Gigapixel whole slide image analysis with deep learning

Zsolt Bedőházi[1,3] , András Biricz[2,3]

[1] ELTE Eötvös Loránd University, Doctoral School of Informatics, Budapest, Hungary
[2] ELTE Eötvös Loránd University, Department of Complex Systems in Physics, Budapest, Hungary
[3] Semmelweis University, Health Services Management Training Centre, Faculty of Health and Public Administration, Budapest, Hungary

Dr. István Csabai

Zsolt Bedőházi

András Biricz

Dr. Péter Pollner

ELTE EÖTVÖS LORÁND UNIVERSITY

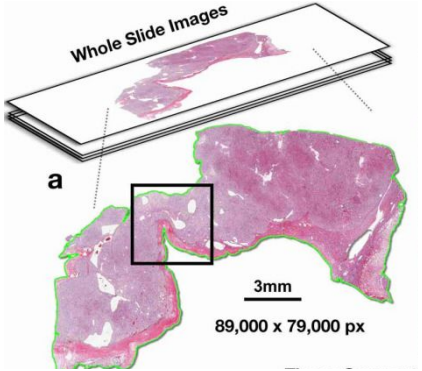
Csabai bio

http://csabai.web.elte.hu/

Semmelweis Egyetem EMK Egészségügyi Menedzserképző Központ

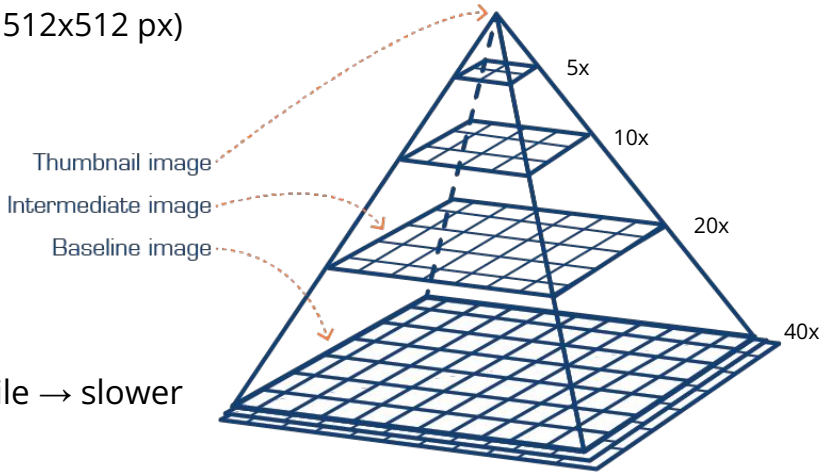
Universitas Budapestinensis de Semmelweis Nominata 1769

What is Whole Slide Imaging (WSI)



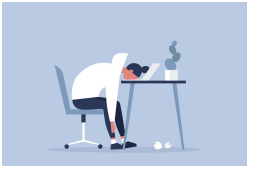
https://pixelscientia.com/articles/from-patches-to-slides/

- refers to scanning of conventional glass slides to produce digital images
- **WSIs are very large:**
  - 30 x 20 mm at 40x (0.25 um/px) ~10 gigapixels, ~30 GB uncompressed
  - **Tiling**
    - image divided into grid of rectangular tiles (e.g. 512x512 px)
    - index of tile to file offset
    - only read tiles overlapping with ROI
    - fast random access, fast panning
  - **Pyramid levels**
    - each downsampled by 0.5x
    - zooming in → larger ROI → more pixels and tile → slower



https://medical.sectra.com/resources/introduction-dicom-standard-digital-pathology/

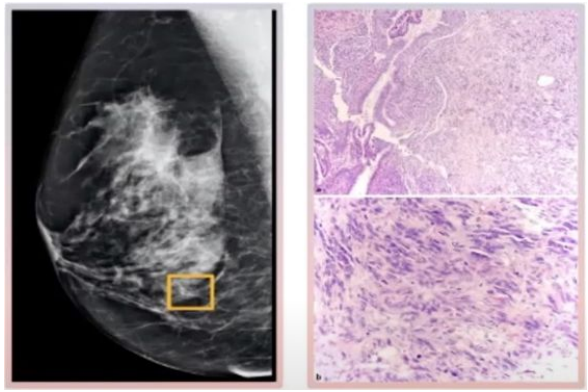
WSI preprocessing and difficulties



- huge dataset size (TB scale)
- disk read and write speed is essential
- deep learning models cannot take raw slides as input
- annotation of regions/patches is time consuming, leads to losing global context
- **Workarounds:**
    - multi-instance learning: learn with global slide label and patches
    - learning with smaller representations (embeddings)

