BACHELOR'S DEGREE THESIS

# Degree in Data Science and Engineering

**Title: Tumorous area detection in breast cancer biopsy images using multiresolution networks**

**Author: Fernando Gastón Codony**

**Advisor:** Verónica Vilaplana & Josep Ramon Casas

**Department:** Image Processing Group

**Month and year: June 2022**


UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

**Facultat d'Informàtica de Barcelona**
**Facultat de Matemàtiques i Estadística**
**Escola Tècnica Superior d'Enginyeria de Telecomunicació de**

## Abstract

In this project we want to perform tumorous area segmentation of breast tissue samples stained with Hematoxylin-Eosin. In particular, we want to detect in situ and invasive tumors separately. Since pathologists use multiple resolutions when performing this task, we have chosen and trained a multiresolution network for semantic segmentation called HookNet. The architecture of this network allows the segmentation model to take a pair of concentric images with different resolutions, this way we provide the network with contextual information as well as high resolution details of the cells. The predictions of HookNet obtained an average F-score of 0.71, outperforming a U-Net trained on image patches at 10x magnification level which obtained an F-score of 0.49. However, a single-resolution U-Net trained with imaged at 2.5x magnification level still beat HookNet with an F-score of 0.81.

### Keywords

Semantic segmentation, Deep Neural Networks, Breast Cancer, Multiresolution Networks

## Resum

En aquest projecte, volem fer detecció de zona tumoral de mostres de teixits mamaris tenyides amb Hematoxilina-Eosina. En particular, volem detectar tumors in situ i invasius per separat. Com els patòlegs utilitzen múltiples resolucions en la realització d'aquesta tasca, hem entrenat una xarxa de multiresolució per segmentació semàntica anomenada HookNet. L'arquitectura d'aquesta xarxa permet al model de segmentació prendre un parell d'imatges concèntrices amb diferents resolucions, d'aquesta manera proporcionem a la xarxa informació contextual, així com detalls d'alta resolució de les cèl·lules. Les prediccions de HookNet van obtenir un F-score mitjà de 0,71, superant una U-Net entrenada amb imatges a un nivell de zoom de 10x, que va obtenir un F-score de 0,49. No obstant això, una U-Net entrenada amb imatges amb una magnificació de 2,5x millora

els resultats de HookNet i obté un F-score de 0,81.

**Paraules clau**

Segmentació semàntica, Xarxes Neuronals, Càncer de mama, Xarxes multiresolució

## Resumen

En este proyecto queremos segmentar el área tumoral de muestras de tejido mamario teñidas con Hematoxilina-Eosina. En particular, queremos detectar tumores in situ e invasivos por separado. Dado que los patólogos utilizan múltiples resoluciones al realizar esta tarea, hemos elegido y entrenado una red multiresolución para segmentación semántica llamada HookNet. La arquitectura de esta red permite que el modelo de segmentación tome un par de imágenes concéntricas con diferentes resoluciones, de esta manera proporcionamos a la red información contextual y detalles de alta resolución del tejido. Las predicciones de HookNet obtuvieron un F-score promedio de 0,71, superando a una U-Net entrenada con imágenes con un nivel de aumento de 10x que obtuvo un F-score de 0,49. Sin embargo, una U-Net entrenada con imágenes a un nivel de aumento de 2,5x superó a HookNet con un Fścore de 0,81.

**Palabras clave**

Segmentación semántica, Redes neuronales, Cáncer de mama, Redes multiresolución

# Contents

# 1. Introduction

Breast cancer refers to a set of pathologies characterized by an uncontrollable growth of the cells in the breast which may develop into malignant tumors. It is one of the most commonly diagnosed types of cancer and, according to the European Cancer Information System [1], in 2020 a total of 34.088 new cases were diagnosed in Spain. Even though it is most often diagnosed in women, around 1% of all cases are diagnosed in men.

The diagnosis process involves analysing images of biopsy samples of breast tissue. Once the sample has been extracted it is prepared into slices that can be stained using several immunohistochemistry techniques that mark different structures of the tissue. The analysis of these images is usually performed by pathologists with the goal to detect tumorous areas within the breast tissue. This process can be time consuming and the goal of this project is to help pathologists by giving them tools to speed it up.

This project has been carried out at the Image Processing Group (GPI) at the Polytechnic University of Catalonia (UPC) as part of a broader project called DigiPatICS. DigiPatICS is a collaboration between the GPI and the Catalan Institute of Health (ICS) and its main goal is to optimize the diagnosis process within the ICS network of hospitals.

My project focuses on tumorous area detection and, in particular, we want to be able to distinguish two different kinds of tumorous areas: *in situ* tumors and invasive tumors. In order to do so, we develop neural networks designed to perform semantic segmentation. We will cover all of these concepts in Section 3.

# 2. Goals of the project

The main goal of this project is to explore the use of multiresolution neural networks to perform semantic segmentation of WSI. Pathologists often switch between different resolutions when visualizing tissue samples to determine the presence of tumor, therefore, some multiresolution approaches have been proposed for semantic segmentation of digital histopathology images. In particular, we want to detect and segment in situ and invasive tumors and the healthy regions in the tissue samples of breast cancer patients. The

objectives of this project are:

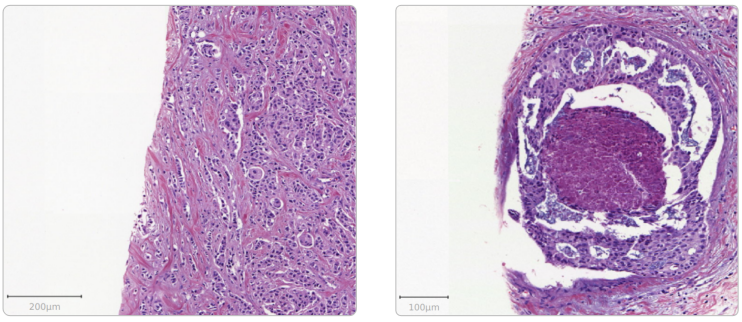
1. Bibliography research on multiresolution networks for semantic segmentation.
2. Choose and implement one of the multiresolution networks we study.
3. Use the multiresolution network to detect in situ and invasive tumors and healthy tissue in biopsy images of breast cancer.

# 3. Theoretical background

## 3.1 Breast cancer

As we stated earlier, breast cancer is characterized by an abnormal development of the breast cells. In general, when cell division takes place, there are a set of mechanisms that trigger the death of the cell (apoptosis) if some irreparable mutations take place during cell division. However, cancerous cells are those cells that develop mutations that are not repaired and keep dividing uncontrollably. These mutations are left unrepaired and are replicated when the cancerous cells divide.

There are many types of cancer and the one we focus on in this thesis are carcinomas, those that originate in epithelial cells. Epithelial cells are the cells that cover the surfaces of the body and the internal organs, such as skin and blood vessels. In this project, we refer to the epithelial cells of the breast which can be found in the lobules and the ducts




Invasive tumor *In situ* tumor

Figure 1: Differences between invasive and in situ tumor. Source: images provided by ICS

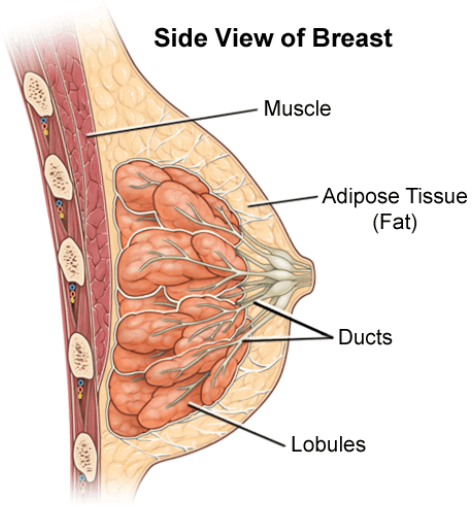



Figure 2: Breast biology diagram. Source: Source: "Normal Breast Development," Database, Ohio State: Wexner Medical Center.

of the breast (Fig. 2). The lobules are the glands that produce milk and the ducts are a set of thin tubes that transport the milk from the lobules to the nipple.

If the tumor is confined to the lobule or the duct we call them *in situ*, but if they spread to nearby tissues they are referred to as *invasive* (see Fig. 1). Given this taxonomy we can categorize breast cancer into ductal carcinoma in situ (DCIS), lobular carcinoma in situ (LCIS), invasive ductal carcinoma (IDC) and invasive lobular carcinoma (ILC), depending on the origin of the cancerous cells and whether or not the cancerous cells have spread to nearby tissue [2]. In this project we will not make any differences between lobular and ductal carcinoma: IDC and ILC will be labeled as "invasive", and DCIS and LCIS will be labeled as "in situ".

In order to do a pathological analysis and detect the presence of tumor, the following process is followed. Firsly, a sample of breast tissue is extracted and put into a mold with paraffin. The paraffin wax block is then cut into very thin slices that are stained using different immunohistochemistry (IHC) techniques. The stains that different projects at Digipatics work on are Hematoxylin-eosyn (HE), the marker of proliferation KI-67 (KI-67), Progesterone Receptors (PR), Estrogen Receptors (ER) and Human Epidermal growth factor Receptor (HER2).

Hematoxylin-eosyn is a dye combination that is widely used in histology because it

gives a good overview of the tissue sample and it is not expensive. Hematoxylin stains cell nuclei blue and eosin stains the cytoplasm pink. Other structures take on different intensities and hues that give more information about the structure of the tissue. HE is the first immunohistochemistry stain that is used by pathologists and it is used to determine the presence of tumor. This task is important because the presence of tumor determines whether or not more stains will be requested. Furthermore, the number of positive and negative cells for KI-67, RP, RE and HER2 is counted on tumorous areas only in order to give a score for every stain, which means an accurate detection of tumorous areas in H&E is crucial for the analysis of these stains. That is why having a tumorous area detection model could speed up the breast cancer detection process. Right now all of these tasks (cell counting and tumorous area detection) are done manually which is very time consuming. In this project we focus on the task of tumorous are detection over HE stained biopsy images.

