Software Tools for Handling Magnetically Collected Ultra-thin Sections for Microscopy

Student:
Jelena Banjac
Data Science, EPFL, Switzerland

Professor:
Martin Jaggi
Machine Learning and Optimization lab (MLO)

Supervisor:
Thomas Templier
Center for Interdisciplinary Electron Microscopy (CIME)

June 2019

Contents

Chapter 1

Abstract

This report presents the software tools that will be available online to help users to accurately segment ultra-thin sections of brain tissue in a large image to determine section's coordinates. In order to predict these coordinates, the goal was to use state-of-art object instance segmentation framework called Masked Region-based Convolutional Neural Network (Masked R-CNN) on the dataset containing sections of brain tissue in the light microscopy images. The predicted coordinates of the sections will later be used for automated image acquisition in high resolution electron microscope. We will demonstrate that Masked R-CNN can be used to perform highly effective and efficient automatic segmentation of microscopy images containing the sections. The machine learning pipeline used will be explained in detail. In addition, we will show the results of how different parameters and settings of this pipeline were changing the performance. This semester project was done in collaboration with the Center for Interdisciplinary Electron Microscopy (CIME) lab and Machine Learning and Optimization (MLO) lab at EPFL.

Keywords: *Mask R-CNN, ML Pipeline, brain tissue, section segmentation, electron microscopy*

Chapter 2

Introduction

There is a field of neuroscience that aims to discover how neurons in the brain are wired together. In order to do that, electron microscopes are used to observe a small volume of brain tissue. To deal with this challenge, Thomas Templier, the scientist from CIME lab, uses the techniques to make thousands of ultra-thin cuts of tissue blocks that are united with the block of resin containing superparamagnetic nanoparticles. These fine cuts of the brain are collected on a surface called silicon wafer, see Fig. 2.1.

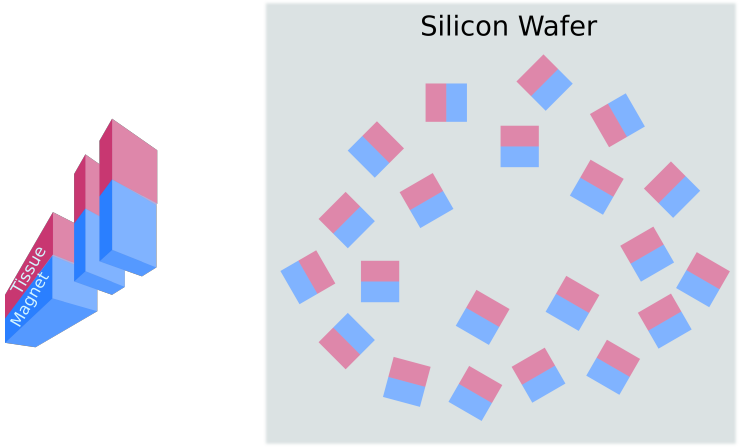


Figure 2.1: Slices of block containing brain tissue and magnet displayed on a silicon wafer

Accurate detection and segmentation of brain sections that can be observed with light microscopy play an important role for further brain section analysis. Using electron microscope imagery, a very high-resolution image of each of the sections will be acquired. In order to do that, the electron microscope needs to know the exact coordinates of the brain section. The objective of this project

was therefore to segment and detect each of these sections precisely and provide the tools that will be easy to use for users in the field. This project is a natural continuation of the project done during the Machine Learning course [4]. It implements some of the previous suggestions for improvements. This time, it is using Mask-RCNN framework developed by Facebook [4]. It was mostly concentrated to find the model that has the highest and correct predictions score. In the end, it makes tools that are more easy to use and are accessible from a user side.

In Chapter 3 we explore input data and define classes that will be used in the machine learning pipeline. Also, we mention potential problems when detecting these classes.

In Chapter 4 we explain the machine learning pipeline in detail. We mention how the software tools are used. We also quickly mention the hardware and software details and specifications.

In Chapter 5 we discuss in detail the outcome of some of the important pipeline settings, providing the results and short explanation.

In Chapter 6 we mention some of the main conclusions of the project as well as fields of improvement.

In Chapter 7 we do a quick summary of the project development process.

Chapter 3

Data Exploration

For each silicon wafer, user provides two images and the initial template. The first image is the original image of the silicon wafer containing hundreds or thousands of sections with thickness around 50 nm. Each section consists of a part containing the brain tissue and another part which contains magnetic material. See Fig. 3.1.

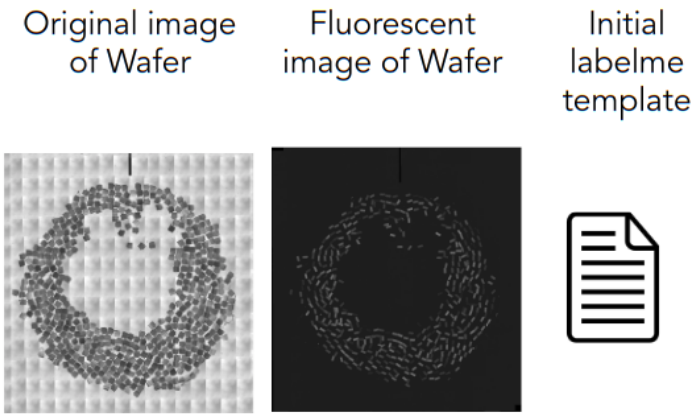


Figure 3.1: Input to the machine learning pipeline

In the magnetic material, there is a cloud of particles that is used to recognize and order each section by the order it was cut. Due to magnetic properties and present magnetic field during the section collection, the sections on the silicon wafer never overlap. Therefore, the second image is the fluorescent image of the silicon wafer containing information that we can use to locate the magnetic part of the section. See Fig. 3.1.

The initial template is a labelme [12] JSON file that usually contains five manually labeled sections and one background label (background), see Fig. 3.1.

Each section is labeled with three classes, tissue (b1), magnet (m1), and envelope (e1), see Fig. 3.2. The envelope is a label that includes the whole section as well as the small dummy part. This dummy part is not relevant to extract. It is there to help during the slicing of the sections, but also during the artificial patch generator that will be explained in Section 4.1.2.