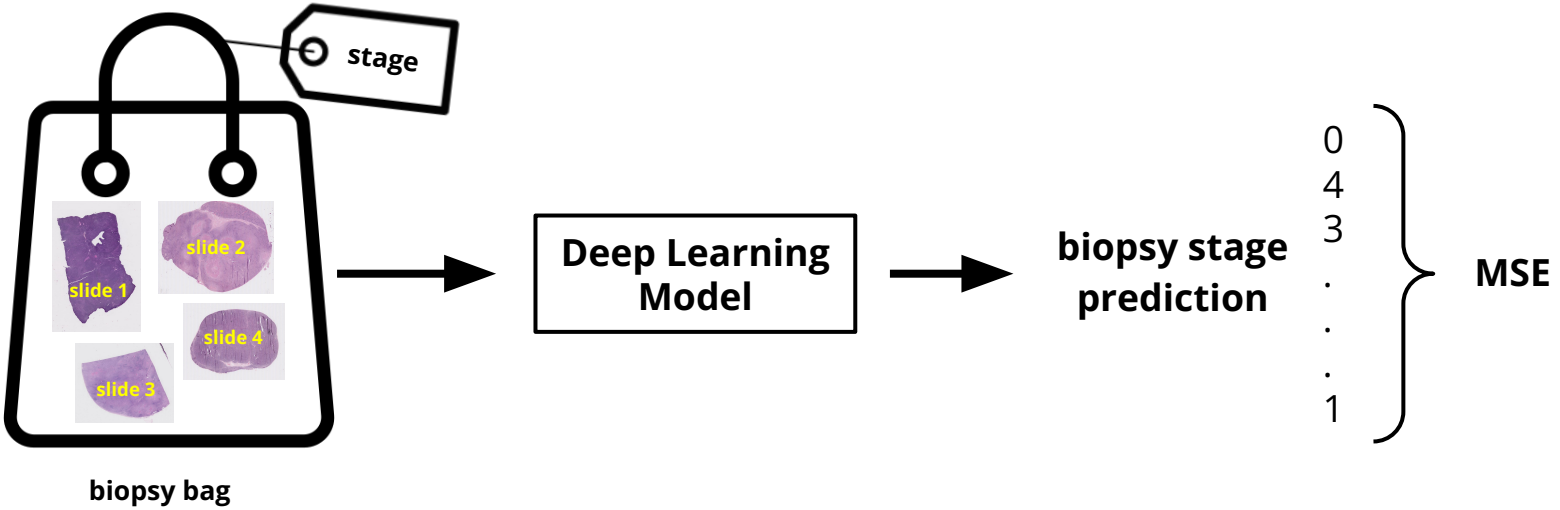
Breast cancer stage prediction using gigapixel pathology images (A Nightingale Open Science Challenge)

- **Goal:**
  - predict the stage of a patient's cancer using only the slide images generated by breast biopsy
- **Data:**
  - 4.335 breast biopsies, **72.400 slides**, 4.200 cases
  - slide resolution around 100.000 x 150.000 pixels
  - average slide size ~2 GB
  - total data ~ **130 TB**
  - **difficulty:** work possible only on a cloud platform



https://www.nightingalescience.org/news/winners-of-the-high-risk-breast-cancer-prediction-contest-1