### 3.1.1 Whole Slide Images

Once the stain has been applied to the tissue slice, it is digitized using a scanner which generates a file that we refer to as Whole Slide Image (WSI). The scanners used at the hospitals of the ICS network use a WSI format called MIRAX and have a resolution of to 0.25 $\mu$m per pixel at the highest magnification level.

These WSI are very big, since the resolution of the pixels is very high, in this project

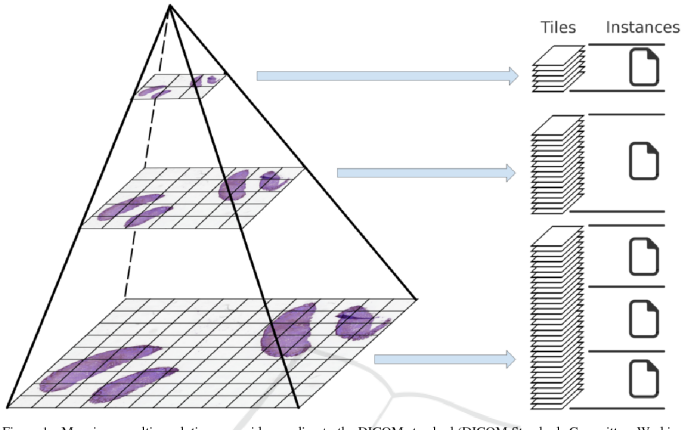



Figure 3: WSI multiresolution architecture and tile organization. cf. Jodogne et al. [3]

some of them can reach a size of about 200.000x200.000 pixels. The WSI stores this image at multiple resolution levels and tiled (Fig. 3) so that when it is being visualized using specialized software, such as QuPath [4], the user can navigate the image smoothly without the need to load the entire 200.000x200.000 image. When a certain region has to be loaded, only the necessary tiles of the image are read.

## 3.2 Semantic segmentation

Semantic segmentation is the task of assigning a class label to every pixel on an input image (Fig. 4).

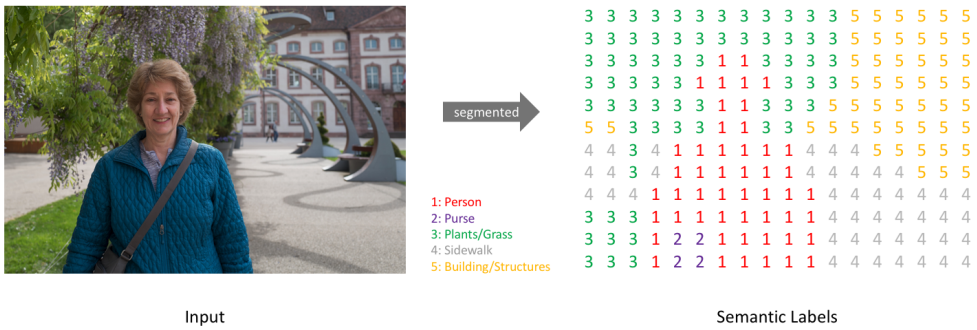



Figure 4: Semantic segmentation example of a natural scene. cf. Jeremy Jordan [5]

Medical image semantic segmentation is a very important field in computer vision and has been one of the most researched areas within semantic segmentation. Since the introduction of deep neural networks in computer vision, semantic segmentation tasks have been mainly tackled using deep learning. In particular, the U-Net architecture has been become the default architecture for these tasks.

U-Net has been the standard architecture for medical image semantic segmentation models ever since it was first introduced by Ronneberger et al. [6]. It has been used extensively in several biomedical image segmentation tasks (see [7] for an extensive review of the usage of U-Net in medical image segmentation) as well as image segmentation tasks in other areas (e.g. [8]).

U-Net is a Convolutional Neural Network (CNN), characterized by its encoder-decoder architecture (Fig. 5). The encoder and the decoder are also commonly referred

to as the contracting and expansive paths. The encoder's architecture is similar to that of CNNs trained for image classification tasks, consisting of convolutional and max-pooling layers that learn and downsample feature maps, respectively. The decoder is basically symmetric to the encoder with up-convolutions (upsampling operator) instead of max pooling, giving the architecture its characteristic 'U' shape from which the name U-Net stems. Finally, a set of skip connections connect the feature maps in the encoder with the corresponding feature maps in the decoder, to help with localization precision.

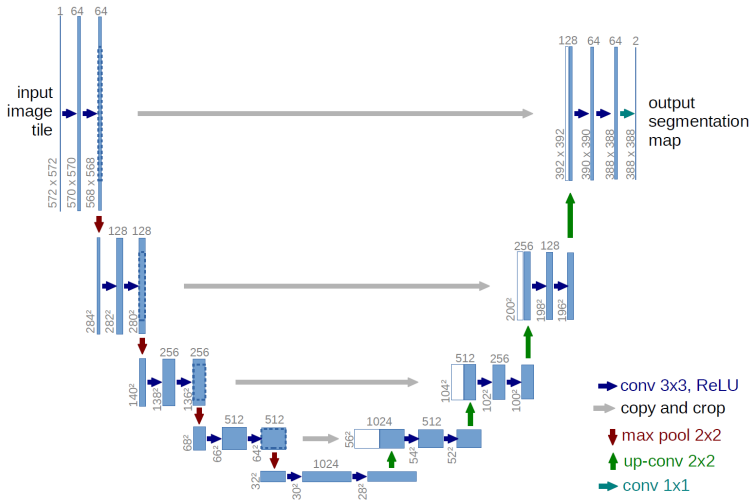



Figure 5: U-Net architecture. cf. Ronneberger et al.[6]

### 3.2.1 Tumorous area detection

The problem of detecting tumorous areas in WSI of breast tissue samples con be addressed as a semantic segmentation problem. The literature for this problem is extense.

In Digipatics some projects have been devoted to this task and this project is a continuation of those efforts. Firstly, Sonia Rabanaque's bachelor thesis [9] focused on segmenting WSI into several different regions using the Breast Cancer Semantic Segmentation (BCSS) dataset [10]. This is an external large-scale dataset containing thousands of segmentation annotations of breast tissue samples. The original dataset used 21 different classes, but Sonia reduced the number of classes to just 5, namely "tumor", "stroma", "inflammatory infiltration", "necrosis" and "other", given the needs of the DigiPatICS project. The segmentation model used in this work was a Deep Neural

Network with a U-Net architecture [6]. She compared the use of different encoders and pretrained weights that the Segmentation Models Pytorch library [11] provides. The EfficientNet encoder [12] achieved the best overall performance and it was tested on the WSI that the Vall d'Hebron hospital provided to the DigiPatICS project. However, since the ground truth was not available for those WSI the results could only be assessed qualitatively.

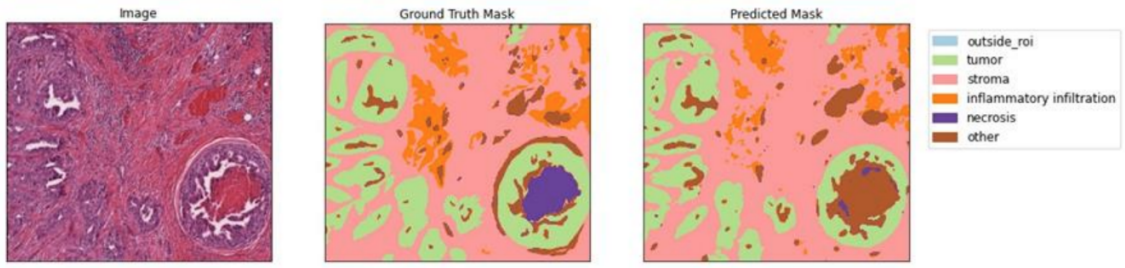



Figure 6: Prediction for a test image of the BCSS dataset. cf. Sonia Rabanaque [9]

Then, Arnau Turch did his Introduction to Research project on tumorous area detection of H&E stained WSI of breast tissue [13]. In this case, the dataset was generated with the help of pathologists from the ICS and consisted of only 3 classes "tumor", "other" and "background". A strong focus was put on selecting the magnification level of the input images of the segmentation model. In Sonia's bachelor thesis [9] the WSI tiles were extracted at 10x, 20x, and 40x (the best results were obtained with 20x) while in Arnau's project 10x, 5x and 2.5x were tested. This is very important, since pathologists often switch between different resolutions in order to study different structures of the tissue. The best performance was obtained when using 2.5x magnification. Furthermore, once this magnification level was selected, an extensive hyperparameter search was conducted. However, the model was very similar to Sonia's, a U-Net architecture with a pretrained encoder using the same Segmentation Models Pytorch library [11].