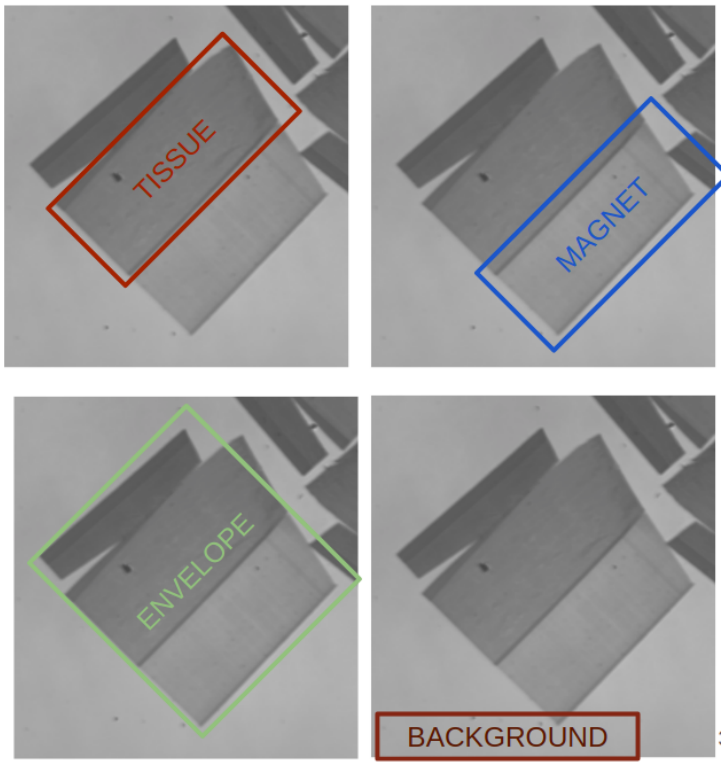


Figure 3.2: Labeling classes

3.1 Datasets

The data used for this project were images of four different silicon wafers. The Table 3.1 shows the total number of sections on each wafer:

Wafer Name	Number of Sections	Mean area tissue	Mean area magnet	Mean distance tissue -magnet	Min distance tissue	Min distance magnet	Overlap	Distance inside contour
Wafer 17	496	2065	2108	31	35	55	122	76
Wafer 16	433	2032	2124	31	33	42	121	76
Wafer 1	514	6740	5548	52	125	114	218	142
Wafer 2	503	4443	4329	45	102	104	185	113

Table 3.1: Table of wafer names and the number of sections it contains

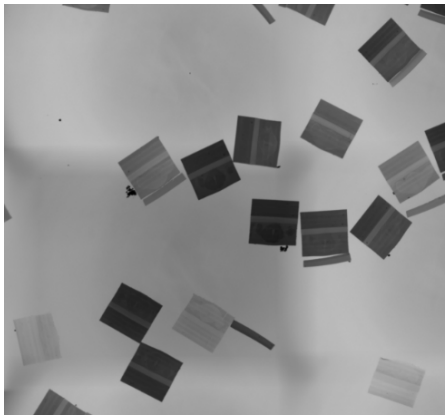
These silicon wafer images have some properties that they share. First, the sections on one silicon wafer are very similar (e.g. similar section area size, distance between tissue centroid and magnet centroid, etc.). Second, sections are placed randomly onto the silicon wafer. Third, sections do not overlap. Fourth, sections are placed on the light background. Since this amount of images alone is not sufficient for the training, we will be generating artificial patches of different sizes using the information from these wafers.

3.2 Classes

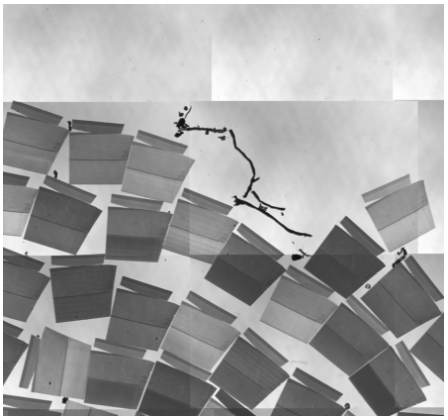
The classes we have are: *tissue*, *magnet*, *background*. The tissue and magnet class are visually very similar. It can get hard to differentiate the magnetic parts from those containing the brain sections. For that we use the fluorescent images.

The potential problems we can have when detecting these classes are:

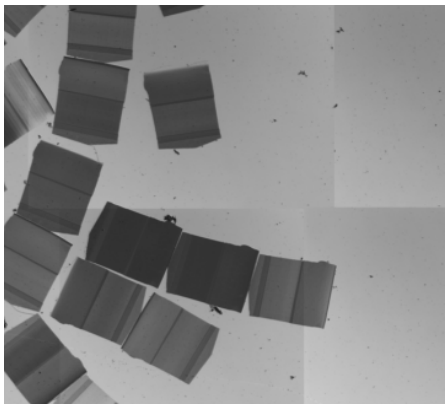
- Sections on one silicon wafer can look very different. Some of them can be darker than other ones due to the thickness of the section, see Fig.3.3a.
- Sometimes black bands can appear on the silicon wafer. They originate from the microscope's photography technique used, see Fig.3.3b.
- Magnet and tissue have very similar properties, see Fig.3.3c.
- Some sections can be slightly ripped, see Fig.3.3d. The sections were not ripped during the collection, but during subsequent handling (human mistake during microscopy).



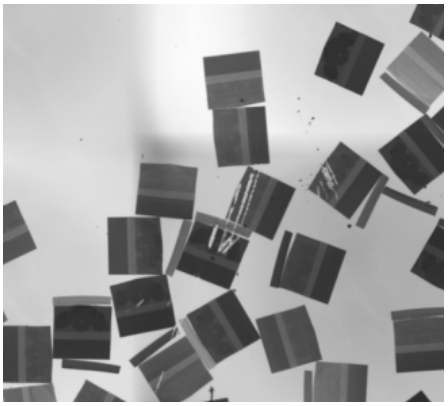
(a) Example of light (thin) and dark (thick) sections on the silicon wafer (image taken from Wafer 17)



(b) Example of black bands (image taken from Wafer 1)



(c) Example of similar looking brain and magnet part on the section (image taken from Wafer 2)



(d) Example of several ripped sections (image taken from Wafer 16)

Figure 3.3: Potential problems when detecting sections on the wafer

Chapter 4

Models and Methods

4.1 Pipeline

The task of the machine learning pipeline is to generate the list of the best section coordinates. Therefore, the count of detected sections as well as correctness of the final result are observed. The general workflow can be seen on Fig. 4.1.