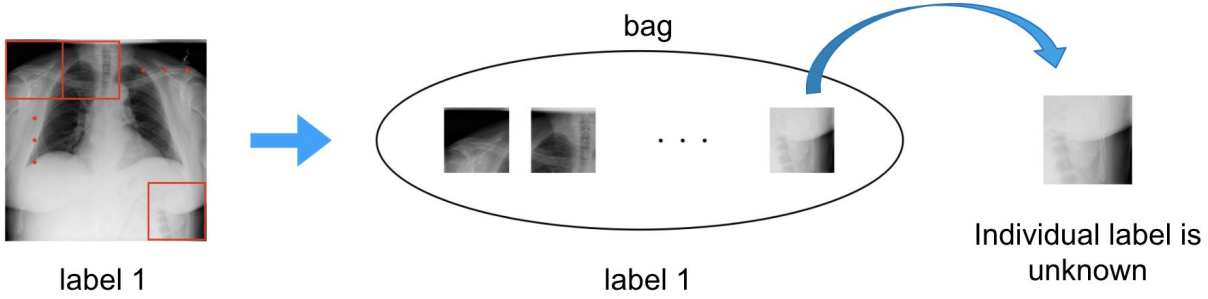
Problem to solve

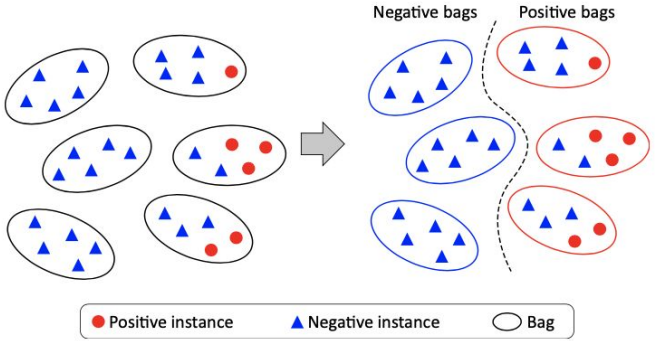


- A biopsy bag has many slides (between 1-100)
- Only biopsy label is given (stage: 0,1,2,3,4), slide labels are not available

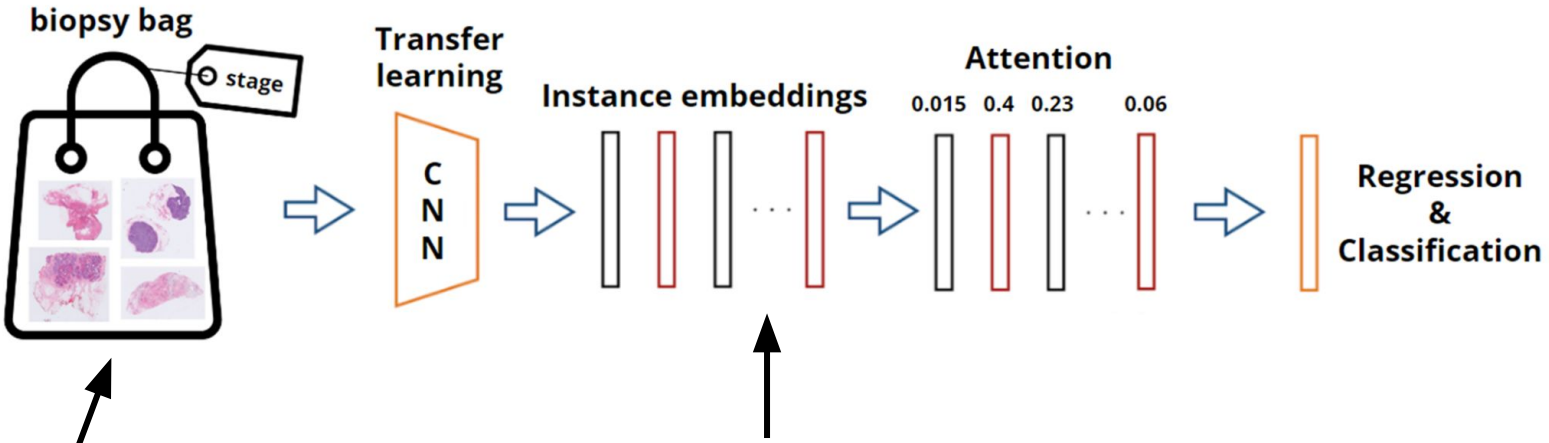
Multi Instance Learning

- **Motivation**: only global label is known



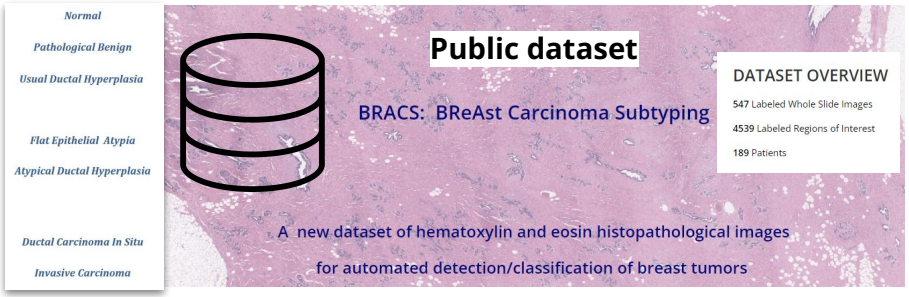


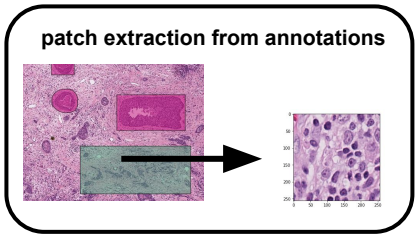
Our workflow - Deep multi instance learning and attention