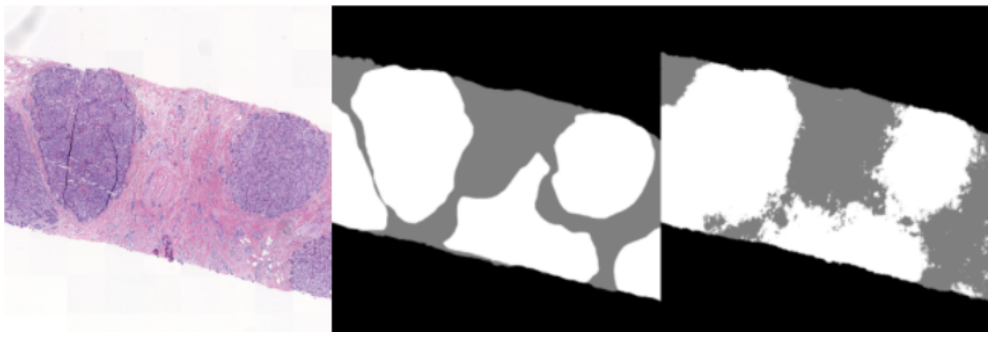


Figure 7: Tumorous area segmentation on a test image. From left to right: image, ground truth mask and prediction. White, black and gray correspond to tumorous area, background and other tissue, respectively. cf. Arnau Turch [13]

### 3.2.2 Multiresolution networks

Previous efforts in the Digipatics project to perform semantic segmentation of WSI have taken the same approach: extracting patches at different resolutions and selecting the resolution at which better results are obtained for the validation set. However, pathologists do not perform tumor detection this way. To detect the presence of tumors in a WSI and to decide whether a tumor is invasive or in situ pathologists usually zoom in and out and use different magnification levels and resolutions. On the one hand, they use low resolution information to see the global architecture of the tissue and its context. On the other hand, they use high resolution information to see the morphological details of the cells like the size and shape of the cytoplasm and the nucleus. Low resolution patches have a higher field of view than high resolution patches of the same size, but the morphological details of the cells and other fine details are lost.

In section 3.2 of "Deep Learning for Whole Slide Image Analysis: An Overview" [14] some alternatives to the patch extraction that previous Digipatics projects have used are presented. In particular, they mention using multiple patches extracted from multiple magnification levels. Basically, the idea is to extract concentric patches at multiple magnification levels and use all of them as input to your model instead of using just one resolution. In Figure 8 an example of a set of concentric multiresolution patches is shown. For example, Liu *et al.* [15] use this approach to detect metastasis in breast cancer.

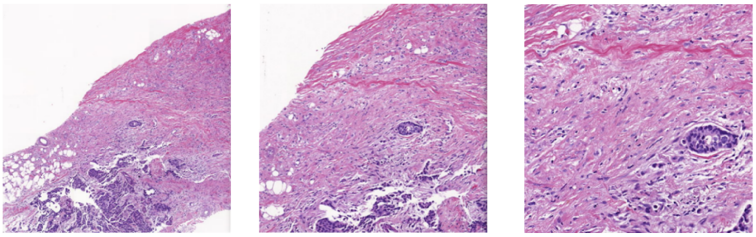


Figure 8: Concentric patches with the same size at different resolutions. From left to right: 10x, 5x and 2,5x

Some papers have also used this multiresolution approach for semantic segmentation of medical images. In Gu *et al.* [16] a multiresolution network made up of several encoders (one per resolution) and a single decoder is proposed. At every depth, the feature maps of all the encoders are concatenated with the feature maps of the decoder The network is then trained to predict the segmentation of the patch with the highest resolution. In Figure 9 the case where just two resolutions are used is shown. There are two separate encoders and each one of them takes a patch with a different resolution as input. The encoder at the top takes in an image with a resolution of $0.5\mu m$ per pixel (mpp) and the one at the bottom takes a patch with a lower resolution, $1\mu m$ per pixel.

Another paper that uses multiresolution patches for semantic segmentations tasks in digital histopathology is Rijthoven *et al.* [17]. They propose HookNet, a multiresolution network heavily inspired by the work of Gu *et al.*. Instead of having one encoder per input patch, HookNet uses an entire U-Net per input image, these U-Nets are called

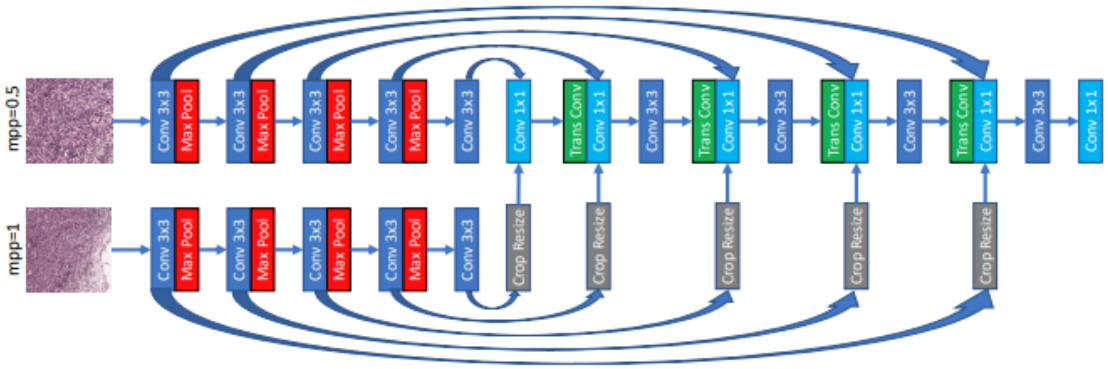



Figure 9: Architecture of the multiresolution network proposed by Gu *et al.* [16]

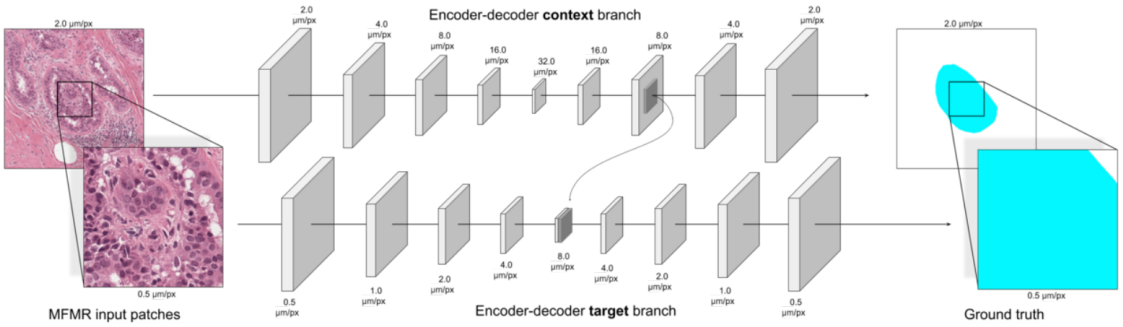



Figure 10: HookNet architecture. cf. Rijthoven et al. [17]

branches. In the case where 2 multiresolution patches are used, the branch that has the low resolution patch as input is called the context branch and the other is called the target branch. The idea is that by having multiple branches, we can compute the loss of the output of every branch. We can then compute a weighted average of the loss of the context branch and the loss of the target branch.

The context and target branches are connected through a hooking mechanism. The hook crops and concatenates one of the feature maps of the decoder of the context branch with the feature map of the bottleneck of the target branch, see Figure 10. The depth at which the hooking mechanism takes place is determined by the ratios between the resolution of the input patches. In the example shown in Fig. 10, the feature map that matches the resolution of the bottleneck of the target branch is at depth 2, since the ratio between the input resolutions is $2.0/0.5 = 4 = 2^2$.

# 4. Proposed solution

In order to detect in situ and invasive tumors in WSI, we will use the multiresolution approach proposed by Rijthoven et al. [17], i.e. HookNet.

## 4.1 Dataset

Firstly, we need to generate a dataset with annotated WSI of Hematoxylin and Eosin stained samples of breast tissue. The annotations have to include in situ and invasive tumors as well as a third tissue mask including all tissue other than tumorous areas. Since generating this dataset requires expertise in histopathology and great knowledge of the biology of breast cancer, we asked a pathologist from the Bellvitge hospital in L'Hospitalet de Llobregat to help us with the annotations.

To generate the dataset, we started by selecting a set of 6 WSI which contained a good variety of tissue samples. Thus, we selected a set of WSI that were diverse enough ensuring that we had some samples with in situ tumors. Furthermore, we also selected a WSI with a big surface of healthy tissue in order to ensure that we had examples of non-pathologic tissue in our training set. In this project we used QuPath [4] to annotate the WSI in the dataset.

Firstly, we met a pathologist at Bellvitge hospital and they annotated 6 WSI. Using the annotation tools in QuPath, they could easily define polygons around the in situ and invasive tumors in the WSI that we brought to the meeting. Then, we refined the annotations by setting more precise contours and adding an additional tissue mask. Finally, these annotations were shown to the pathologist in an additional meeting to revise them.

The resulting annotations are saved in a geojson file that can be loaded onto QuPath in order to be visualized or modified (Fig. 11). The resulting annotations have 4 classes: "background" for the non-tissue regions, "in situ" for in situ tumors, "invasive" for invasive tumors and "other" for tissue that is not tumorous.

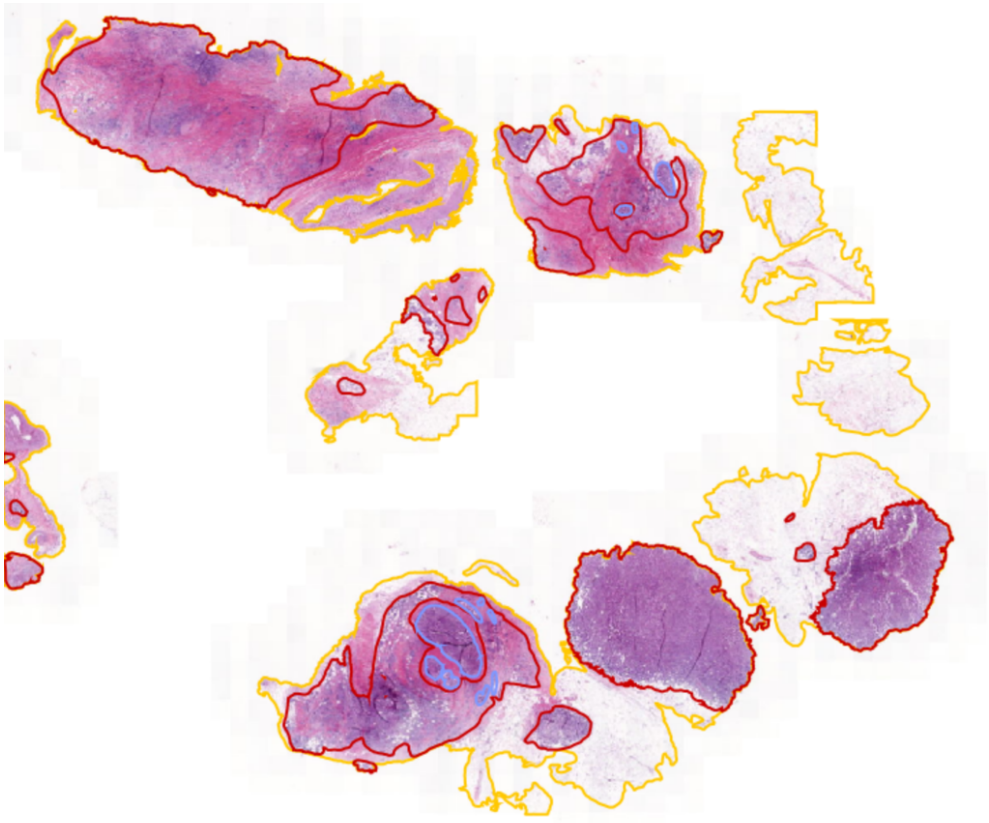


Figure 11: Example of an annotated WSI (5251 HE.mrxs). "other" in yellow, "invasive" in red, "in situ" in blue. Class "background" is not annotated directly, but any region that does not have an annotation corresponds to this class.

