), and all its tiles are stored in the `pathToFile_files` folder. When new tiles are required, the viewer calls `getTileUrl` function (Figure 13).

```
/**
 * @function
 * @param {Number} level
 * @param {Number} x
 * @param {Number} y
 */
getTileUrl: function( level, x, y ) {
    return [ this.tilesUrl, level, '/', x, '_', y, '.', this.fileFormat, this.queryParams ].join( ' ' );
},
```

Figure 13 `getTileUrl` function

(Source: <https://openseadragon.github.io/docs/openseadragon.js.html#line12171>)

In practice, for the first tile in level 10, the `getTileUrl` function would generate the following URL: `.../srcimg/TP-2019_6786-01-1.tif_files/10/0_0.jpg`. As was mentioned in the chapter 1.2.2, DZI format stores tiles in separate folders named after levels (`/10`), and individual tiles are named after column and row (`0_0.jpeg`). To load different file formats than DZI, the `getTileUrl` function needs to be modified to retrieve correct tiles. Another option is to use a file serving protocol that simulates the DZI structure, as we illustrate in implementation chapter 6.2.

Concerning plugins, OpenSeadragon does not require any formal plugin architecture. New functionality or enhancement can be added to the new JavaScript file that is intended to be loaded after `openseadragon.js`.

From existing functionalities: we can add several images as a tile source, which will create overlays. Moreover, an index parameter can be used to ensure their order. The viewer also supports efficient and continuous zooming and panning on high-resolution images. Other necessary functionalities need to be implemented.

There are several challenges to overcome before the viewer can comply with the majority of the project requirements. Specifically: the change of opacity for individual layers, thresholding computations, efficient creation of DZI files, annotation support, and file browsing capability.

In the next chapter, we will implement these modifications to achieve the requested environment.

6.2 Implementation

All implemented viewer alterations were done in the index HTML file (index.html), which consists of CSS, HTML DOM, and necessary scripts (written in JavaScript or jQuery for easier manipulation of HTML elements). Annotation support was added with separate JavaScript file (osd_fabric_annotation.js).

Overlays with thresholding

Overlays are created by passing an array of images to tileSources (Figure 11) option while initializing the viewer, or by built-in addTiledImage method after the viewer was initialized. Opacity can be altered by setOpacity method on a particular image.

To implement an opacity control, we need to acquire:

- index of the image whose opacity is about to change
- user input for the desired opacity value

Although an index for corresponding images can be set during the initialization of the viewer, in our use case number of overlays varies (not all images include annotations, explainability masks, or generated probabilities). Consequently, a function that will return the index for each overlay after images are loaded is required. Handler was added on viewer open event, and the user-defined function getIndex is called to store indexes for annotation, probabilities, and decision layers if these layers were provided; otherwise, the index will be an empty string.

The slider was created in a combination of HTML (range element with a scale from 0 to 1) and CSS to assure a fixed position on the page. The event listener was set on the input to the slider, and its value is used as opacity for the corresponding layer (Figure 14).

```
496  /* annotation opacity control
497  - on slider change, set the current value of slider
498    as an opacity for annotation layer(if loaded)
499  */
500  $(".sliders-wrapper #annot-layer").on("input",function (){
501    var val = $(this).val();
502    if( a_index ){
503      viewer.world.getItemAt(a_index).setOpacity(val);
504    }
505
506  });
```

Figure 14 Opacity control for the annotation layer

Another necessary functionality is thresholding. For thresholding implementation, user input via slider is required. After the value of the threshold is obtained, the whole image (e.g., probability mask) needs to be recalculated so that only areas with a sufficient color intensity are displayed.

The event listener is set to the threshold slider, and after its change corresponding image will be reloaded (loaded anew and replace its predecessor on the same index position). Loading the image (same as zooming or panning) evokes the tileLoaded event. Handler on this event will call the user-defined changePixel function.