wsingredients

- data preprocessing
- masking
- patch extraction
- bag generation





Contest Phase 1 - 1st place

NIGHTINGALE
OPEN SCIENCE

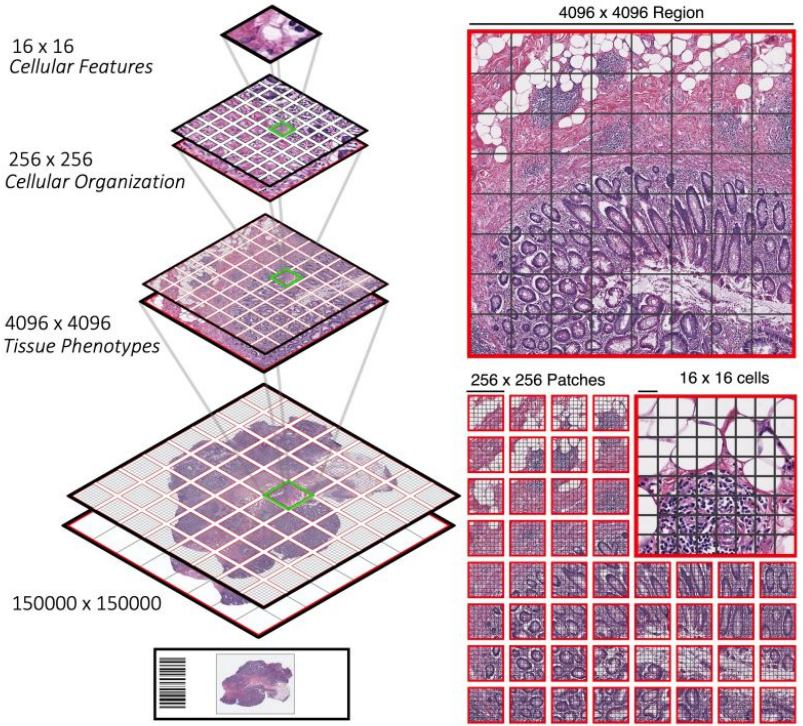
Contests / Predicting High Risk Breast Cancer - Phase 1 (2022) (3jmp2y128nxd)

Predicting High Risk Breast Cancer - Phase 1 (2022)

Ranking	Team Name	Score	Description
1	csabAIbio	0.5222186	csabAIbio ensemble comb1
2	Bonaventure Dossou	0.5543481	Deep Ensembles of 8 models with mixed learning rates, batch_size: 32, n_epochs: 50, and AdamW optimizer
3	PKU-Edinburgh	0.5895417	Resnet50 & SwinLarge Ensemble
4	Equitech Research Labs	0.6353726	ResNet34 with new data, using MLP regression model with 0.612 MSE
5	tp_jh_hw_brca_nov_2022	0.7399518	tp_jh_brca_nov_2022: fine tuned EfficientNet
6	Nightingale benchmark - ResNet18 with entire slides downsampled to 224x224	0.7426637	resnet18 basic
7	Nightingale benchmark - Predicting stage one for all holdout biopsies	0.7878104	A submission of all stage one
8	Breast cancer	0.7878104	This is sample submission
9	Breast Cancer Project	0.9364919	V0.1

https://github.com/csabaiBio/nightingale_breast_contest_phase1

Unique Properties of WSIs



https://arxiv.org/pdf/2206.02647.pdf

- visual concepts are objective, at a given magnification the image scale is fixed (20x - 0.5mpp)
- WSIs exhibit a hierarchical structure of visual tokens across varying resolutions: each part is semantic and fits into a larger object
  - 16x16 images: captures individual cells, cell activity
  - 256x256 images: cell to cell interactions
  - 4096x4096 images: interaction within the tissue microenvironment, spatial organization of cells
  - WSI: overall tissue microenvironment