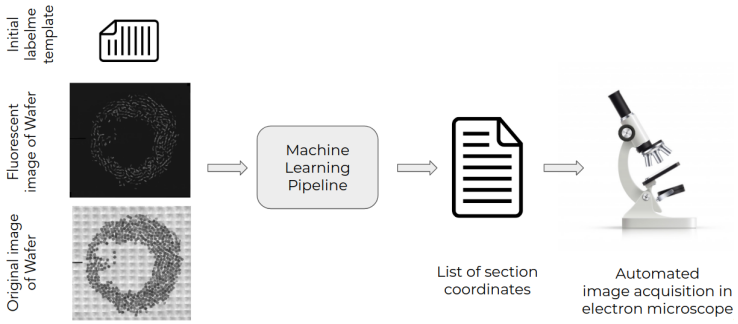


Figure 4.1: General workflow

4.1.1 Trimming Detectron Model

Since transfer learning is particularly useful for rapid progress and improved performance, we will use pre-trained models from Detectron [7]. Detectron is Facebook AI Research's software system that implements state-of-the-art object detection algorithms, including Mask R-CNN. The goal of Detectron is to provide a flexible codebase to support rapid implementation and evaluation of object detection research. In the Detectron github repository they provide a large set of trained models available for download[1]. Detectron uses the back-

[1] https://github.com/facebookresearch/Detectron/blob/master/MODEL_ZOO.md

bone models pretrained on ImageNet[2] and models are trained on the COCO dataset[3]. The Mask R-CNN github repository also provides the set of baselines[4] which are trained using the same setup as Detectron. For this project, the pre-trained model with backbone *X-101-32x8d-FPN* is used.

After the pre-trained model is downloaded, the first step of the machine learning pipeline has the goal to finetune Detectron weights on custom datasets, as described in the Mask R-CNN repository[5]. Since the last layers of this pre-trained model have classes that are different (e.g. cat, dog, etc.) from the classes we have in our dataset (tissue, magnet), we should not load these weights. For this we create the script called **trim_detectron_model.py** to remove keys corresponding to the last layer. Trimmed model is stored and ready to be used as a starting point in the Mask R-CNN training on our dataset.

4.1.2 Artificial Patch Generator

The goal of artificial patch generator is to create a high amount of images of different sizes that will be used in the training, see Fig.4.2. These images are created from the ML pipeline input data, see Fig.3.1. To make use of both input images we have, we combine them into 3-channel images. The first channel contains the original image; the second channel contains fluorescent image; and the third channel is empty.

For example, let's say that user annotated five sections and one background in the input template file. The artificial patch generator will use the background information to create the background of the patch. Also, it will use the information of five previously annotated different sections to generate new ones by rotating known ones and randomly throwing them on the patch background. This way of augmenting data is the reason why it is important to initially annotate as many different looking sections as possible.

With each new section on the patch, the annotation dictionary was filled in with the corresponding annotations. The set of transformations used on the section was the same for the contour points annotating the tissue and magnet parts. The annotation dictionary will be stored in a JSON file next to the directory with images. The structure of this annotation file is the same as the those used for the training on the COCO dataset[6]. Each patch has a unique set of annotations (contour points). Number of categories for each annotation we had is 3 (background=0, tissue=1, magnet=2). The script that performs artificial patch generator is called **artificial_patch_generator.py**. Some of the script input parameters that are used are specifying how data will be generated, e.g. number of patches to generate, number of angles to rotate the artificial patches, etc.

[2] http://www.image-net.org/

[3] http://cocodataset.org/#home

[4] https://github.com/facebookresearch/maskrcnn-benchmark/blob/master/MODEL_ZOO.md

[5] https://github.com/facebookresearch/maskrcnn-benchmark#finetuning-from-detectron-weights-on-custom-datasets

[6] http://www.immersivelimit.com/tutorials/create-coco-annotations-from-scratch/#coco-dataset-format

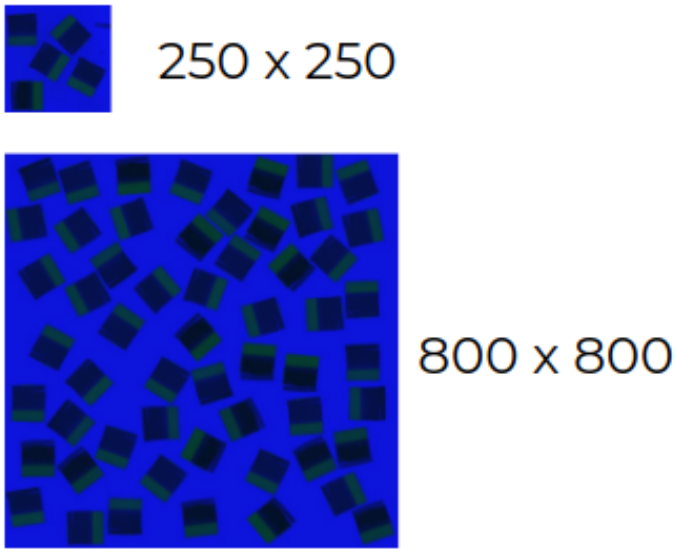


Figure 4.2: Different patch sizes

The output of this routine are generated train and validation sets as well as corresponding annotation files.

The general idea of artificial patch generation can be seen in Fig. 4.3.

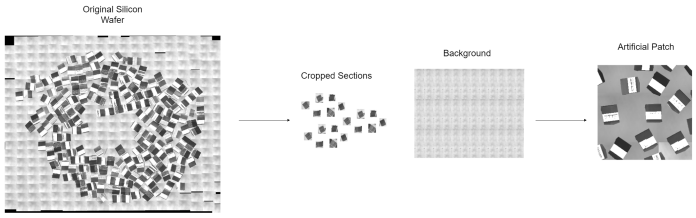


Figure 4.3: Idea of data augmentation [4]

What we are trying to do here is in fact to teach our neural network to be invariant, i.e. to recognize the features, regardless of the conditions that it is presented in. This process consists of using an image in the dataset, on which the model will theoretically have little difficulty to learn.