The purpose of the changePixel function is to create a new canvas (with dimensions of currently loaded tile) and redrawn only desired pixels.

For the probabilities layer, the following algorithm is used (Figure 15):

```
421     var tiledData = renderedContext.getImageData(0, 0, canvas.width, canvas.height);
422     var n = event.image.width * event.image.height;
423
424     // probability layer
425     if( event.image.src.includes(urlProbabilities)){
426         for (var i = 0; i < n; i++) {
427             var pix = tiledData.data.slice(i*4, i*4+4);
428             if(pix[0]+ pix[1]+pix[2] < 10){
429                 pix[3] = 0;
430             }else if((pix[0]+pix[1]+pix[2])/3 >= (threshold*255.0/100.0)){
431                 pix[3] = (pix[0]+pix[1]+pix[2])/3;
432                 pix[0] = 210;
433                 pix[1] = 235;
434                 pix[2] = 0;
435             }else {
436                 pix[3] = 0;
437             };
438             tiledData.data.set(pix, i*4);
439         };
440     }
```

Figure 15 Thresholding computation for probabilities mask

If the currently loaded tile originates from the probability image (l. 425), the tile is sliced to individual pixels (l. 426). Each pixel is stored as an array with red, green, black, and alpha values. In the first condition (l. 428), black (background) pixels are eliminated (opacity is set to 0) to ensure transparency of the image.

The following condition (l. 430) handles the threshold. Initially, the probability mask is generated in grayscale (Figure 4), which means that the intensity of each color is the same, therefore by dividing RGB values by three, we will get the final intensity on a scale from 0 to 255. This intensity is compared to the threshold (e.g., if the threshold is set to 50%, color intensity needs to be at least 128). Pixels which satisfy the condition will have an alpha value set according to their original intensity (pixels closer to the threshold will be more transparent) and RGB values set to 210,235,0 (yellow-ish) for better visibility on the tissue. Pixels that do not pass the threshold (l. 436) will have alpha set to 0 and thus will not be visible.

A similar algorithm is used for the decision layer (e.g., occlusion mask), where the initial mask had only green and red shades instead of grayscale. The algorithm might also contain another transformation (logarithmic scaling) to enhanced visualization, as the distribution of pixels intensity might be specific (close to zero). As for the annotation layer (which does not require any thresholding as it depicts annotations from the pathologist), all pixels, except for the black background, will be redrawn (with changed color for better visibility).

After the described modifications, the user can change the opacity and set the threshold for individual layers. Remaining challenges include creating annotations, effective generating of DZI files, and file browsing capability.

Annotations

After experimenting with both, Paper.js¹² and Fabric.js¹³ plugins, Fabric.js library¹⁴ was selected for implementation of annotation support since there was already an existing project¹⁵ that provided fundamental functionalities enabling to create simple annotations forms. Without this project, all interactions necessary for an annotation manipulation would need to be implemented as the initial plugins offer only a clean canvas overlay that resizes and moves with the viewer. The following section will describe the necessary modifications of the mentioned project to suit our use case.

¹² <https://github.com/eriksjolund/OpenSeadragonPaperjsOverlay>

¹³ <https://github.com/altert/OpenSeadragonFabricjsOverlay>

¹⁴ <https://github.com/fabricjs/fabric.js>

¹⁵ <https://github.com/WilliamCarlos/OpenSeadragonAnnotationPlugin>

Since the original project offered only rectangle and elliptical annotations; polygon drawing and editing were implemented to allow pathologists to annotate custom areas. Recalculation of coordinates was also enhanced since, even though Fabric.js allows modifying size and position of existing annotations, it does not recalculate their coordination attributes by default.

Another modification concerned input form (Figure 16), which will allow pathologists to insert information about an annotation (the type of region, additional commentary). The input form was implemented as an HTML table, which is displayed when the mouse cursor is positioned over an annotation. The form can be edited together with annotation in edit mode. Apart from manually inserted information, the input form also contains the following automatically generated attributes: type of annotation and the probability threshold value that was set when the annotation was being drawn.