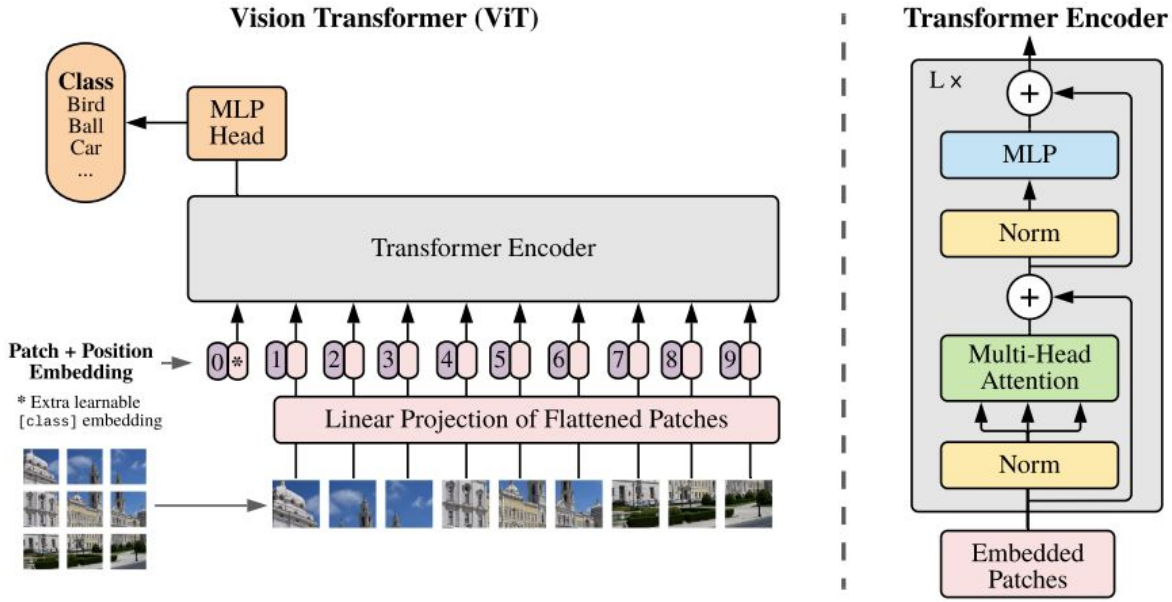
Vision Transformer

- very similar architecture and mechanism to the original Transformer
- input sequence is sequence of image patches with position embeddings
- current state-of-the-art for wide range of tasks

AN IMAGE IS WORTH 16X16 WORDS:
TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

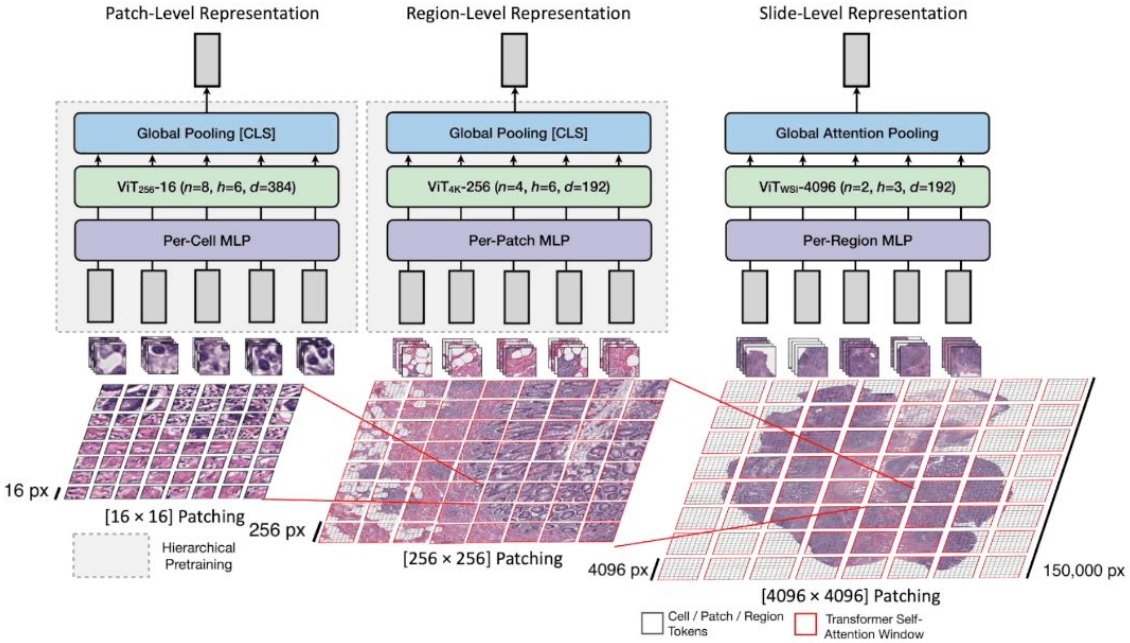
Alexey Dosovitskiy*,†, Lucas Beyer*, Alexander Kolesnikov*, Dirk Weissenborn*,
Xiaohua Zhai*, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer,
Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby*,†
*equal technical contribution, †equal advising
Google Research, Brain Team
{adosovitskiy, neilhoulsby}@google.com