4.1.3 Training

The implementation of the project relies on existing framework Mask-RCNN developed by Facebook AI [6]. This framework is based on the open-source libraries PyTorch [9], Torchvision [11], and cocoapi [3]. Mask-RCNN [8] is flexible

and general framework for object instance segmentation. It relies on region proposals which are generated via a region proposal network (RPN). Mask R-CNN, extends Faster R-CNN [10] by adding a branch for predicting segmentation masks on each Region of Interest (RoI), in parallel with the existing branch for classification and bounding box regression, see Fig.4.4. Therefore, mask and class predictions are independent. Masks are covering the parts of the brain or magnet and class predicts if it is a brain or magnet. More information about details of implementation can be found in their paper [8].

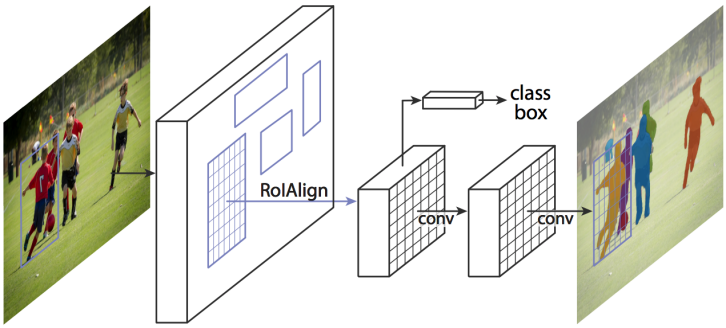


Figure 4.4: The Mask R-CNN framework for instance segmentation [8]

For the Mask-RCNN settings we used backbone R-101-FPN. For bounding boxes we use feature extractor called FPN2MLPFeatureExtractor and for masks we use MaskRCNNFPNFeatureExtractor.

For the training, we generated around 700 artificial patches, and for validation we created around 200 images. Also, we specified the anchor size to be in range from 4 to 64 when having smaller section sizes (like Wafer 17 and 16) and in range from 16 to 256 when having bigger section sizes. In case we were training all wafer images in parallel, this anchor size was changed accordingly to go from 8 to 128.

Mask R-CNN training is called with the following command:

```
python -m torch.distributed.launch --nproc_per_node=$NGPUS
    tools/train_net.py
    --config-file=/path/to/e2e_mask_rcnn_X_101_32x8d_FPN_1x.yaml
```

4.1.4 Inference

Inference is the part of the machine learning pipeline that is in charge of performing prediction on a original-size wafer image. In order to do that, the original image is cut into several subimages usually with the same dimensions as the patch dimensions used in the artificial patch generator. Then, on each of the subimages, the prediction is performed using the predictor class, implemented in **predictor.py**. The result of the prediction is the list of contours and its

labels (tissue or magnet) which we call candidates. The candidates will be used in the next step of the pipeline in order to merge the prediction results on the full wafer. The inference is implemented in the script called **inference.py**.

4.1.5 Wafer Merge

The wafer merge represents the last step of the machine learning pipeline. The main goal of this step is to find the best representatives of the sections. To achieve this, several steps are performed. First we need to explore the properties of the sections on one silicon wafer we got in the initial JSON template. The section properties (we call it template statistics) include the information of the tissue's and magnet's average area sizes, mean distance between tissue and magnet contour centroids, minimal distance between two magnet centroids or minimal distance between two tissue centroids. Therefore, that is all the information we extract from the template with, e.g. five, manually annotated sections.

Second, we start by filtering out the candidates that are smaller than or bigger than specified area size threshold, see Fig.4.5.

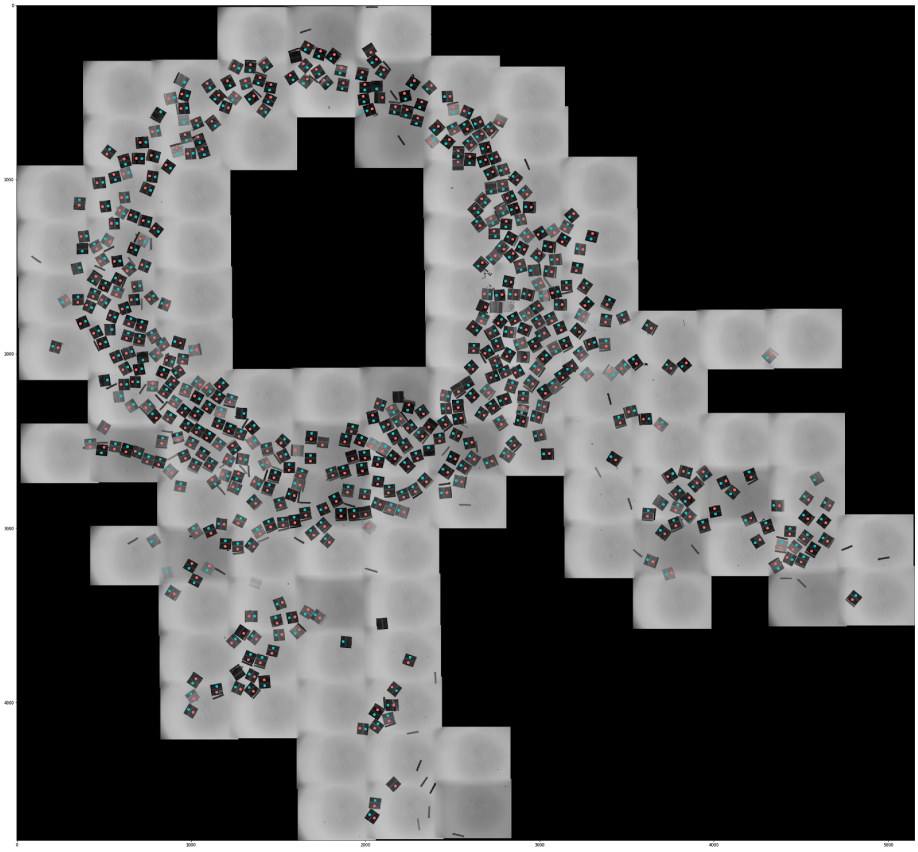


Figure 4.5: Detected centroids of each class

Afterwards, we pair the magnet with the corresponding tissue detection based on the previously calculated tissue-magnet distance, see Fig. 4.6.