### 4.1.1 Concentric multiresolution patches

For this project we have chosen to use a pair of concentric multiresolution patches. The high resolution patch will be extracted at 10x magnification level and the low resolution patch will be extracted at 2.5x. The maximum magnification level of our WSI is 20x and corresponds to a resolution of 0.25 microns per pixel. That means that 10x magnification level corresponds to a resolution of 0.5 microns per pixel (downsampling by a factor of 2) and 2.5x corresponds to 2 microns per pixel (downsampling by a factor of 8). This pair of resolutions was chosen because it was used in the original HookNet paper. Both patches will have a width and height of 512 pixels.

QuPath can run Groovy[1] scripts to extract patches from a WSI and save them as regular image files (e.g. PNG file). Additionally, it is able to extract the corresponding mask at the desired resolution and save it as a PNG file as well. This way we can create the patches that one could use to train a regular U-Net. However, as we have previously mentioned, our solution will use a multiresolution network, so we need to generate concentric multiresolution patches.

First, we extract the high resolution patches (10x) and their corresponding masks using the Groovy script. These images are saved in PNG files whose name indicate the location of the patches within the WSI. With the information in the filename[2], one can easily find the coordinates of the central pixel. To extract the low-resolution concentric patch (2.5x), we use the Python interface to the OpenSlide C library [18] which provides several functions to interact with WSI. Using this library we can read a region of interest by indicating the top-left corner of the patch we desire and its height and width.

To extract the corresponding mask for that newly extracted patch, we load the geosjon file with the annotations of the whole slide image and use the Python library Shapely [19] to load the annotations as a set of polygons. Then, we use a square polygon corresponding to the bounding box of the patch to intersect the polygon of the annotations, that way we obtain the polygon of the annotation for the patch. Using

[1] a scripting language with Java-like syntax

[2] For example, 18B001854 HE [d=8,x=12288,y=94208,w=4096,h=4096].png is an example of a patch image file. 18B001854 HE is the name of the WSI the patch comes from, $d$ is the downsample which is related with the resolution of the patch, $x$ and $y$ are the coordinates of the top-left corner of the patch within the WSI, and $w$ and $h$ are width and the height of the patch (number of pixels in the highest resolution of the WSI)

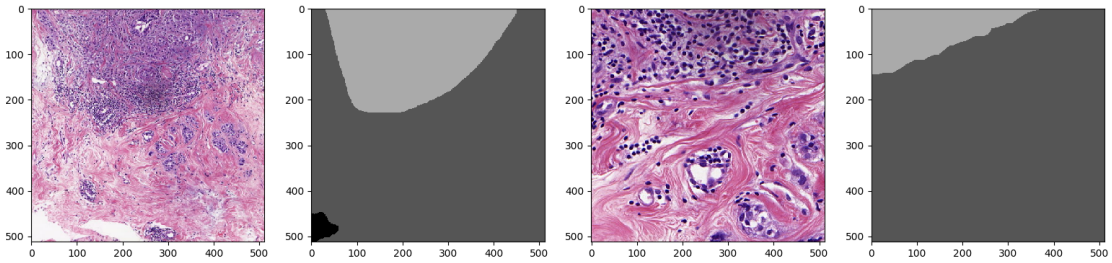


Figure 12: Example of a multiresolution patch. Left: image and mask at 2.5x magnification level. Right: concentric image and mask at 10x

then the scikit-image library [20] we can load the polygon into a numpy array that can be saved to a PNG file.

### 4.1.2 Train, validation and test split

One important question to consider about the evaluation of our model is how to split the patches we extracted from all the WSI into train, validation and test sets. Arnau Turch's I2R project [13], for example, randomly selected the train, validation and test sets among all the patches from all the WSI. That means that two patches that were extracted from the same WSI and that may correspond to very similar regions of tissue (e.g. side-by-side patches) could end up in the train and test set respectively. We think that this may create a bias in the validation and test metrics.

Ideally, we would like to evaluate our model on patches that come from a WSI that the model has not been trained on, that means that no training patch has been extracted from that WSI. However, since we only have 6 annotated WSI, we think that this strategy is not very appropriate So, instead we decided to select some individual tissue samples (cylinders) from the 6 WSI for validation and test. The patches extracted from these cylinders were used only for validation and test, the rest of the cylinders are used for train.

Using this approach, we got 3290 (86%), 265 (6%) and 360 (8%) pairs of multiresolution patches for train, validation and test, respectively.

## 4.2 HookNet

As we mentioned at the beginning of this section, we will use the multiresolution network HookNet to perform semantic segmentation of the insitu and invasive tumors and other tissue in the dataset we generated. The original implementation of the paper was published in TensorFlow [21]. Since the GPI at UPC uses PyTorch [22] for their deep learning models, we will have to implement the architecture from scratch. The implementation is in the appendix A.

Both the context and target branches correspond to a U-Net network (with depth equal to 4). At every depth of the encoder, U-Net uses a convolution block and a downsample operation. The convolution block is made up of three convolutional layers after every one of which a ReLu activation and a Batch Normalization layer are added. In the encoder, the first convolutional layer of the convolution block doubles the number of channels, therefore at every depth the number of channels of the feature maps is doubled. The initial number of channels is 16, and thus the number of channels at the bottleneck is $16 \cdot 2^4 = 256$. The downsample operation is the max-pooling operator. In the decoder the previous feature map is upsampled by a factor of 2 and concatenated with the feature map of the encoder at the same depth, after that a convolution block follows.

To perform the hooking mechanism however, the feature map at depth 2 of the decoder of the context branch is cropped and concatenated to the last feature map of the target branch before the decoder. Therefore, we need to perform an additional convolution block to preserve the number of channels that the feature map of the target branch had before the concatenation.

### 4.2.1 Loss function

We have chosen to use Negative Log-Likelihood (NLL) as the loss function for our model:

$$NLL(y, \hat{y}) = \sum_{i=1}^{n} (y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i))$$

where $n$ is the number of classes, $y_i$ is an indicator variable that has value 1 if the correct class is $i$ (0 otherwise) and $\hat{y}_i$ is the probablity that the model assigns to class $i$. The formula above is computed for every pixel of the output image. HookNet computes the loss of the prediction for both the context and the target branches and computes their weighted average.

$$L(y_t, y_c, \hat{y}_t, \hat{y}_c) = \lambda \cdot NLL(y_t, \hat{y}_t) + (1 - \lambda) \cdot NLL(y_c, \hat{y}_c)$$

where $y_t$ and $y_c$ are the ground truth for the pixels of the target and context images, and $\hat{y}_t$ and $\hat{y}_c$ are the predictions for those pixels. The parameter $\lambda$ which corresponds to the weight given to the target loss is a hyperparameter of the model, therefore several values should be tested.

It's also important to note that there is a very strong class imbalance in our dataset. In the target images of the training set, the 4 classes, i.e. background, other, in situ and invasive, have a relative frequency of 0.147, 0.545, 0.021 and 0.287, respectively. Therefore we decide to use a weighted version of NLL where each term in the sum is weighted by the inverse frequency ($\frac{1}{w_i}$) of the corresponding class.

$$NLL(y, \hat{y}) = \sum_{i=1}^{n} w_i (y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i))$$

where $w_i$ is the weight for class $i$.

### 4.2.2 Data augmentation

Since we have a limited amount of training data we will use several data augmentation strategies. Data augmentation is frequently used in deep learning in order to effectively increase the amount of data one uses to train a neural network. In our case, we will use the following data augmentation techniques:

- Flip: the image is mirrored along either the x axis (vertical flip) or the y axis (horizontal flip)
- Random rotation (90º): rotates the image by a multiple of 90 degrees.

All of these are implemented in the Albumentations library [23]. The same set of augmentations are applied to the context and target images.

### 4.2.3 Metrics

The metrics we will report for the models we train are precision, recall and F-score. These will be reported for every class and for every model we train. The metric we will use to decide if a model is better than another is the simple average of the F-score of the 4 classes. We use the simple average and not the weighted average because the in situ class has a very low representation (2%) and we are interested in a good F-score for this class.