https://arxiv.org/pdf/2010.11929.pdf



Scaling Vision Transformers to Gigapixel Images via Hierarchical Self-Supervised Learning

Richard J. Chen[1], Chengkuan Chen[1], Yicong Li[1], Tiffany Y. Chen[1],
Andrew D. Trister[2], Rahul G. Krishnan[3,*], Faisal Mahmood[1,*]
[1]Harvard, BWH, Broad Institute [2]Bill & Melinda Gates Foundation [3]University of Toronto
richardjchen@g.harvard.edu, faisalmahmood@bwh.harvard.edu



https://arxiv.org/pdf/2206.02647.pdf

Hierarchical Self-Supervised Learning

Benefits:

- unsupervised training
- compression of data -> smaller representations (embeddings)
    - e.g: slide with 50.000 patches (256x256x3), uint8 ~ 7.86 GB → ~ 150x192 embeddings float16 ~ 57.6 kB
    - e.g: 100 TB -> ~ 1 GB
- fitting into the RAM, increase of batch size
- handy for downstream tasks and experiments

Contest Phase 2 - 1st place

NIGHTINGALE
OPEN SCIENCE

Contests / Predicting High Risk Breast Cancer - Phase 2 (2023) (vd8g98zv9w0p)

Predicting High Risk Breast Cancer - Phase 2 (2023)

Ranking	Team Name	Score	Description
1	csabAIbio	0.7606058	Seventh submission
2	Bonaventure Dossou	0.7305834	preds_all_geo_times_arith_mean
3	Nightingale baseline - CLAM implementation	0.6832263	This submission utilized CLAM -- https://github.com/mahmoodlab/CLAM
4	Central Parquet	0.6679352	final
5	ML4HC@NYUAD	0.6337327	CLAM k5 p20 ensembling submission
6	Predicting High Risk Breast Cancer	0.6037197	Learners
7	Purplestein's Monsters	0.6035635	First test manual ensemble model
8	Equitech Research Labs	0.5872650	Exp 6, split 1, KimiaNet + CLAM
9	PKU-Edinburgh	0.5747690	ResNet-50 with larger train/val set. Err: Too many class 1
10	CrazyThursdayVme50	0.5163983	This is naive solution submission

https://github.com/csabaiBio/nightingale_breast_contest_phase2

Computational challenges - how we achieved these results

- Get to know the hardware - first very important step!
  - Is it a standalone server or cloud? If cloud, is it free/paid?
  - Amount of available RAM/CPU/GPU?
  - Calculate/measure theoretical limits of data processing (disk speed, network speed, etc.)

- Get to know the restrictions of data sharing policy (medical images).

- Dive into the packages, insert/add codes/implementations.
  - Data loaders
  - Models