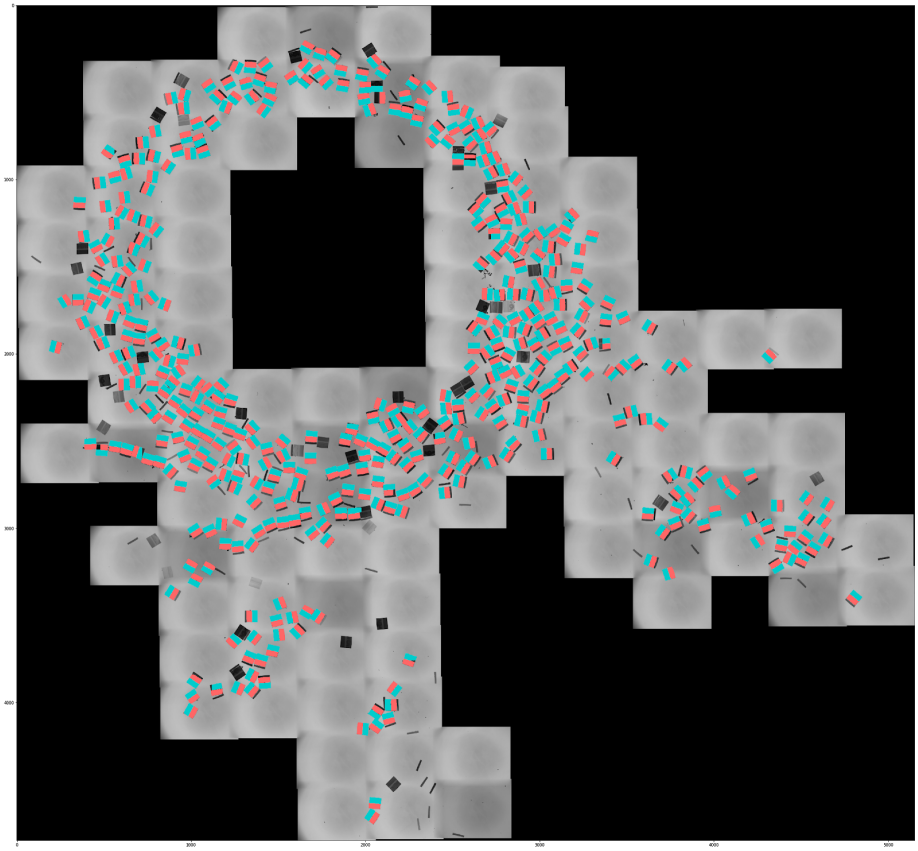


Figure 4.6: Section contours after filtering by area and pairing is finished

At the last step, we have found the contours and centroids of desired classes. However, these contours usually have way more than 4 points (which is enough to present the rectangle and provide microscope with). Therefore, there was a need to make an absolute template that will cover all detected classes. In addition to template statistics we mentioned in the beginning of this section, we also choose one template (out of five) that will be an absolute template, section shape representative. The absolute template is stored in the (0, 0) coordinates. Tissue and magnet have their one absolute template. For example, to cover one predicted tissue, we would translate the absolute template to the predicted tissue centroid. After, we rotate the absolute template based on the orientation we get from connecting this tissue's centroid with its magnet's centroids. This is done so the display and proofreading is simplified, see Fig. 4.7. The wafer merge is implemented in the script called **wafer_merge.py**.