Precision is a performance metric that measures the amount of false positives. For a given class, for example "invasive", the precision in the test set is computed by dividing the amount of pixels predicted as invasive that were actually invasive (true positive) by the total amount of pixels that were labels as invasive (true positive and false positive):

$$\text{Precision} = \frac{TP}{TP + FP}$$

On the other hand, recall measures the true positive rate. For a given class the recall is computed by dividing the number of correct detections (true positive) of the class divided by the total number of samples of that class (true positive and false negative):

$$\text{Recall} = \frac{TP}{TP + FN}$$

Finally, the F-score or F1 score is the harmonic mean of the recall and the precision. It measures a trade off between precision and recall and will only have a high value if both the precision and recall are high:

$$\text{F-score} = \frac{2}{\text{Recall}^{-1} + \text{Precision}^{-1}}$$

# 5. Results

All the neural networks we will present in this section have been trained in the CALCULA server of the Image Processing Group. All the experiments were tracked using Weights and Biases [24] a tool that can be used to track experiments by logging metrics, images and even plots.

Due to a lack of time we did not perform hyperparameter optimization. Therefore, all of the networks we present have been trained using the following paremeters:

- Loss function: the weighted version of the Negative Log-Likelihood loss presented in Section 5.1.
- Optimizer: the optimizer used in all cases is the Stochastic Gradient Descent (SGD) algorithm with a momentum of 0.9.
- Learning rate: the learning rate is set to 0.005.
- Batch size: the batch size is set to 4.
- Weight of target loss ($\lambda$): 0.8.

No L2 regularization is used, unless specified.

To select these parameters, we performed some experiments with different subsets of the training set in order to adjust the parameters we would use for the entire training set, making sure the models could fit the training data using these hyperparameters. That being said, with more time we would have liked to perform hyperparameter tuning, since many of these parameters are key for model performance.

In sections 5.1 and 5.2 we show the confusion matrices (for the train, validation and test sets) and loss curves of the models we have trained. Then, in section 5.3 the metrics for all the models are computed for the test set and compared.

## 5.1 HookNet

When training HookNet with the entire dataset an epoch took about 35 minutes. Due to the time limit of 24 hours per job in the SLURM queue in the CALCULA server, we could only execute around 38 epochs per job. Since at the end of those 40 epochs the validation loss was still decreasing, we decided to restart the training where we left off. In total, 94 epochs were completed. As stated in Section the $\lambda$ parameter of the HookNet loss is set to 0.8.

The results from the training are shown below, starting with the loss curves for the train and validation sets:


Figure 13: Loss curve for train (orange) and validation (blue) sets

Looking at Fig. 13, we can tell that there is in fact some overfit to the train data since the loss curve for the validation set starts to plateaus after epoch 40, while the loss of the train set keeps steadily decreasing after that point. Another interesting finding is in the evolution of the target and context losses that are plotted in Fig. 14. We can clearly see that the loss of the two branches show a strong correlation. As expected, we can also see that the weighted average of the context and target losses is closer to the target loss, since the weight for the target loss is 0.8. Initially, the target loss is higher that the context loss, but by the end of the training, they reach similar values.

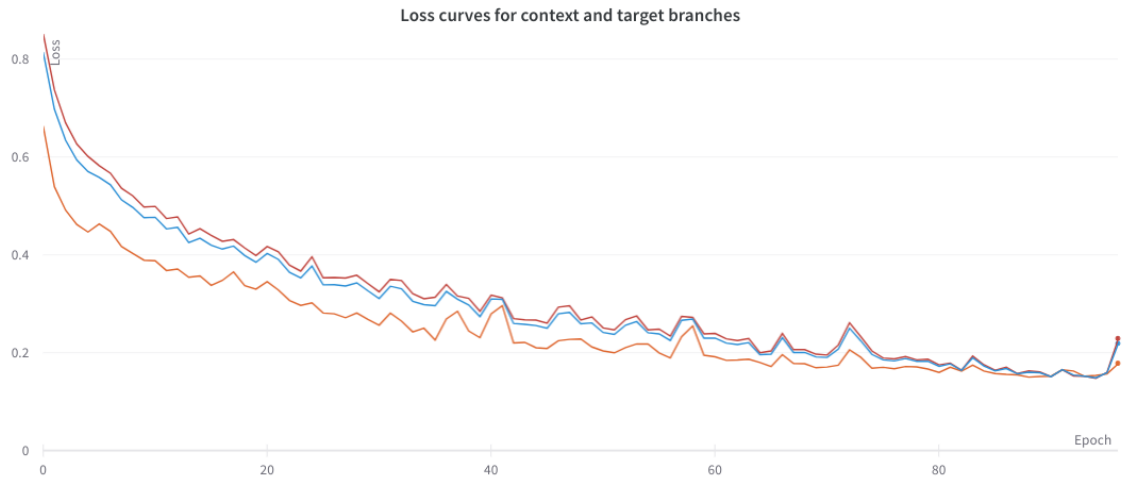



Figure 14: Loss curve for target branch (red), context branch (orange) and weighted average (blue)

Now, we will take a look at the confusion matrix of the predictions of the target branch for the train and validation sets (Fig. 15). In order to visualize them we will show the confusion matrices normalized by rows and columns. The y-axis for all the confusion matrices we will show correspond to the true labels and the x-axis corresponds to the predicted labels. Therefore, when we normalize the rows of the confusion matrices, we get the recall for every class in the diagonal and normalizing by columns we get the precision in the diagonal. This way we will be able to see the recall and precision for every class. When presenting the confusion matrices, we will specify whether the diagonal corresponds to the recall or the precision in the caption of the figure.

We can see that the distribution of the errors in the train, validation and test sets is fairly similar. As expected, the "background" has both a high recall and a high precision in both the train and validation sets, since this class should be relatively easy to segment. However, for the in situ and invasive classes there is a significant amount of errors.

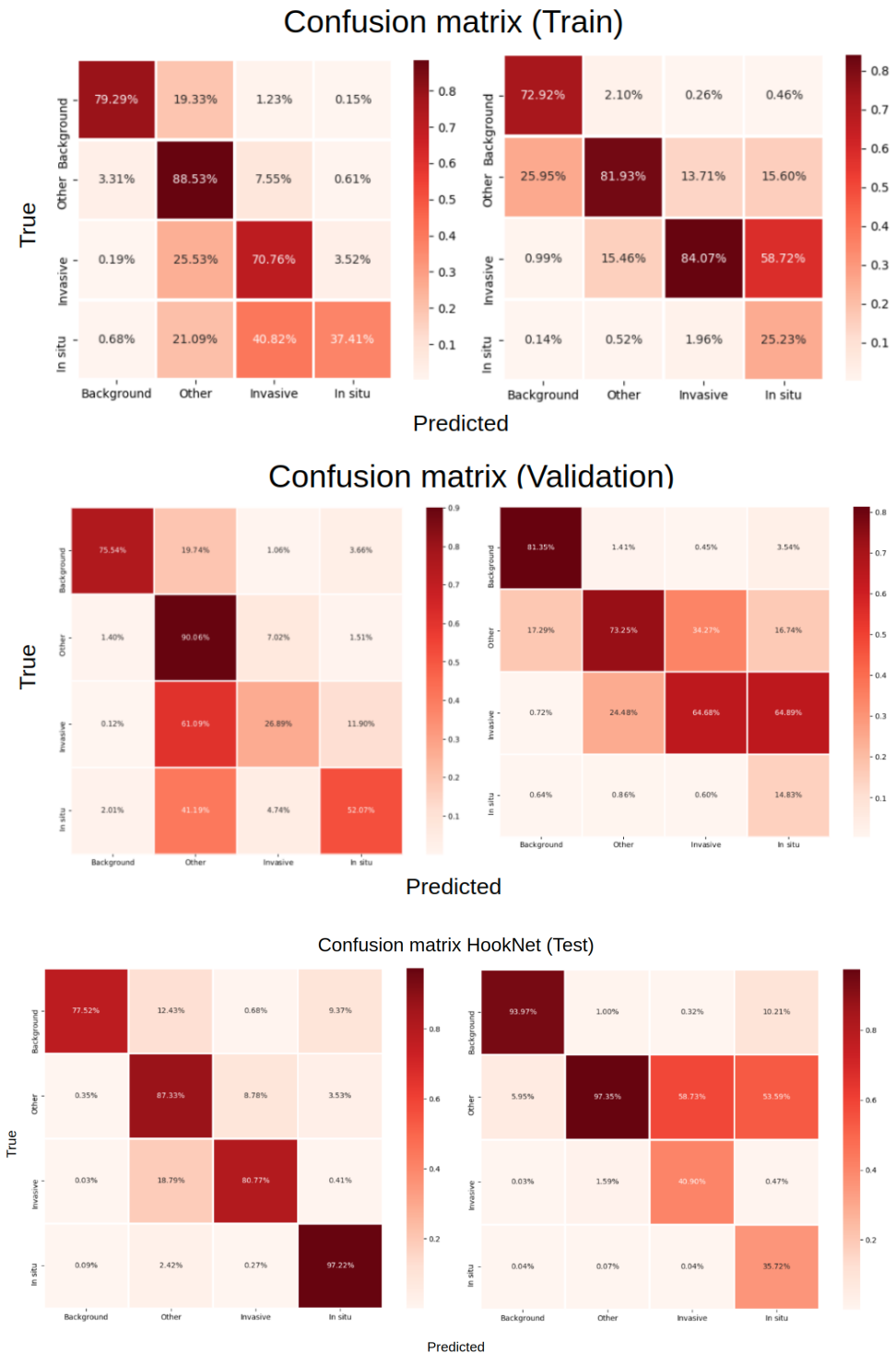



Figure 15: HookNet confusion matrices on the train and validation sets. Left: Recall. Right: Precision.

## 5.2 Single resolution U-Net

In order to evaluate the performance of HookNet, we will train some U-Nets from the Segmentation Models Pytorch library. This way we will be able to compare the results of HookNet with a simpler model and see if using the multiresolution approach contributes to the quality of the segmentation. In particular, we will use a U-Net with a VGG11 encoder pretrained on ImageNet, since it is the encoder architecture that is most similar to the branches of HookNet out of the encoders offered in the Segmentation Models PyTorch library. We will train two single-resolution U-Nets: one with images at 10x magnification level and the other with images at 2.5x corresponding to the input resolutions of the target and context branches, respectively.