Computational challenges - how we achieved these results

```python
class CLAM_h5_Dataset_memory(Dataset):
    def __init__(self, parent_folder, transform=None):
        self.parent_folder = parent_folder
        self.transform = transform
        slide_fp = os.path.join(self.parent_folder, f'*.h5')
        self.files = np.array( sorted( glob.glob(slide_fp) ) )
        self.dataset_name = 'imgs'
        self.num_images = self.get_num_images()
        print('NUM IMAGES to load:', self.num_images )
        self.data = self.load_data()

    def get_num_images(self):
        # Get the total number of images
        num_images = 0
        for input_file in self.files:
            with h5py.File(input_file, 'r') as in_h5:
                num_images += len(in_h5[self.dataset_name])
        return num_images

    def load_data(self):
        # Define the shape of the final array
        shape = (self.num_images, 256, 256, 3)

        # Create an empty numpy array to hold the images
        data = np.zeros(shape, dtype=np.uint8)
        data.fill(0)

        curr_idx = 0 # Copy the data from each file to its required place in the final array
        for input_file in tqdm(self.files):
            #print('loaded file:', input_file)
            with h5py.File(input_file, 'r') as in_h5:
                num_images = len(in_h5[self.dataset_name])
                data[curr_idx:curr_idx+num_images] = in_h5[self.dataset_name]
                curr_idx += num_images

        return data

    def __len__(self):
        return self.num_images

    def __getitem__(self, idx):
        image_data = self.data[idx]        # Get the image data from the final array

        if self.transform:
            image_data = self.transform(image_data)

        return image_data # RETURNS PIL IMAGE!
```

```python
class CLAM_h5_Dataset_disk(Dataset):
    def __init__(self, parent_folder, transform=None):
        self.parent_folder = parent_folder
        self.transform = transform
        slide_fp = os.path.join(self.parent_folder, f'*.h5')
        self.files = np.array( sorted( glob.glob(slide_fp) ) )
        self.dataset_name = None
        self.file_index_map = self.get_file_idx_map()
        print('Files initialized!')


    def get_file_idx_map(self):
        # Define the paths to the HDF5 files and the dataset name
        input_files = self.files
        self.dataset_name = 'imgs'

        # Open the HDF5 files in "r"ead mode and build an index to file map
        curr_idx = 0
        file_index_map = {}
        for input_file in tqdm(input_files):
            with h5py.File(input_file, 'r') as in_h5:
                num_images = len(in_h5[self.dataset_name])
                file_index_map.update({(i+curr_idx): input_file for i in range(num_images)})
                curr_idx += num_images

        return file_index_map

    def __len__(self):
        return len(self.file_index_map)

    def __getitem__(self, idx):
        input_file = self.file_index_map[idx]

        # Open the file and get the image data
        with h5py.File(input_file, 'r') as in_h5:
            # Get the local index of the image in the file
            local_idx = idx - next(k for k, v in self.file_index_map.items() if v == input_file)

            # Get the image data from the file
            image_data = in_h5[self.dataset_name][local_idx]

            if self.transform:
                image_data = self.transform(image_data)

        return image_data # RETURNS PIL IMAGE!
```

Thank You!

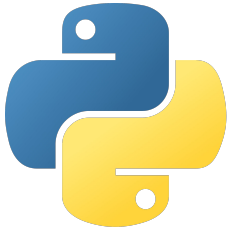
Acknowledgement:

EU project RRF-2.3.1-21-2022-0004 within the framework of the Artificial Intelligence National Laboratory and RFR-2.3.1-21-2022-00006, Datadriven Health Division of Health Security NL

We acknowledge the computational resources provided by the Wigner Scientific Computing Laboratory (WSCLAB), which were crucial in the successful completion of our work.

WSI preprocessing and difficulties - A solution

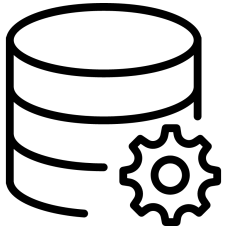
- an in-house developed framework that supports various WSI preprocessing steps that are needed to allow artificial intelligence-based analysis



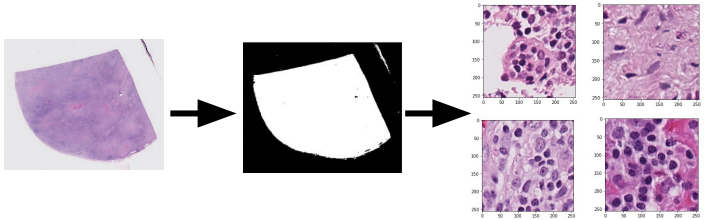
wsingredients

A toolbox to simplify tasks with Whole Slide Images

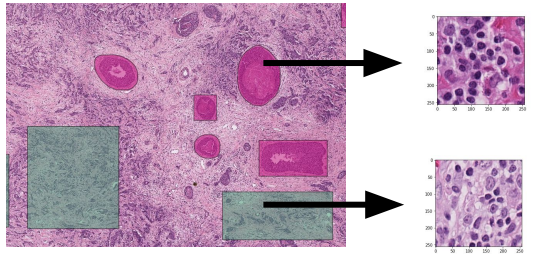
simplified, optimized slide or patch reading