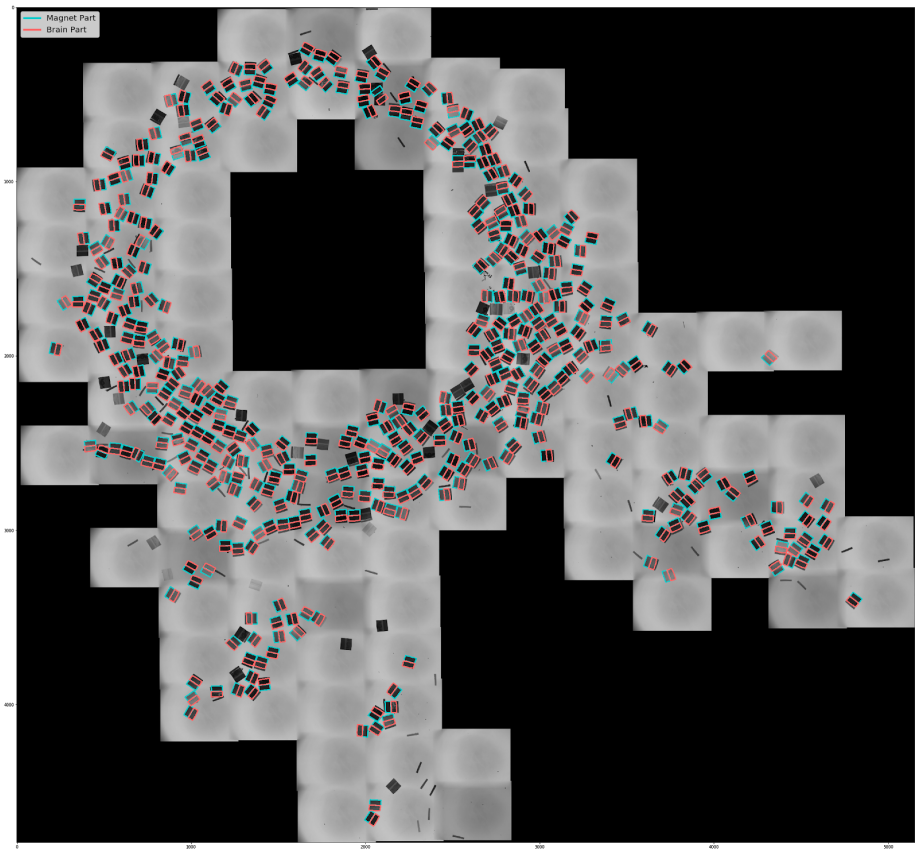


Figure 4.7: Result of the pipeline, final coordinates of detected classes

4.2 Software Tools

To call the pipeline, user first needs to specify where the input data is located (the directory). Then, user is able to initiate the pipeline with the following command:

```
usage: main.py [-h] --wafer-dir WAFER_DIR [--ngpus NGPUS]
               [--num-patches NUM_PATCHES] [--num-throws NUM_THROWS]
               [--num-angles NUM_ANGLES] [--ratio RATIO]
               [--patch-size PATCH_SIZE] [--area-limit AREA_LIMIT]
               [--min-distance-limit MIN_DISTANCE_LIMIT]

Machine Learning Pipeline for Brain Segmentation

optional arguments:
  -h, --help            show this help message and exit
  --wafer-dir WAFER_DIR
                        Directory that contains all necessary
                        data of the wafer
  --ngpus NGPUS         Number of GPUs used for training
  --num-patches NUM_PATCHES
                        Number of patches to generate with
                        artificial patch generator
  --num-throws NUM_THROWS
                        Number of throws used in artificial
                        patch generator
  --num-angles NUM_ANGLES
                        Number of angles to rotate generated
                        sections from 0 to 360 degrees in
                        artificial patch generator
  --ratio RATIO         Ratio of split data into training and
                        validation sets
  --patch-size PATCH_SIZE
                        Patch sizes to be generated in artificial
                        patch generator
  --area-limit AREA_LIMIT
                        Area threshold to use when merging wafer
  --min-distance-limit MIN_DISTANCE_LIMIT
                        Min distance threshold to use when merging wafer
```

4.3 Details and Specifications

In this section, we will just mention some of the hardware and software environment setup we had during the project development. This project was developed on Google Cloud Platform [2] virtual machine instance.

4.3.1 Hardware

For training, we used 4 NVIDIA Tesla K80 GPUs. Therefore, for training of 360 iteration, we spent around 15 minutes. For training of 3600 iterations, we spent around 2,5 hours. The hard disk size was 100 GB, which was enough to place a large amount of artificially generated data for the training and validation.

4.3.2 Software

Some of the software packages we used are:

- PyTorch version: 1.0.0.dev20190316
- CUDA 10.0
- CUDNN 7.4.2
- NCCL 2.2

For the Mask R-CNN package, INSTALL.md[7] instructions were followed.

[7] https://github.com/facebookresearch/maskrcnn-benchmark/blob/master/INSTALL.md

Chapter 5

Results

To determine the settings for which the model will detect the highest amount of correctly labeled sections, we performed several training with different settings in order to get the optimal model. In this chapter, we will discuss the effect of each setup on the performance of the model.

5.1 Patch Size

When generating artificial patches, the patch size is one of the parameters that defines the dimensions of the artificially generated image. The input images usually have dimensions around 5000x5000 px. From these input images, we create smaller patches that have dimensions that range from 200x200 px to 800x800 px. Also, the bigger patches contain more sections than the smaller patches. These new artificial images will later be used as a training and validation sets during the training. We assumed that changing the patch size will affect the performance of the training as well as the number of discovered sections. During this test, we worked only on one wafer, wafer 17. However, the final results did not show any improvement nor decline.

5.2 Number of Iterations

One of the next things we tested was the number of total iterations during the training on one wafer. The range of iterations we tested was in range from 180 to 3600. The results can be seen on Fig. 5.1.

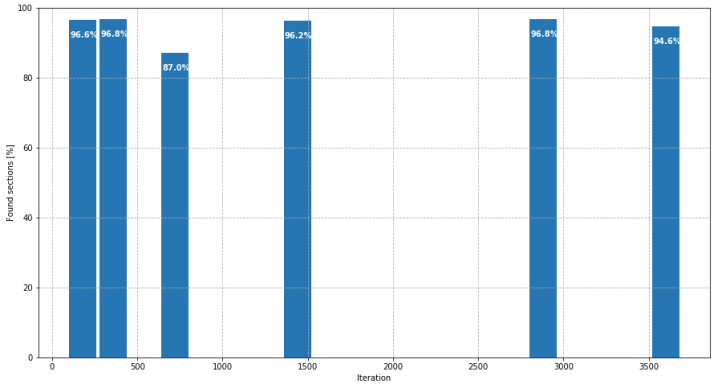


Figure 5.1: How number of iterations affects the final result

From the results we can see that number of detected sections at 180th iteration and 3600th iteration is very similar. The difference varies in around +- 2%. From this we concluded that the model reaches its peak in prediction score already at 360th iteration. Therefore, for further analysis we used this setting.

5.3 Number of Initial Templates

Number of initial templates is specified by user in the beginning of the pipeline. It is a file that contains several manually labeled sections. Therefore, each section has its own template. In this test we wanted to see how number of initially labeled sections affects the final prediction results. The results can be seen on Fig. 5.2.

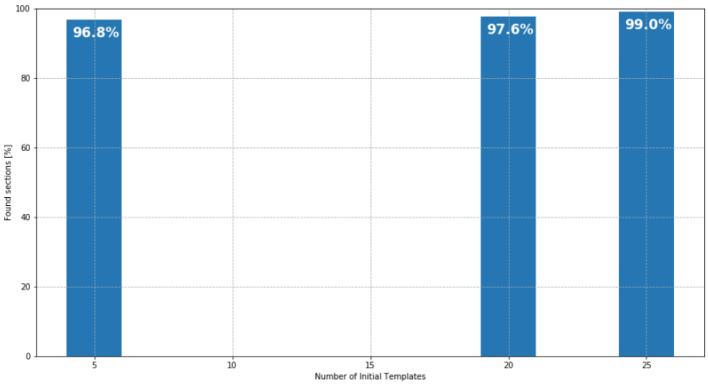


Figure 5.2: How number of templates affects the final result

We conclude that the number of detected sections increases by increasing the number of initial section templates. It is important to note that the best output

will be achieved if the user is manually labeling a diverse set of sections. For example, the initial template should contain darker (thicker) sections, lighter sections, sections that have small deformations, etc. The bigger is the variance of labeled sections, the higher is the number of correctly predicted sections.