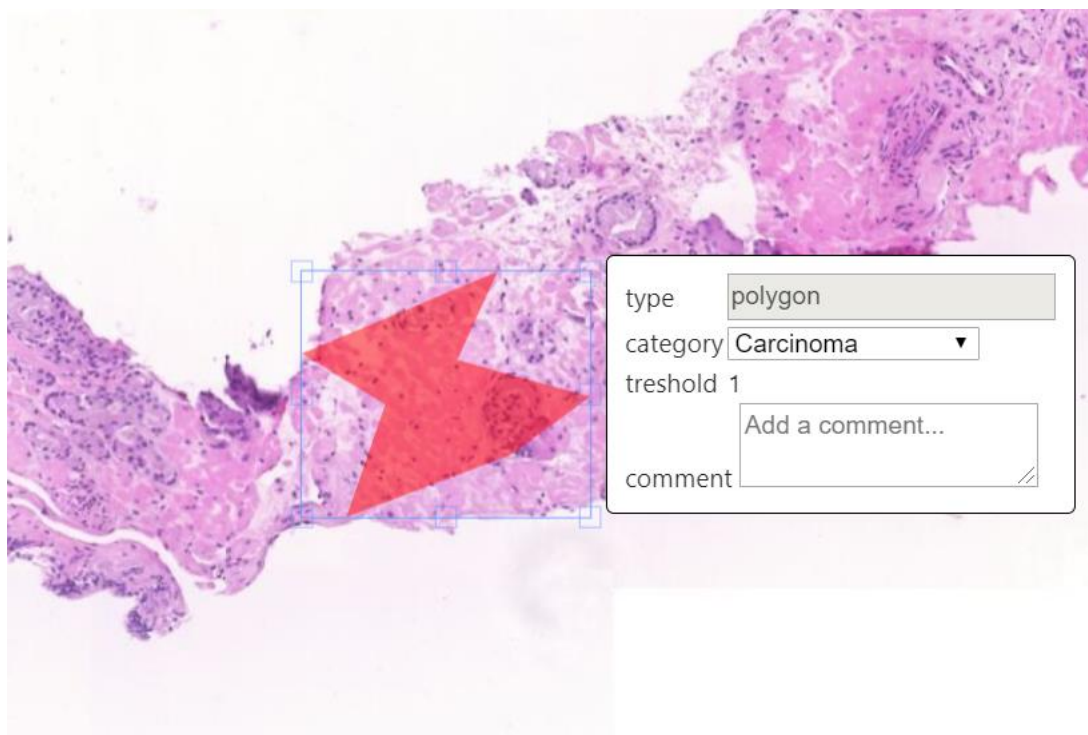


Figure 16 Annotation input form

Lastly, annotations need to be downloaded in a specific ASAP XML format compatible with the project pipeline. For this purpose, after the user clicks on the download button, XML DOM is generated, and coordinates of each annotation are recalculated and inserted into DOM with correct tags. Apart from the XML file, the system also generates a JSON file with the complete description of all canvas objects, which can be later loaded into a viewer and redrawn.

To sum up, after implemented alterations, users can create polygonal annotations, edit them, add the necessary information, and download annotations description. Additional functionality is under development.

Interactiveness

Efficient zooming and panning were already employed by OpenSeadragon; therefore, from the interactiveness perspective, only file browsing capability needs to be improved. Currently, users cannot browse through images and load them into the viewer.

The first step is to facilitate the change of input files. To achieve this, the path to images is stored in URL parameters, which are then read by the script and passed to the viewer as the tileSources argument during initialization. To load new images, respective paths can be altered in URL without accessing and modifying internal scripts.