### 5.2.1 Zoom level 2.5x

In this case, training the U-Net with the parameters we mentioned at the beginning of this section lead to a big overfit to the train set. Therefore, we added L2 regularization in order to prevent it. In PyTorch when using SGD as the optimizer, L2 regularization can be applied by adding the weight decay parameter to the optimizer which we set to 0.001.

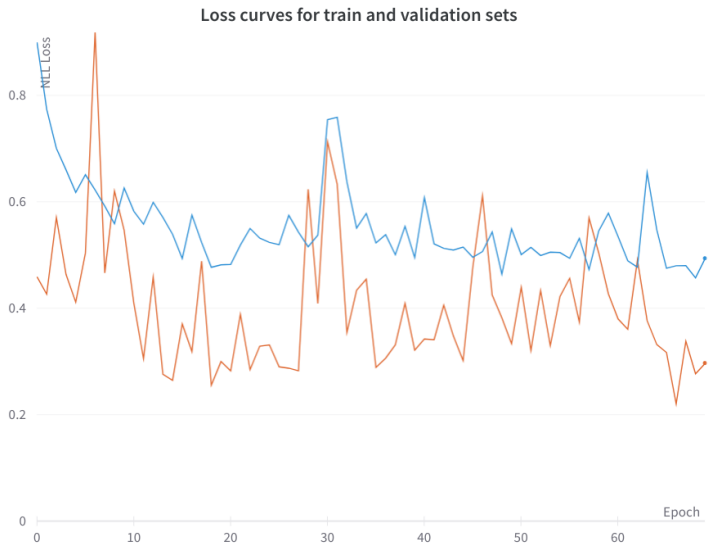



Figure 16: Validation (orange) and train (blue) loss curves, U-Net 2.5x

,

In Fig. 16 the loss curve is presented, we can see that the loss is quite noisy during training. However, the loss for the validation set is lower than for the train set and by using the L2 regularization we prevented the overfitting.

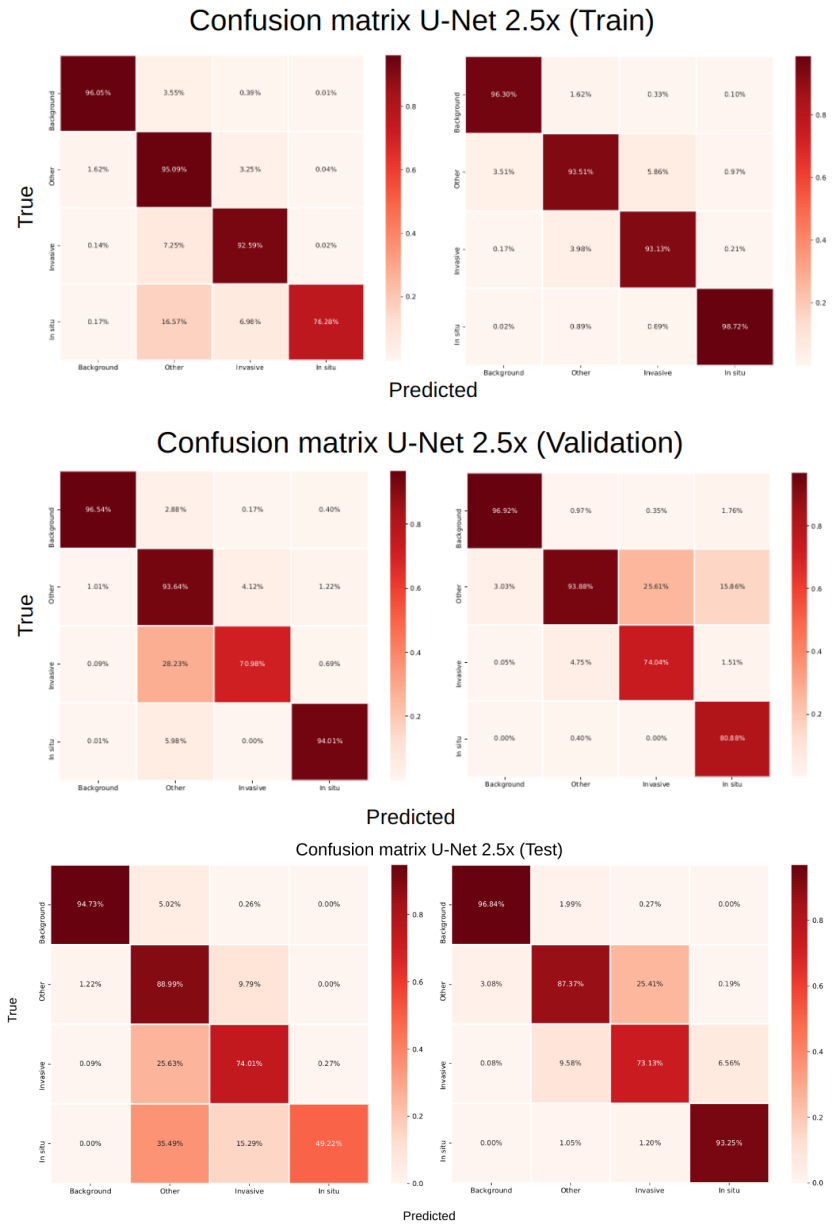



Figure 17: U-Net (2.5x) confusion matrices on the train, validation and test sets. Left: Recall. Right: Precision.

### 5.2.2 Zoom level 10x

Then, we trained a U-Net using images at 10x, which corresponds to the resolution of the target branch of HookNet. Similarly to what we saw in the loss curve of the U-Net trained at 2.5x (Fig. 16), the loss curve of the U-Net trained at 10x (Fig. 18) is not very stable.

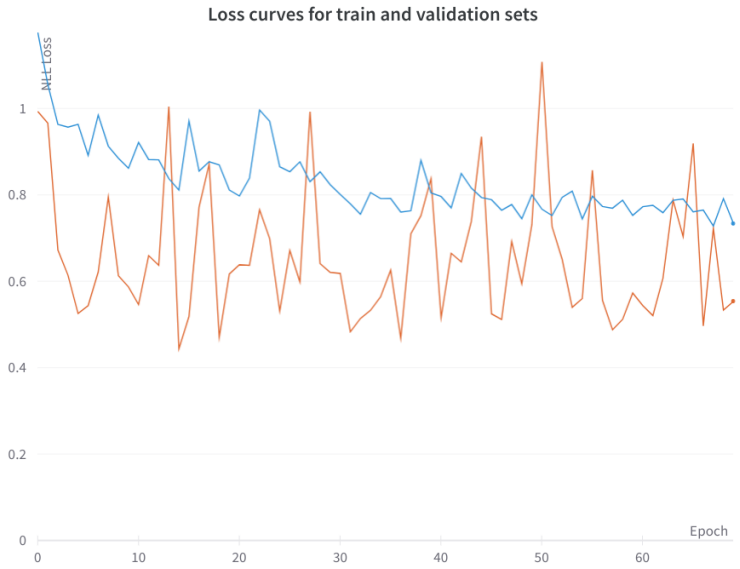



Figure 18: Loss curves U-Net 10x. Validation (orange) and train (blue)

From the confusion matrices in Fig. 19, we can see that the performance of the U-Net model at 10x is worse than the performance of the U-Net at 2.5x. These results correlate with the results Arnau Turch's I2R [13] obtained when evaluating several pretrained U-Nets at different resolutions, where the higher the resolution was, the lower the average F-score achieved.

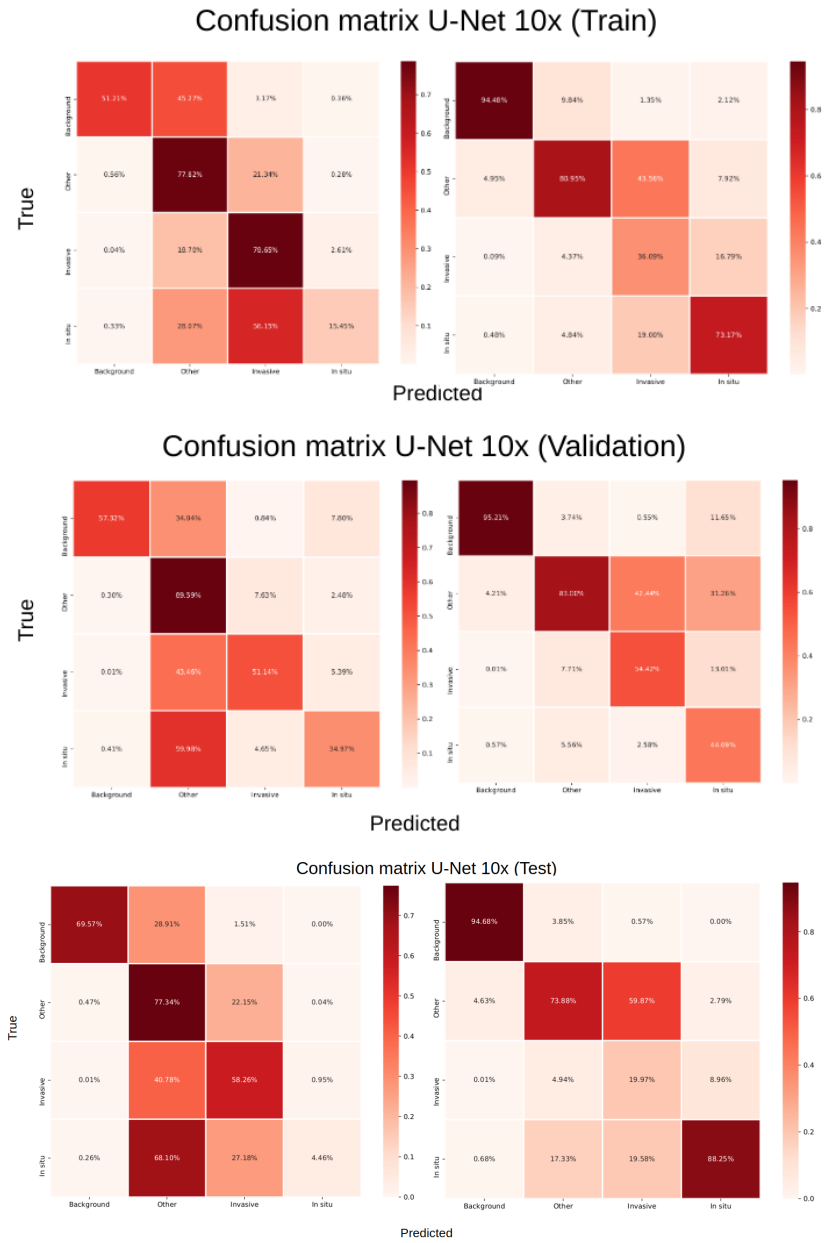



Figure 19: U-Net (10x) confusion matrices on the train, validation and test sets. Left: Recall. Right: Precision.