5.4 Adding New Class

There was an idea of adding the new class for the training called section (see Fig. 5.3). Therefore, we modified our implementation to support prediction of the sections. In order to do that, we need to modify initial JSON template with manually annotated sections and adding the new polygon with the name section (s1). The artificial patch generator was slightly modified in order to include the new class in the annotations file used for training. New category was added, section=3. However, the way data was generated was not changed. For the training, we needed to change the configuration file for Mask R-CNN training, specifying the new value for NUM_CLASSES. For the last two steps of the pipeline, we add new class also called sections to our predictor which is used to predict the contours around our classes of interest.

Figure 5.3: Result of the pipeline, final coordinates of detected classes

After analysing the results, we saw that the performance of finding just magnet and brain tissue decreased. Without new section class, the discovery of sections was 490/496 (98.8%) sections, whereas with the new section class, the discovery of sections was 400/496 (80.65%). We can notice that adding new object class decreased the accuracy of object detection by 18.15% in the final result. Intuitively, we thought that the more classes we have the better model will be performing. However, if we think about the discriminant function approach, our inter-class variation was not big enough. The magnet and tissue classes already seem very similar. By adding the new class that contains them, we did not manage to draw the discriminant function between classes [1]. For the next step, we wanted to check if the detection of sections alone performs better.

After gathering the final output results, we saw that it performs much better than when combined with tissue and magnet classes. However, the discovery rate was around 2-3 new sections and the time spent detecting objects was doubled since we had to run the pipeline twice. We decided to discard this approach in benefit of time spent in the pipeline.

5.5 New Image Channel

When generating artificial patches, we use 3-channel images from which we create artificial patches. The first channel contains the original image, the second channel contains the fluorescent image, and the third channel contains blank image. Therefore, the third channel was not used. We wanted to see if adding new information to the third channel will improve the results. By adding preprocessed fluorescent image in the third channel, we did not see any change in the accuracy of object detection.

5.6 Parallel Training and Master Model

We wanted to create a master model that will perform well on different wafer images. In order to do that, we decided to run the training on all wafer images we had. The first step was to create artificial patches using templates from all wafers we have. This resulted in having more data for training and validation of the model. The results can be seen in the following tables:

Wafer name	# tissue	# magnet	# discovered	# total	%
Wafer 17	268	364	235	496	47.38
Wafer 16	213	331	187	433	43.19
Wafer 1	475	465	417	514	81.13
Wafer 2	378	484	316	503	62.82

Table 5.1: Anchor stride: (16, 32, 64, 128, 256), iteration: 360, confidence level: 0.7

In the table 5.1 we can notice that discovery percentage of Wafer 16 and 17 are very low, whereas Wafer 1 and 2 have around 15% higher discovery rate. When exploring the reason of this output, we found that the predefined anchor sizes used during the training could be the reason. This can be explained with the fact that Wafer 16 and 17 have a somewhat smaller area than Wafer 1 and 2. Therefore, shifting the anchor sizes to smaller dimensions, we reran our training. The results could be seen in the table 5.2.

Wafer name	# tissue	# magnet	# discovered	# total	%
Wafer 17	457	470	437	496	88.1
Wafer 16	390	414	377	433	87.07
Wafer 1	496	504	459	514	89.3
Wafer 2	544	592	449	503	89.26

Table 5.2: Anchor stride: (8, 16, 32, 64, 128), 360 iteration, confidence level: 0.7 - 88.43%

We can notice that the results got improved. For the next step, we wanted to see what will happen if we change the number of training iterations. When changing the number of iterations on Wafer 17 in the Section 5.2, we concluded that there is no effect on the final results. However, we wanted to see whether this applies to the parallel training that includes several different wafer images and section properties. We increased the number of iterations to 3600 and the results can be seen on Table 5.3.

Wafer name	# tissue	# magnet	# discovered	# total	%
Wafer 17	469	474	454	496	91.53
Wafer 16	398	412	389	433	89.84
Wafer 1	487	505	463	514	90.08
Wafer 2	495	557	419	503	83.3

Table 5.3: Anchor stride: (8, 16, 32, 64, 128), 3600 iteration, confidence level: 0.7 - 88.69%

The last setting we wanted to change was the confidence level setup. The confidence level specifies the threshold for prediction score, i.e. if the discovered contour has a prediction score lower than the confidence level, and we discard this candidate. Tables 5.4, 5.5, and 5.6 present the performance of the pipeline when the confidence level is modified.

Wafer name	# tissue	# magnet	# discovered	# total	%
Wafer 17	470	477	456	496	91.94
Wafer 16	401	416	392	433	90.53
Wafer 1	488	509	466	514	90.66
Wafer 2	509	569	425	503	84.49

Table 5.4: Anchor stride: (8, 16, 32, 64, 128), 3600 iteration, confidence level: 0.5 - 89.4%

Wafer name	# tissue	# magnet	# discovered	# total	%
Wafer 17	481	483	472	496	95.16
Wafer 16	407	420	399	433	92.15
Wafer 1	489	511	462	514	89.88
Wafer 2	537	575	443	503	88.07

Table 5.5: Anchor stride: (8, 16, 32, 64, 128), 3600 iteration, confidence level: 0.1 - 91.32%

Wafer name	# tissue	# magnet	# discovered	# total	%
Wafer 17	482	483	472	496	95.36
Wafer 16	410	421	403	433	93.07
Wafer 1	489	511	459	514	89.30
Wafer 2	547	574	448	503	89.07

Table 5.6: Anchor stride: (8, 16, 32, 64, 128), 3600 iteration, confidence level: 0 - 91.7%

Therefore, by creating a master model we generalized the model that will perform equally well on different wafer images, with around 91.7% accuracy of correct detection. However, when performing a training only on one wafer, the number of found correct sections was higher, 98.8%.