Moreover, for improving user experience simple web page was created, which allows users (pathologists, scientists...) to access an overview of all past results. It is possible to browse through individual experiments and examine their overall metrics and results. Each experiment has its own set of images used for training, which are displayed in a table (Figure 17). Users can sort the table by the basic metrics, and the link in the first column will direct them to the OpenSeadragon viewer set with corresponding images (source image and its masks if available.)

- GLADOS-CHCK-99c3-456b-a66a-84c06777c382
 - 2020-02-11 14:10
 - October model retraining. Prospective slides used for training. Flattened 3x3 context area. Validation done across entire slides (no random sampling). Testing is done per-slide.
 - Parameters
 - Training
 - Validation
 - results for epoch_11: {'binary_accuracy': '0.88631', 'recall': '0.82096', 'precision': '0.91424', 'loss': '0.31898', 'f1': '0.86509'}

Results overview												
Image	Annotation	Empty annot	Probabs	FP 50	FN 50	FP 70	FN 70	FP 90	FN 90	FP 90	FN 90	
TP-2019_6786-02-1	1	1	1	1532	0	729	0	0	0	0	0	More info
TP-2019_2824-01-1	1	0	1	481	111	130	161	0	200	0	200	More info
TP-2019_6785-13-1	1	0	1	240	522	100	914	0	1642	0	1642	More info
TP-2019_6786-01-1	1	0	1	74	243	21	404	0	503	0	503	More info
TP-2019_2623-12-1	1	0	1	946	59	237	113	0	221	0	221	More info
TP-2019_6785-14-1	1	0	1	216	938	85	1612	0	2518	0	2518	More info
TP-2019_2941-01-1	1	0	1	465	85	120	122	0	158	0	158	More info
TP-2019_6786-03-1	1	0	1	224	775	62	1378	0	2360	0	2360	More info
TP-2019_6785-12-1	1	0	1	86	88	9	136	0	169	0	169	More info
TP-2019_7362-07-1	0	1	1	321	0	101	0	0	0	0	0	More info

[Back to overview](#)

Figure 17 Web page with results

To sum up, after performed alterations, users can easily browse through all the experiments and examine selected results in OpenSeadragon from a simple web page

Performance

As we saw in the performance summary table (Table 3), the creation of DZI files is time and space inefficient. However, there are several images serving protocols that could generate DZI files on the run from a single pyramidal TIF.

We are using IIPIMAGE with the Deepzoom protocol (Figure 18). The protocol will generate the necessary JSON description file and will provide requested JPEG tiles. Continuous zooming stays unaffected, and storage space is slightly decreased as we only need to store a single TIF file.

```
167 // set tile sources for individual images. IIPIMAGE with deepzoom protocol
168 // will provide dzi tiles from tif (urlImage, urlAnnotation ...)
169 var baseTileSource = "/iipsrv/iipsrv.fcgi?Deepzoom="+ urlImage + ".dzi";
170 var probasTileSource = "/iipsrv/iipsrv.fcgi?Deepzoom="+ urlProbabilities + ".dzi";
171 var annotTileSource = "/iipsrv/iipsrv.fcgi?Deepzoom="+ urlAnnotation + ".dzi";
172 var decsTileSource = "/iipsrv/iipsrv.fcgi?Deepzoom="+ urlDecision + ".dzi";
```

Figure 18 Implementation of Deepzoom protocol

In conclusion, the main advantage of the implementation of the Deepzoom protocol is a conversion time into the desired format, which is reduced from almost 2 hours to several minutes per slide (depending on computation power).

Summary

To sum up, after the alterations mentioned above, OpenSeadragon is ready to be fully incorporated into the project workflow. Required images and results are adequately stored and accessible to users from a simple web page. After the opening of an experiment in the viewer, the user can examine all the layers individually with desired thresholding and outline an interesting region with new annotations.

The final environment with the original tissue and two overlays can be observed in Figure 19. Yellow areas represent results from the network, the blue area depicts the initial annotation from the pathologist, and the green polygon illustrates new annotation created by the user while browsing.