## 5.3 Model comparison

We can summarize all the information in the confusion matrices above by getting the precision and recall in the test set for each class and every model. Then, we can compute

the F-score for every class. All the metrics for every class are reported in Table 20.

| | Background | | | Other | | | Invasive | | | In situ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Model | Prec. | Rec. | F1 | Prec. | Rec. | F1 | Prec. | Rec. | F1 | Prec. | Rec. | F1 |
| **U-Net (2.5x)** | 0,97 | 0,95 | **0,96** | 0,87 | 0,89 | 0,88 | 0,73 | 0,74 | **0,74** | 0,93 | 0,49 | **0,64** |
| **U-Net (10x)** | 0,95 | 0,70 | 0,80 | 0,74 | 0,77 | 0,76 | 0,20 | 0,58 | 0,30 | 0,88 | 0,04 | 0,08 |
| **HookNet** | 0,94 | 0,78 | 0,85 | 0,97 | 0,87 | **0,92** | 0,41 | 0,81 | 0,54 | 0,36 | 0,97 | 0,53 |

Figure 20: Test metrics for all models and all classes

Now, we can average the F-score and the other metrics over all the classes for every model to see which one performs best (Table 21).

| | Average Precision | Average Recall | Average F-score |
|---|---|---|---|
| **U-Net (2.5x)** | 0,875 | 0,7675 | 0,805 |
| **U-Net (10x)** | 0,6925 | 0,5225 | 0,485 |
| **HookNet** | 0,67 | 0,8575 | 0,71 |

Figure 21: F-score average over the 4 classes for all models

The metrics show that the best tumorous detection is performed by the single-resolution U-Net that is trained on 2.5x images. Furthermore, it is also relevant to mention that the predictions of the target branch of HookNet outperform the single-resolution network trained on the target resolution (10x), the most likely cause is the additional contextual information provided by the context branch. However, one important thing to take into account is that we did not perform a very extensive hyperparameter search, therefore it could be that with a different set of hyperparameters the single-resolution U-Net at 10x performed better. Lastly, in figures 22 and 22 present some examples of the predictions of the three models make on the same test images.

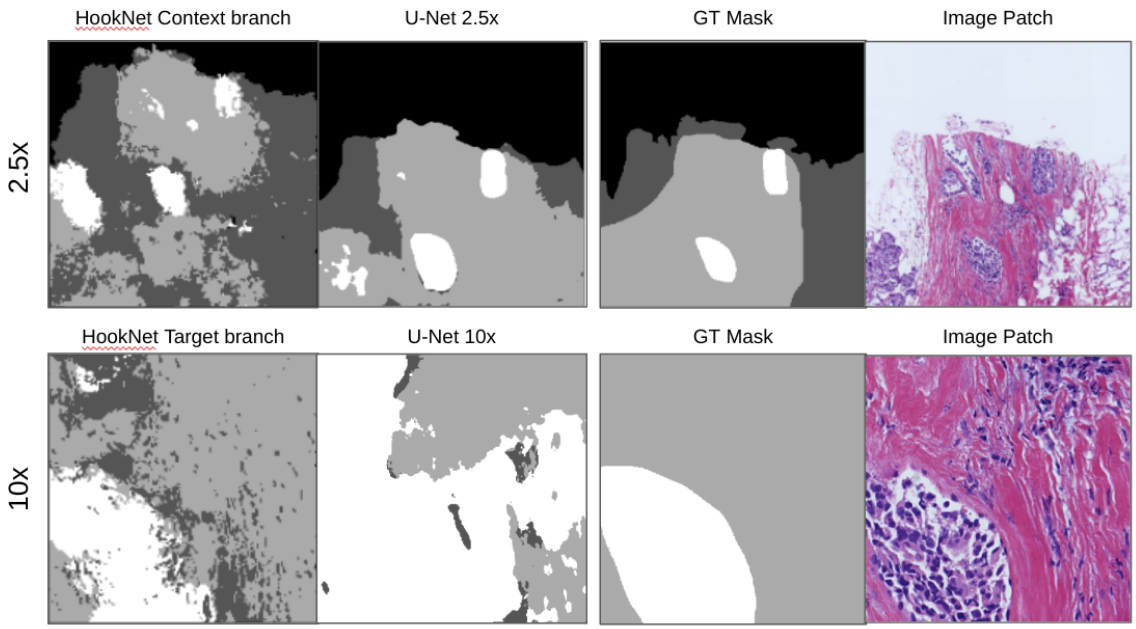



Figure 22: Example of prediction on test multiresolution patch (1)

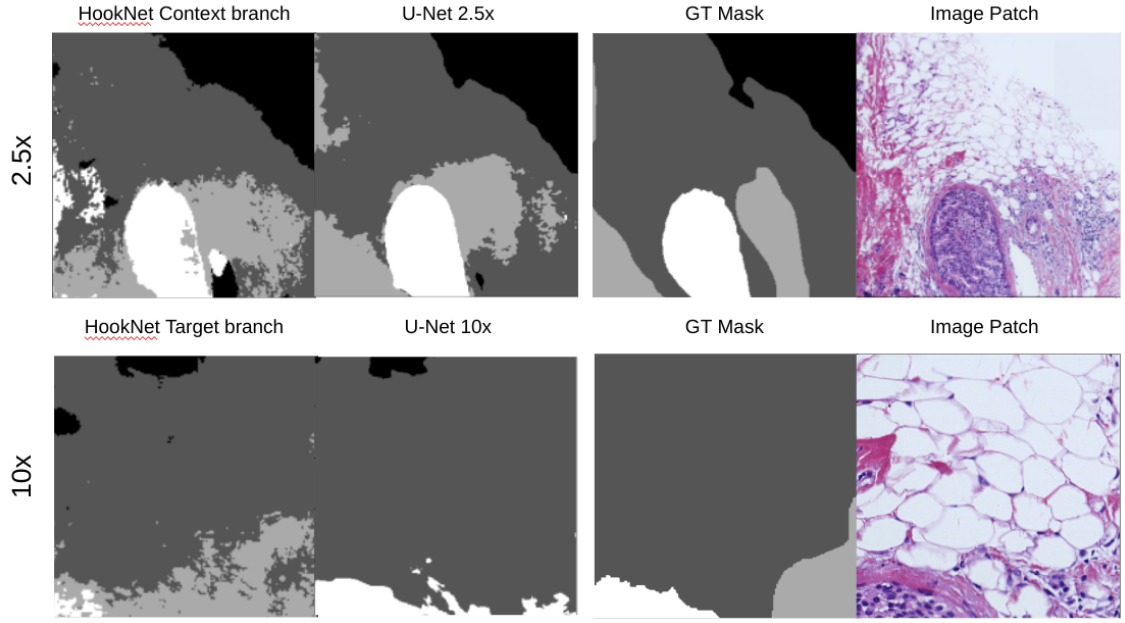



Figure 23: Example of prediction on test multiresolution patch (2)

# 6. Conclusions

In this project, we have implemented a multiresolution approach to semantic segmentation of digital pathology images that has not been previously used in the DigiPatICS project. We have used a multiresolution approach to segment the images using both high-resolution details of the tissue and its contextual information. The multiresolution network we used, HookNet, was implemented using PyTorch and was trained on a dataset we generated with the help from pathologists from the ICS network. The dataset consists of 6 whole-slide images annotated with 4 classes: in situ and invasive tumors, other tissue and the background (no tissue). HookNet performed better at segmenting the WSI at the target resolution (10x magnification level) than a single-resolution U-Net trained at that resolution without context information. However, a single-resolution U-Net trained at 2,5x magnification level achieved a better average f-score than HookNet.

In conclusion, we think the original goals for this project were met.

1. We reviewed the bibliography of multiresolution networks.
2. We chose HookNet for this project and implemented HookNet completely from scratch in PyTorch
3. We developed a dataset for tumorous area segmentation with in situ and invasive tumors and trained HookNet on it.

# 7. Future work

There are many other experiments that we would have liked to perform but were unable to due to the time constraints of this project. We did not have the final dataset until the beginning of May and the decision to use HookNet was introduced into the project in mid April, that's why many of the ideas we had for the project were not implemented in the end. Some experiments we would have liked to conduct are listed next.

First, we would like to train the single-resolution U-Nets with no pretraining. Since, HookNet did not have any pretrained weights it would be good to see the effect of the

pretraining in the U-Net and see if the U-Net at 2,5x would still outperform HookNet if it did not have pretraining. Similarly, we would have also liked to adapt our implementation of HookNet to be able to incorporate the pretrained weights provided by the Segmentation Models PyTorch library. Of course, we would have also liked to have enough time to perform hyperparameter tuning in order try to improve the results of all the models. This is a very important step when designing any deep learning model but we did not have enough time to do it.