ROI segmentation and patch extraction

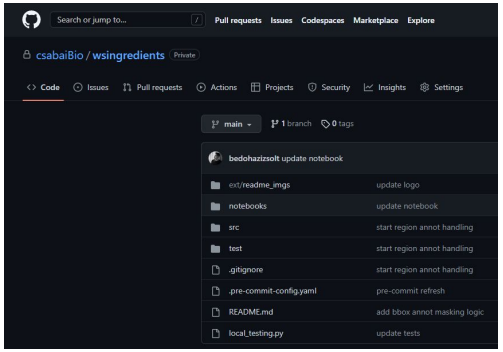


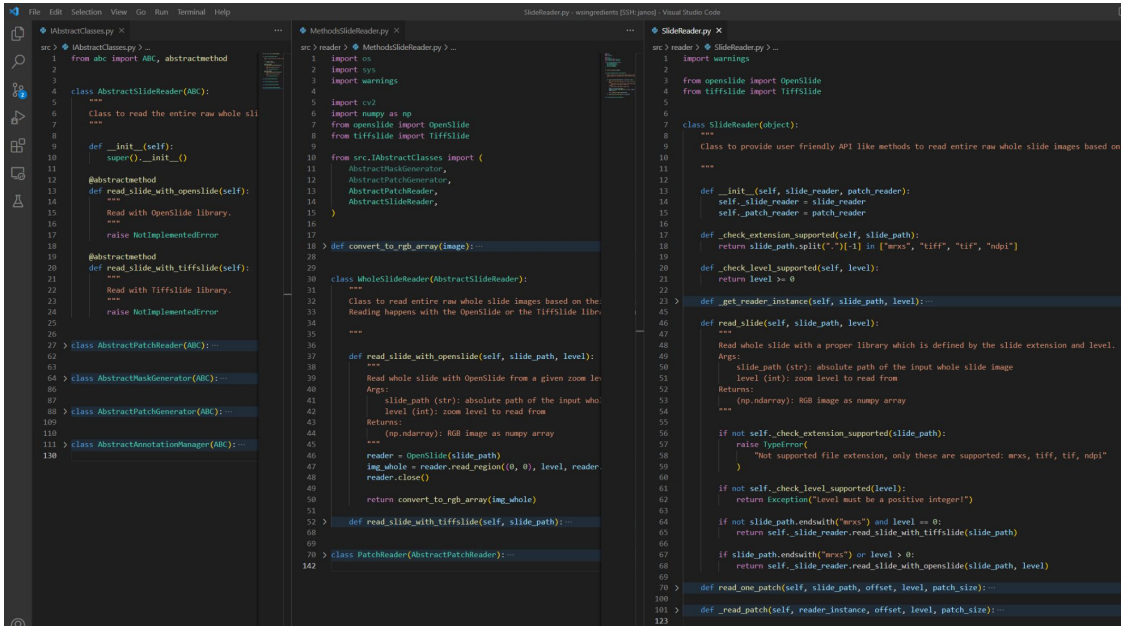
patch extraction from annotations



wsingredients

- developed with focus on general usability, modularity, scalability, maintainability
- use of design patterns, software design principles
- user-friendly API-like commands
- easily repurposable not just for other tissue types but in general for any digital microscopy problem
- will be open-sourced and released to the Python PyPi index as installable package





Example of usage:

easy installation via pip, one-line to import

instantiation of desired objects (e.g. reader, patcher, annotator etc.) needed only once, can be used multiple times

based on the file extension and reading level, the reader objects automatically determines the optimal reading method for maximum speed

Import the package

```
import wsingredients
```

Instanciate a reader or patcher object

```
my_reader = SlideReaderFactory().SlideReaderFactoryInstance()
```

```
my_patcher = PatchGeneratorFactory().PatchGeneratorFactoryInstance()
```

Use the user-friendly methods

```
slide_path = '/local_dir/Sample.mrxs'
```

```
slide = my_reader.read_slide(slide_path, level=3)
```

```
patch = my_reader.read_one_patch(slide_path, offset=(0, 0), level=0, patch_size=(128, 128))
```