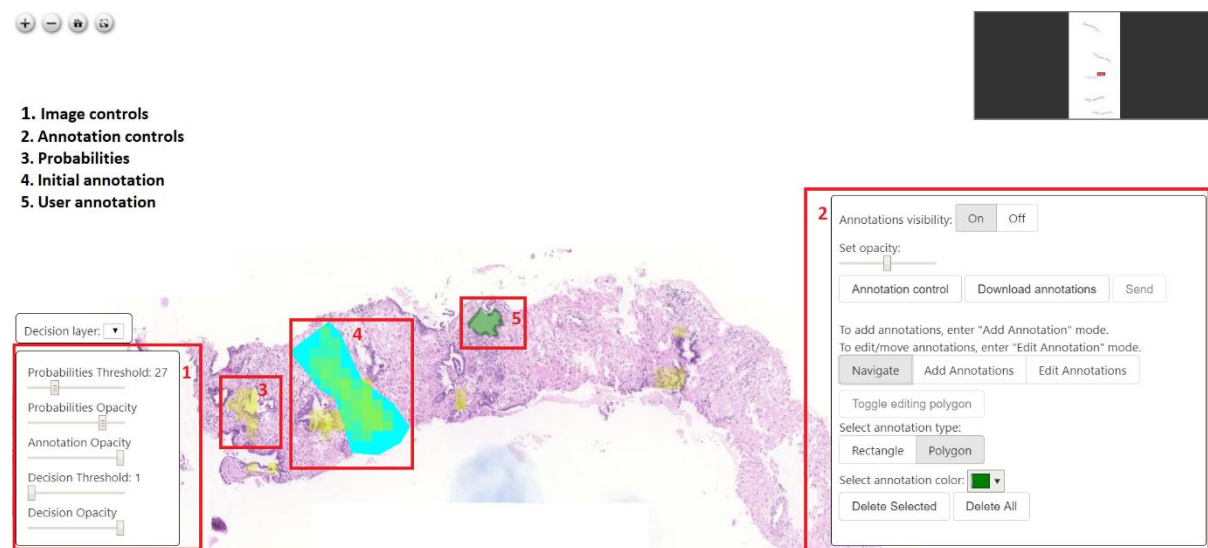


Figure 19 Final design of OpenSeadragon

Concerning future updates, any potential additional functionalities (more layers, new thresholds, watershed algorithm ...) should be feasible due to the modification capabilities of OpenSeadragon viewer.

Conclusion

To conclude, the first goal of the thesis was to compare existing tools for the visualization of large-scale digital pathology images, which was done in the Analysis chapter. The second goal was to implement the most suitable viewer for the project between Masaryk University and Masaryk Memorial Cancer Institute, which was done in the Implementation chapter, where OpenSeadragon viewer was modified.

After completed modifications, OpenSeadragon offers a requested number of layers with opacity and threshold control and annotation support. Together with a simple web page serving as the "storage" for conducted experiments system was successfully implemented into a workflow, and it is now the inseparable part of the project.

Thanks to an opensource code, sufficient documentation, and community support, OpenSeadragon viewer is also prepared for other modifications to further enhanced user experience and support future project requirements. Future requirements should include enhanced input form, automatic annotation of a region based on probability mask, collaborative annotations, interactive analysis, and connection to the database so that created annotations can be stored and reloaded.

