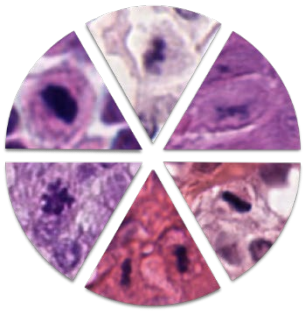
Programming Exercise Digital Pathology and Deep Learning (WiSe 22/23)

In this programming exercise, we will look into mitosis detection – following the task from the MIDOG 2021 challenge [1]: The goal is to train (and test) an object detection model that identifies mitotic figures, i.e., cells undergoing cell division, on images from different scanners. We will utilize only the MIDOG training set [2] and use the train / test split from the baseline approach. For this task, we will start with the RetinaNet [3], which uses a feature pyramid network and the focal loss.



MIDOG 2021

Mitosis Domain Generalization

MICCAI2021

In the following, you will find multiple tasks with the goal to train and implement such an approach. The tasks may contain additional *optional* subtasks which we have added in case that you would like to look deeper into the corresponding topics. You can refactor the code such that it suits your needs.

We expect that you have a basic understanding of pytorch (link to tutorial), as this framework is also used in the Lecture "Deep Learning". We further use the pytorch lightning (link to tutorial) library to simplify training and logging.

Document your experiments and the answers to the questions raised in the tasks below in a experiment report. This report should contain brief answers and summarize the most important concepts that you implemented, illustrate training/test results etc. If your seminar (talk) topic connects to the experiments we perform here, add a short paragraph discussing this connection.

Important notes:

0) We have done our best to test the code, the exercise task, and the cluster beforehand; however, there may have been issues that we have overlooked. Please adapt the code to your needs and please don't hesitate to contact us if you believe that there is a problem with the setup. In case you have questions w.r.t. to the tasks after you have spent a reasonable amount of time trying to solve them, we will provide open office hours for additional guidance.

1) Have a look at the cluster tutorial in the github repository for this exercise: https://gitlab.cs.fau.de/aimi-lab-fau/cluster-instructions-dpdl

   This git repository also contains the code skeleton in mitosis_detection as well as this task description (well, you downloaded it from there ;)

2) We provide the data for this exercise on the cluster accessible for everyone in the following folder to avoid issues with the data setup[1]:

   `/home/janus/iwb6-datasets/MIDOG2021`

3) You don't need to install the framework/python environment yourself on the cluster, but we instead highly recommend using the following virtual conda environment directly[2]:

   `/home/woody/iwb0/iwb0002h/software/privat/conda/envs/seminar_dpdlv2`

[1] If you train your model for a longer time, it makes sense to copy the data to the local cluster compute node. have a look at the batch_script.sh file – there is code for that that can be used for this purpose.

[2] The fact that there is a v2-version may point toward some challenges we have had with the cluster setup ;)

The github repository describes how you can add this to your conda configuration such that you can easily use it with

```
conda activate seminar_dpdlv2
```

Task 1: Getting familiar with the code

Make yourself familiar with the dataset and the provided framework:

1.1. Check out the description of the MIDOG challenge dataset description and make yourself familiar with the characteristics of this dataset: https://zenodo.org/record/4643381

You can have a look at the data using the jupyter notebook provided with the repository. What are the essential characteristics of the data and the labels? (see RRZE cluster description for juypter notebooks)[3]

1.2. Have a look at the file training.py and make yourself familiar with the parameters. Similarly, have a look at `dpdl_defaults.py` and check out how the training and test set are being defined. Note that there is something wrong with how the test and training set are being used.

What is the problem in this case and why is it a problem? Correct the problem to be able to train the networks with the dataset at hand. Why is the training set provided like this by the MIDOG challenge?

1.3. Make yourself familiar with `network.py` and the RetinaNet pytorch implementation that we use: What are the important parameters for this network and why are they set this way? What would be further interesting parameters to adapt?

Task 2: Data sampling during training

Have a look at the file `midog_dataset.py`: This file provides the implementation for the dataset classes (MIDOGTrainDataset and MIDOGTestDataset). In this task, we are going to look at sampling during training.

2.1. Make yourself familiar with how a data sample is retrieved from the MIDOGTrainDataset method (via the `__getitem__` method). The images (regions of interest – RoIs) in the MIDOG dataset are fairly large (7000x5000 px), so we typically want to work with smaller-sized image patches for training and inference. Therefore, one essential aspect here is the acquisition of coordinates which are used to then get a sample patch of the desired size. At the moment, we sample random patches from the ROIs without regarding the content of the image. This can lead to some issues: Mitotic figures are rare events, so the network sees too few positive examples during training. It has been shown in prior work that oversampling mitotic figures can be helpful.

Therefore, we want to implement a sampling function my_improved_sampling_func that samples

- with a probability of 0.5 a random patch from a WSI and
- with a probability of 0.5 a patch that centers around a (random) mitotic figure in this RoI.

If the RoI does not contain any mitotic figures, the function should always sample a random patch.

[3] If you setup the conda environment as described above, you can select the kernel seminar_dpdlv2 for your jupyter notebook.