Also, we would like to use different pairs of resolutions for the target and context branches. We set the context and target images to 2,5x and 10x magnification level because these were used in the original HookNet paper. However, they also tried other combinations, so it would be a good idea to try different pairs of multiresolution patches. In particular, seeing if the results of the U-Net trained at 2,5x could be improved if using 2,5x as the target resolution of HookNet and using a lower resolution (e.g. 1,25x) as context.

Lastly, getting more images for the dataset is clearly another area where we could improve, theis way we could provide the networks with more diverse traing data. Furthermore, we could test the models on unseen WSI, as opposed to unseen parts of the WSI, which could give a better idea of the generalization capabilities of these models.

# Bibliography


[1] "Breast cancer facts," https://www.europadonna.org/breast-cancer/, accessed: 2022-06-13.

[2] "Types of breast cancer - national breast cancer foundation," https://www.nationalbreastcancer.org/types-of-breast-cancer/, accessed: 2022-06-17.

[3] S. Jodogne, E. Lenaerts, L. Marquet, C. Erpicum, R. Greimers, P. Gillet, R. Hustinx, and P. Delvenne, "Open implementation of dicom for whole-slide microscopic imaging," in *VISIGRAPP*, 2017.

[4] P. Bankhead, M. B. Loughrey, J. A. Fernández, Y. Dombrowski, D. G. McArt, P. D. Dunne, S. McQuaid, R. T. Gray, L. J. Murray, H. G. Coleman *et al.*, "Qupath: Open

source software for digital pathology image analysis," *Scientific reports*, vol. 7, no. 1, pp. 1–7, 2017.

[5] "An overview of semantic segmentation," https://www.jeremyjordan.me/semantic-segmentation/, accessed: 2022-03-17.

[6] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[7] N. Siddique, S. Paheding, C. P. Elkin, and V. Devabhaktuni, "U-net and its variants for medical image segmentation: A review of theory and applications," *IEEE Access*, vol. 9, pp. 82 031–82 057, 2021.

[8] Z. Zhang, Q. Liu, and Y. Wang, "Road extraction by deep residual u-net," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 5, pp. 749–753, 2018.

[9] S. Rabanaque, "Breast tumor detection in hematoxylin eosin stained biopsy images," Bachelor Thesis, UPC, Data Science & Eng. degree, Jun. 2021.

[10] M. Amgad, L. Atteya, H. Hussein, K. Mohammed, E. Hafiz, M. Elsebaie, A. Alhusseiny, M. AlMoslemany, A. Elmatboly, P. Pappalardo, R. Sakr, P. Mobadersany, A. Rachid, A. Saad, A. Alkashash, I. Ruhban, A. Alrefai, N. Elgazar, A. Abdulkarim, and L. Cooper, "Nucls: A scalable crowdsourcing, deep learning approach and dataset for nucleus classification, localization and segmentation," 2021.

[11] P. Yakubovskiy, "Segmentation models pytorch," https://github.com/qubvel/segmentation_models.pytorch, 2020.

[12] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[13] A. Turch, "Upgrading the breast tumor detection in hematoxylin & eosin stained biopsy images," I2R Report, UPC, Data Science & Eng. degree, Jan. 2021.

[14] N. Dimitriou, O. Arandjelović, and P. D. Caie, "Deep learning for whole slide image analysis: an overview," *Frontiers in medicine*, p. 264, 2019.

[15] Y. Liu, K. Gadepalli, M. Norouzi, G. Dahl, T. Kohlberger, A. Boyko, S. Venugopalan, A. Timofeev, P. Nelson, G. Corrado, J. Hipp, L. Peng, and M. Stumpe, "Detecting cancer metastases on gigapixel pathology images," 03 2017.

[16] F. Gu, N. Burlutskiy, M. Andersson, and L. K. Wilén, "Multi-resolution networks for semantic segmentation in whole slide images," in *Computational Pathology and Ophthalmic Medical Image Analysis*. Springer, 2018, pp. 11–18.

[17] M. Van Rijthoven, M. Balkenhol, K. Siliņa, J. Van Der Laak, and F. Ciompi, "Hooknet: Multi-resolution convolutional neural networks for semantic segmentation in histopathology whole-slide images," *Medical Image Analysis*, vol. 68, p. 101890, 2021.

[18] "Openslide python 1.1.2 documentation," https://openslide.org/api/python/, accessed: 2022-06-13.

[19] S. Gillies *et al.*, "Shapely: manipulation and analysis of geometric objects," 2007–. [Online]. Available: https://github.com/shapely/shapely

[20] S. Van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu, "scikit-image: image processing in python," *PeerJ*, vol. 2, p. e453, 2014.

[21] "Github - diagnijmegen/pathology-hooknet," https://github.com/DIAGNijmegen/pathology-hooknet, accessed: 2022-06-14.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[23] E. K. V. I. I. A. Buslaev, A. Parinov and A. A. Kalinin, "Albumentations: fast and flexible image augmentations," *ArXiv e-prints*, 2018.

[24] L. Biewald, "Experiment tracking with weights and biases," 2020, software available from wandb.com. [Online]. Available: https://www.wandb.com/

# A. Hooknet implementation

```
class hooknet(torch.nn.Module):
  def __init__(self, depth, diff, activation, n_filters,
               num_classes=4, device):
      super(hooknet, self).__init__()

      self._depth = depth
      self._diff_res = diff
      self._device = device
      self.n_filters = n_filters
      self._activation = activation
      self.max_pool = nn.MaxPool2d(kernel_size=2, stride=2)

      # Context branch
      self.encoder_c = self.build_encoder()
      self.mid_conv_c = self.conv_block(n_filters*2**(depth-1),
                                        n_filters*2**depth)
      self.decoder_c = self.build_decoder()
      self.classification_head_c = nn.Conv2d(n_filters, num_classes,
                                             kernel_size=1)

      # Hooking conv block
      self.hook = nn.Conv2d(
                  n_filters*2**depth+n_filters*2**(depth-self._diff_res),
                  n_filters*2**depth, kernel_size=1)

      # Target branch
      self.encoder_t = self.build_encoder()
      self.mid_conv_t = self.conv_block(n_filters*2**(depth-1),
                                        n_filters*2**depth)
```

```
        self.decoder_t = self.build_decoder()
        self.classification_head_t = nn.Conv2d(n_filters, num_classes,
                                               kernel_size=1)

    def conv_block(self, in_channels, out_channels, kernel_size=3):
        conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size, padding="same"),
            nn.BatchNorm2d(out_channels, affine=False),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size, padding="same"),
            nn.BatchNorm2d(out_channels, affine=False),
            nn.ReLU(inplace=True),
        )
        return conv

    def build_encoder(self):
        modules_enc = nn.ModuleList([])
        in_channels=3
        n_filters=self.n_filters # default value
        for depth in range(self._depth):
            modules_enc.append(self.conv_block(in_channels,
                                               out_channels=n_filters))
            in_channels=n_filters
            n_filters*=2
        return modules_enc

    def build_decoder(self):
        modules_dec = nn.ModuleList([])
        n_filters=self.n_filters
        in_channels=n_filters*2**self._depth
        for depth in range(self._depth):
            modules_dec.append(nn.UpsamplingNearest2d(scale_factor=2))
```

```
62            modules_dec.append(self.conv_block(in_channels, in_channels//2))
63            modules_dec.append(self.conv_block(in_channels, in_channels//2))
64            in_channels //= 2
65        return modules_dec
66
67    def forward_encoder(self, X, branch):
68        residuals = []
69        if branch=="context":
70            encoder = self.encoder_c
71        elif branch=="target":
72            encoder = self.encoder_t
73
74        for module in  encoder:
75            res = module(X)
76            X = self.max_pool(res)
77            residuals.append(res)
78        return X, residuals
79
80    def concatenate(self, X, res):
81        size = (res.shape[-1]-X.shape[-1])//2
82        if size :
83            return torch.cat((X,res[:,:,size:-size,size:-size]), 1)
84        return torch.cat((X,res), 1)
85
86    def forward(self, X):
87        X_c = X[:,:3,:,:] # Context image
88        X_t = X[:,3:,:,:] # Target image
89
90        # Encoder
91        X_c, residuals_c = self.forward_encoder(X_c, "context")
92        X_t, residuals_t = self.forward_encoder(X_t, "target")
93
```

```
94          # Convolution block between encoder and decoder
95          X_c = self.mid_conv_c(X_c)
96          X_t = self.mid_conv_t(X_t)
97
98          # Context decoder (we have to get the hook as well)
99          skip=1
100         next_hook=False
101         for num_mod, module in enumerate(self.decoder_c):
102             X_c = module(X_c)
103
104             if next_hook: # Hooking taking place at this depth
105                 if next==1: # Last conv block at this depth has been applied
106                     hook=X_c
107                     next_hook=False
108                 else:
109                     next+=1
110             # Every three conv blocks there is a residual connection
111             if num_mod%3==1:
112                 X_c = self.concatenate(X_c, residuals_c[-skip])
113                 if len(residuals_c)-skip==len(residuals_c)-self._diff_res:
114                     # Hooking should happen when condition above holds
115                     next_hook=True
116                     next=0
117                 skip += 1
118
119         # HOOKING
120         X_t = self.concatenate(X_t, hook)
121         # Convolution to preserve number of channels before hooking
122         X_t = self.hook(X_t)
123
124         # Target decoder
125         skip=1
```

```
126         for num_mod, module in enumerate(self.decoder_t):
127             X_t = module(X_t)
128             # Every three conv blocks there is a residual connection
129             if num_mod%3==1:
130                 X_t = self.concatenate(X_t, residuals_t[-skip])
131                 skip += 1
132 
133         X_c = self.classification_head_c(X_c)
134         X_t = self.classification_head_t(X_t)
135 
136         return torch.cat((X_c, X_t), axis=1)
```