References

- [1] Keith J. Kaplan; Luigi K.F. Rao (2015, October 23). *Digital Pathology: Historical Perspectives, Current Concepts & Future Applications*. Springer. ISBN 978-3-319-20379-9
- [2] Sucaet Y, Waelput W (2014). *Digital Pathology*. Springer. ISBN 9783319087801
- [3] Colling, R., Pitman, H., Oien, K., Rajpoot, N., Macklin, P., Snead, D., Sackville, T. and Verrill, C. (2019), Artificial intelligence in digital pathology: a roadmap to routine use in clinical practice. *J. Pathol.*, 249: 143-150. doi:10.1002/path.5310
- [4] Bera, K., Schalper, K.A., Rimm, DL et al. (2019). Artificial intelligence in digital pathology — new tools for diagnosis and precision oncology. *Nat Rev Clin Oncol* 16, 703–715 (2019). doi:10.1038/s41571-019-0252-y
- [5] Niazi, Muhammad Khalid Khan et al. (2019, May 1). Digital pathology and artificial intelligence. *The Lancet Oncology*, Volume 20, Issue 5, e253 - e261. doi: /10.1016/S1470-2045(19)30154-8
- [6] Farahani N, Parwani A, Pantanowitz L (2015, June 11). Whole slide imaging in pathology: advantages, limitations, and emerging perspectives. *Pathology and Laboratory Medicine International*, 2015(7), 23-33. doi: 10.2147/PLMI.S59826
- [7] Andelson E.H., Anderson C.H., Bergen J.R., Burt P.J., Ogden J.M. (1984) "Pyramid methods in image processing."
- [8] Sucaet Y, Pappas A, Waelput W (2013). *A brief history of digital pathology and its applications*. Retrieved from <https://www.pathomation.com/pdf/white%20paper%20history.pdf>
- [9] Otemuyiwa, P. (2017, August 5). The Great JPEG 2000 Debate: Analyzing the Pros and Cons to Widespread Adoption [Blog post]. Retrieved from: https://cloudinary.com/blog/the_great_jpeg_2000_debate_analyzing_the_pros_and_cons_to_wide_spread_adoption
- [10] TIFF format description [online] 2017, [2020-02-15]. Retrieved from: <https://www.loc.gov/preservation/digital/formats/fdd/fdd000237.shtml>
- [11] Deep Zoom File Format Overview [online] 2011, [2020-02-17]. Retrieved from: [https://docs.microsoft.com/en-us/previous-versions/windows/silverlight/dotnet-windows-silverlight/cc645077\(v=vs.95\)](https://docs.microsoft.com/en-us/previous-versions/windows/silverlight/dotnet-windows-silverlight/cc645077(v=vs.95))
- [12] Gasienica, D. (2008, October 11). Inside Deep Zoom – Part II: Mathematical Analysis [Blog post]. Retrieved from: <https://www.gasi.ch/blog/inside-deep-zoom-2>
- [13] Tuominen, V.J., Isola, J. The Application of JPEG2000 in Virtual Microscopy. *J Digit Imaging* 22, 250–258 (2009). doi: 10.1007/s10278-007-9090-z
- [14] MIRAX format [online], [2020-03-08]. Retrieved from: <https://openslide.org/formats/mirax/>

Appendix A – Installation instructions

1. Prerequisites: IIPIMAGE server

<https://iipimage.sourceforge.io/documentation/server/>

- the server will provide DZI files from existing TIF files via DEEPZOOM protocol

(<https://iipimage.sourceforge.io/documentation/protocol/>)

2. Download openseadragon_project

(https://github.com/cesekova/openseadragon_project)

3. Unzip and save demo_images folder on IIPIMAGE server

4. Run openseadragon_project/index.html

- make sure that your path to iipsrv.fcgi is the same as in line 266, or adjust it accordingly (e.g. "/fcgi-bin/iipsrv.fcgi")

5. To load images, fill out URL parameters:

.../openseadragon_project/index.html?**annotation**=demo_images/TP-2019_2824-01-1-annot.tif&**probabilities**=demo_images/TP-2019_2824-01-1-vis.tif&**decision**=demo_images/TP-2019_2824-01-1-occ.tif

- **image** attribute is not provided as the source image contains real patient tissue and cannot be published

- in case the decision layer has multiple images, the decision argument can be repeated (...&decision=path/to/decision1.tif&decision=path/to/decision2.tif&decision=path/to/decision3.tif...)