5.7 Combination of Predictions on Several Patch Sizes

After the training is finished, the next step is the inference. During the inference, a lot of small subimages of an original image are cut in order to run the prediction. The dimension of a subimage is usually in the same range as the patch size during the artificial patch generator. Each subimage prediction has an output that is a list of candidates. The candidates present the contour and the label of detected object. During this test, we wanted to see how does the change of subimage dimension affects the final results. During the training, we used patch size of 400x400. For the inference, we compared the results with 400x400 subimage and 800x800, see Table 5.7.

Subimage size	overlap ratio	# photos	# candidates	# discovery
800x800	0.5	154	3989	475/496
400x400	0.5	624	4920	480/496 (1 wrong)

Table 5.7: How changes of the subimage dimension affects the results

In addition, we decided to combine the size of subimages, the results were much better, see Table 5.8.

Subimage size	overlap ratio	# photos	# discovery
800x800 + 400x400	0.6	1245	484/496

Table 5.8: How combination of different subimage sizes affect the results

5.8 Overlap

The overlap is the parameter used during the inference. It specifies how much the subimages overlap. We define it as a number in range from 0 to 1. The higher the number, the bigger is the overlap. Some of the results we can see in the Table 5.9.

Subimage size	overlap ratio	# photos	# candidates	# discovery
400x400	0.5	624	4920	480/496 (1 wrong)
400x400	0.3	342	2525	481/496 (1 wrong)
400x400	0.2	255	1978	480/496 (1 wrong)

Table 5.9: How changes of the subimage affect the results

The observation was done on the subimages of size 400x400 px. We expected to see that the number of discovery changes w.r.t. overlap threshold changes and follows some trend. From the Table 5.9 we can see that by increasing the overlap ration, the number of photos is increased, as well as, the number of candidates is increased. Since we had more candidates, we expected to have bigger discovery rate. However, the results were not varying much.

Chapter 6

Discussion

At the end of the project, we finished and polished the section segmentation until the end. The developed software tools are now organized and easy to use. In the report, we discussed different settings that we have implemented to make our model more efficient.

We were mostly hoping to make a master model that would be able to detect the sections on different silicon wafers with at least 95% precision without providing manually labeled initial file. But, the current implementation showed that the best performance is reached when wafers are trained separately (with the precision above 95%). In addition, even if we had master model that works better than the one trained on one wafer, we will still need manually annotated sections on wafer. Therefore, as a final decision we choose to perform full pipeline on single wafer images.

Another idea was to implement a website platform, where users would be able to drag and drop the images and receive the prediction in a form of a JSON labelme file. With the master model, the implementation of the website could be done on the default server that has no GPU support. Since we found out that the master model performs less well, the one pipeline per wafer implementation would need the same settings as our virtual machine used on the Google Cloud with 4 GPUs. The solution would be to use the services provided by Google AI Platform[1] or PyTorch on AWS[2] that support training applications in the cloud. Therefore, the next step for this project would be to create a web platform that trains the model in the cloud.

[1] https://cloud.google.com/ai-platform/
[2] https://aws.amazon.com/pytorch/

Chapter 7

Summary

The challenge of the project was to correctly detect sections containing brain tissue on a silicon wafer while fine-tuning the parameters of the machine learning pipeline. This machine learning pipeline includes creating the pretrained model, generating artificial patches, training using Mask R-CNN framework, and making a prediction.

While exploring different results, we learned how different setup affects the performance of the pipeline. In the end, we successfully managed to create easy-to-use software tools that would detect and precisely segment the magnets and tissue parts of the section. The implementation was also made modular keeping in mind the future support for cloud requests.

Chapter 8

Acknowledgements

Working on this project has been an amazing journey. I would like to express my deep gratitude to my supervisors for the opportunity to continue work on this project during the semester.

I would like to thank Thomas Templier for his patient professional guidance and useful critiques of this project. I got an amazing opportunity to share experience and gather new knowledge from Thomas.

I would also like to thank my professor Martin Jaggi for suggesting new questions to ask and giving a useful feedback on the project development and writing.

I would also like to extend my thanks to the people of the CIME laboratory for their help in offering me an additional place in the office.

Bibliography

[1] Felix Abramovich and Marianna Pensky. "Classication with many classes: challenges and pluses". en. In: (), p. 24.

[2] *Cloud Computing Services*. en. URL: https://cloud.google.com/ (visited on 06/06/2019).

[3] *COCO API: Dataset @ http://cocodataset.org/ . Contribute to cocodataset/-cocoapi development by creating an account on GitHub*. original-date: 2015-01-25T20:26:39Z. June 2019. URL: https://github.com/cocodataset/cocoapi (visited on 06/06/2019).

[4] *Detecting tissue and magnet parts using Mask R-CNN : BrainSegmentation/tissue-parts-detection*. original-date: 2018-11-18T16:54:11Z. Mar. 2019. URL: https://github.com/BrainSegmentation/tissue-parts-detection (visited on 06/06/2019).

[5] *FAIR's research platform for object detection research, implementing popular algorithms like Mask R-CNN and RetinaNet.: facebookresearch/Detectron*. original-date: 2017-10-05T17:32:00Z. June 2019. URL: https://github.com/facebookresearch/Detectron (visited on 06/06/2019).

[6] *Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch.: facebookresearch/maskrcnn-benchmark*. original-date: 2018-10-24T17:34:50Z. June 2019. URL: https://github.com/facebookresearch/maskrcnn-benchmark (visited on 06/06/2019).

[7] Ross Girshick et al. *Detectron*. https://github.com/facebookresearch/detectron. 2018.

[8] Kaiming He et al. "Mask R-CNN". In: *arXiv:1703.06870 [cs]* (Mar. 2017). arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870 (visited on 06/06/2019).

[9] *PyTorch*. en. URL: https://www.pytorch.org (visited on 06/06/2019).

[10] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 91–99. URL: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf.

[11] *Start Locally — PyTorch*. URL: https://pytorch.org/get-started/locally/ (visited on 06/06/2019).

[12] Kentaro Wada. *Image Polygonal Annotation with Python (polygon, rectangle, circle, line, point and image-level flag annotation).: wkentaro/labelme*. original-date: 2016-05-09T12:30:26Z. June 2019. URL: https://github.com/wkentaro/labelme (visited on 06/06/2019).