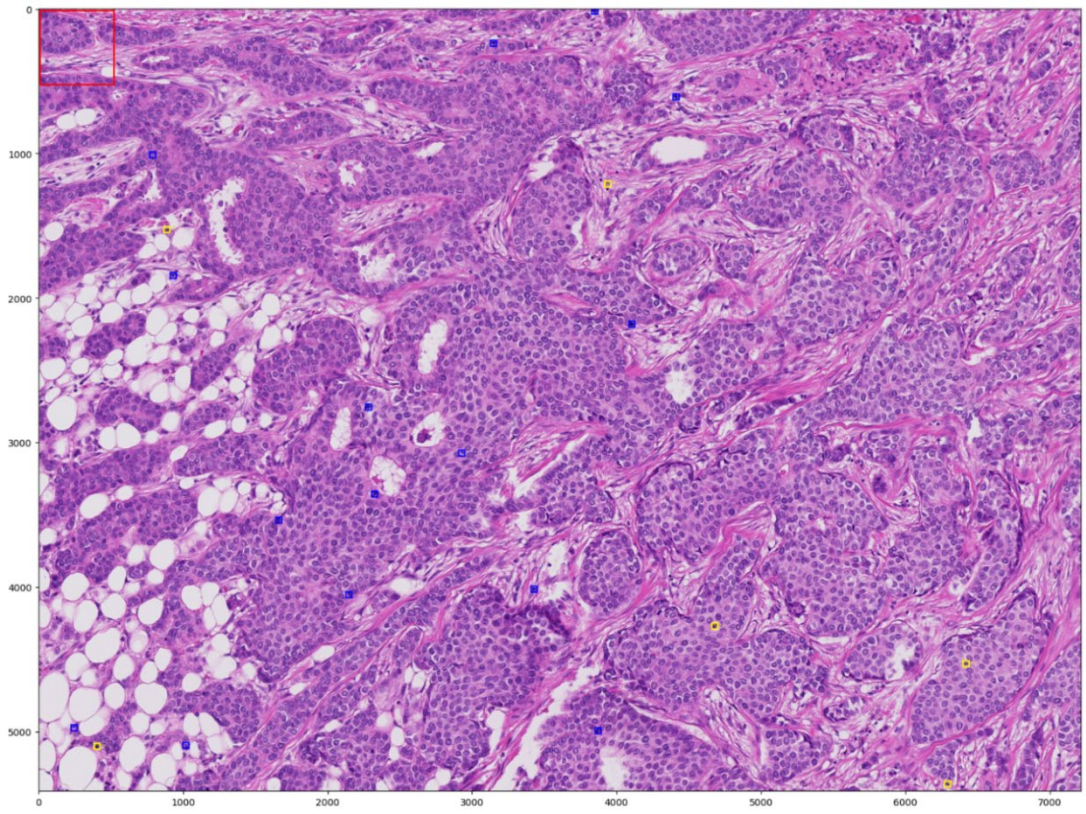


Fig. 1: This figure (faintly) demonstrates a single RoI and the size of a 512x512 patch. The blue and yellow dots represent the object labels from the ground truth.

Your sampling function receives the following parameters:

- `targets: list`
  A list that contains at [0] a list of object bounding boxes, and at [1] a list of classes of these object bounding boxes
- `classes: list`
  List of classes present in the dataset, without background (e.g., [1, 2])
- `shape: tuple`
  Tuple describing the width and height of the patch to be extracted
- `level_dimensions: list`
  List that contains the pixel dimensions per resolution level
- `level: int`
  Resolution level of interest (typically 0)

This function can be used as an argument for the constructor of `MIDOGTrainDataset`. You can find an example for this and how it is used in training.py (see `my_sample_func`) and the creation of the training/val dataset.

Note: For the validation set, we typically don't want to add this kind of sampling bias as it might provide us with overconfident results.

2.2. (optional) With the sampling you have just implemented, you only considered mitotic figures, denoted by the class label 1. However, the dataset provides additional object annotations (which for now, you did not want to include as a separate class). Think about why these labels are not useful as a separate class and why they are still provided by the challenge organizers.

If you are interested in exploring this further: Extend your sampling function such that is possible to leverage these samples. Note: This (likely) requires additional refactoring in the Dataset class(es).

Task 3: Implement inferences for full RoIs

We can now sample randomly from the RoI; however, for inference, we would like to process a whole RoI and report the results jointly. For this, we will implement the Dataset class `MIDOGTestDataset`. Here, we will generate overlapping patches from the WSI in a deterministic manner, which are then provided via the `__getitem__` method. Make yourself familiar with the existing code in the `MIDOGTestDataset` class and the corresponding code in training.py. Then implement the "TODOs" in `MIDOGTestDataset`. Implement an option for different patch overlaps during inference.

Hints: 1) Make sure to look at the implementation of `MIDOGTrainDataset` again; 2) What would be a good way to uniquely associate patches with a continuous index?

Task 4: Training and tuning

Since we have now prepared everything for training and evaluation, let's see how well our framework actually performs for mitosis detection.

4.1 Train the network architecture for mitosis detection and compare different hyper parameters (learning rate, number of epochs, number of patches/slide, etc.). Export the results in a suitable format and report final results on the test set.

Note: We export Tensorboard plots (see general Tensorboard instructions here and for how to use it on the cluster see here). Have a look at the logging functionality used in `training.py` and `network.py`.

4.2 (optional) We have now considered a RetinaNet; how does the performance of this architecture compare to for example a MaskRCNN or Yolo Architecture?

Task 5 (optional): Domain shifts

The MIDOG Dataset contains data from four different scanners. Experiment with how leaving certain scanners from the training set and different distributions of specific scanners in the training/test set impact the results of the evaluation. Discuss your experiments and results in your report.

References:

[1] Aubreville et al.: MItosis DOmain Generalization Challenge. Challenge and Workshop hosted in conjunction with MICCAI 2021. https://zenodo.org/record/4573978

[2] Aubreville et al.: MItosis DOmain Generalization Challenge (MICCAI-MIDOG 2021) Training Data. https://zenodo.org/record/4643381

[3] Lin et al.: Focal Loss for Dense Object Detection. Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), 22-29 October 2017, Venice Italy. https://arxiv.org/abs/1708.02002