Layers in general:

image: source image with tissue, no modifications are done

probabilities: grayscale image with black background (black pixels are filtered out, gray/white pixels are recolored and filtered based on value on Probabilities Threshold slider and opacity is changed by Probabilities opacity slider)

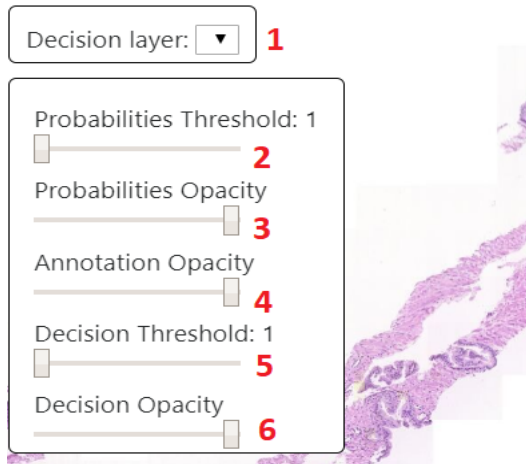
annotations: Black and white image (black pixels are filtered out; the rest is recolored. Opacity is changed by Annotation opacity¹⁶ slider)

decision: green and red image with black background (black pixels are filtered out, red/green pixels are filtered based on value on Decision Threshold slider, and opacity is changed by Decision opacity slider)

¹⁶ In this demo example, opacity for the first ("0"th) layer cannot be changed because the source image is missing

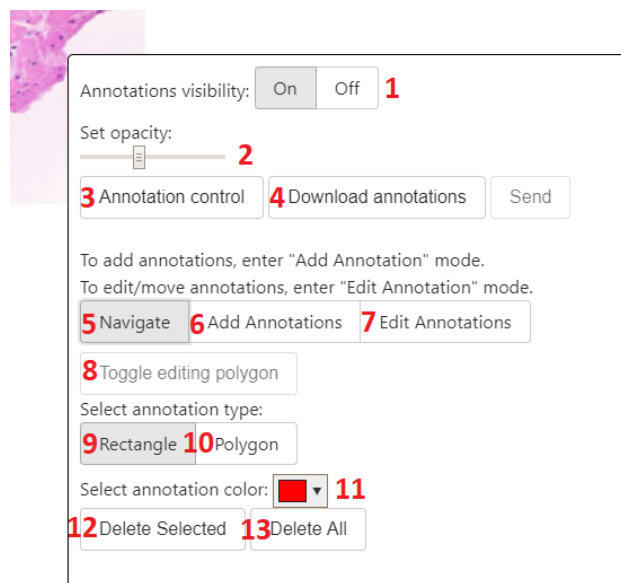
Appendix B – User manual

Image controls:



1. Change the image for the decision layer (if available)
2. Change the threshold for the probability layer
3. Change the opacity of the probability layer
4. Change the opacity for the annotation layer
5. Change the threshold for the decision layer
6. Change the opacity for the decision layer

Annotation controls:



1. Toggle annotations visibility
2. Set opacity for all user-created annotations
3. Show/hide annotation controls
4. Download JSON and XML file with annotation descriptions
5. OSD navigation (zoom and pan) mode – no interaction with annotations
6. In this mode, annotations can be created (not selected). After annotation is created, the input form will appear, and mode will switch to edit
7. In edit mode, annotations can be edited by selecting them (only basic editing – size,

position, and input form)

8. When in editing mode, after polygon was selected, the Toggle editing polygon button enables to change the shape of the polygon (each point of the polygon can be dragged to a different position). Edit is finished by re-clicking the button
9. Create a rectangle by click and drag (Add annotation mode must be active - 6)
10. Create polygon by placing points (polygon drawing is finished when the starting point is selected)
11. Set color for future annotations/change color for the selected annotation
12. Delete selected annotation (might be used for canceling polygon drawing)
13. Delete all user-created annotations