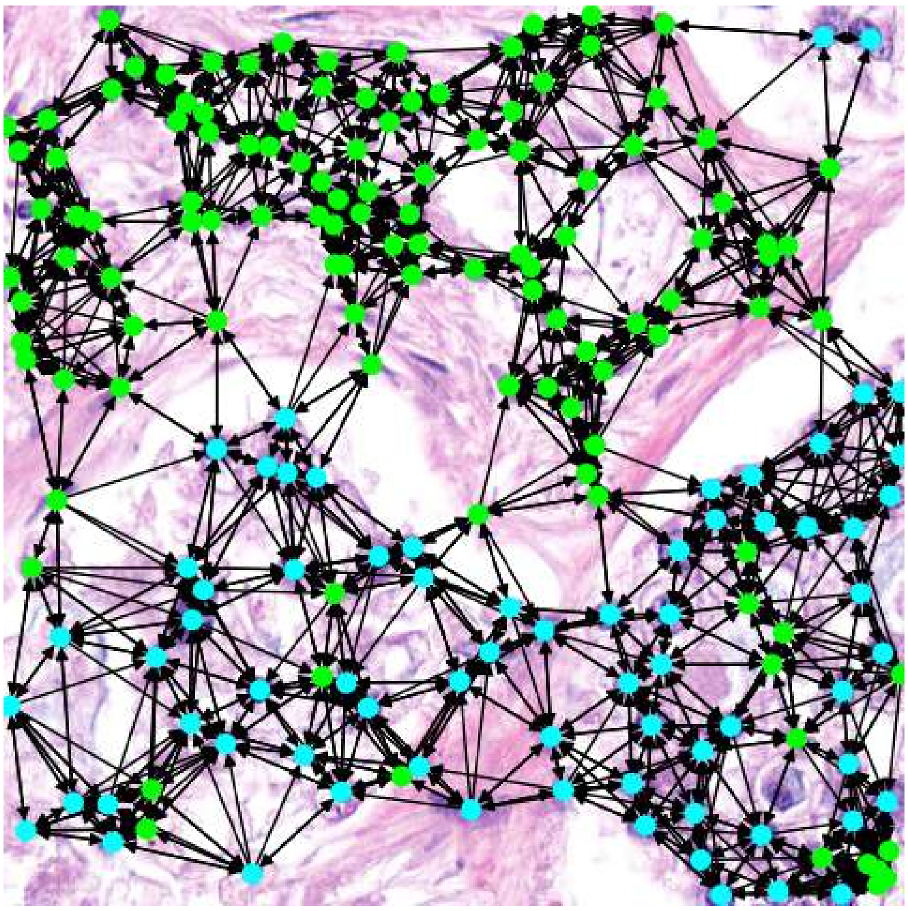
BACHELOR'S DEGREE IN MATHEMATICS

BACHELOR'S DEGREE IN DATA SCIENCE AND ENGINEERING

Automated detection of tumoural cells with graph neural networks



Author: Jose Pérez Cano
Supervisor: Philippe Salembier Clairon
Co-director: Ferran Marqués Acosta

June 2023

To my beloved family,
for their unconditional support.

Abstract

The detection of tumoural cells from whole slide images is an essential task in medical diagnosis and research. In this thesis, we propose and analyse a novel approach that combines computer vision-based models with graph neural networks to improve the accuracy of automated tumoural cell detection. Our proposal leverages the inherent structure and relationships between cells in the tissue. Experimental results on our own curated dataset shows that several different metrics improve by up to 15% compared to just using the computer vision approach. It has been proved to work with H&E stained lung tissue and HER2 stained breast tissue. We believe that our proposed method has the potential to improve the accuracy of automated tumoural cell detection, which can lead to accelerated diagnosis and research in the field by reducing the worload of hystopathologists.
Keywords: Histology, Lung, Breast, Graph Neural Networks, Convolutional Neural Networks, Calibration.
AMS Codes: 92C50, 92C55, 92C37, 68T10, 68T45, 68R10

Resumen

La detección de células tumorales en imágenes de portaobjeto completo juega un papel esencial en el diagnóstico médico y es un elemento fundamental de la investigación sobre el cáncer. En esta tesis proponemos y analizamos un enfoque novedoso que combina modelos de visión por ordenador con redes neuronales en grafos para mejorar la precisión de la detección automatizada de células tumorales. Nuestra propuesta aprovecha la estructura inherente y las relaciones entre las células del tejido. Los resultados experimentales obtenidos sobre nuestra propia base de datos muestran que varias métricas mejoran hasta en un 15 % en comparación con solo usar el enfoque de visión. Se ha demostrado que funciona con tejido pulmonar teñido con H&E y tejido mamario teñido con HER2. Creemos que nuestro método tiene el potencial de mejorar la precisión de los métodos automáticos de detección de células tumorales, lo que puede llevar a acelerar los diagnósticos y la investigación en este ámbito al reducir la carga de trabajo de los histopatólogos.
Palabras clave: Histología, Pulmón, Mama, Redes Neuronales en Grafos, Redes Neuronales Convolucionales, Calibración.
Códigos AMS: 92C50, 92C55, 92C37, 68T10, 68T45, 68R10

Resum

La detecció de cèl·lules tumorals en imatges de seccions completes és una tasca essencial en el diagnòstic mèdic i la investigació. En aquesta tesi, proposem i analitzem un enfocament innovador que combina models basats en visió amb xarxes neuronals en grafs per millorar la precisió de la detecció automatitzada de cèl·lules tumorals. La nostra proposta aprofita l'estructura inherent i les relacions entre cèl·lules en el teixit. Els resultats experimentals en el nostre propi conjunt de dades curat mostrin que diversos indicadors milloren fins a un 15% en comparació amb només usar l'enfocament de visió. S'ha demostrat que funciona amb teixit pulmonar tenyit amb H&E i teixit mamari tenyit amb HER2. Creiem que el nostre mètode proposat té el potencial de millorar la precisió de la detecció automatitzada de cèl·lules tumorals, el que pot portar a uns diagnòstics més ràpids i una investigació accelerada en el camp degut a la reducció en la càrrega de treball dels histopatòlegs.
Paraulas clau: Histologia, Pulmó, Mama, Xarxes Neuronals en Grafs, Xarxes Neuronals Convolucionals, Calibració.
Codis AMS: 92C50, 92C55, 92C37, 68T10, 68T45, 68R10

Contents

Chapter 1

Introduction

1.1 When technology meets histology

This project was born due to the necessity of alleviating physicians and researchers workload in the field of digital pathology. On a typical day, our experts have to go through a wide variety of tissue images in order to detect some anomaly or disease. Depending on the task at hand they sometimes need to estimate the proportion of tumoural cells with respect to healthy ones. How wonderful would it be if a machine could perform that task for them. That is the purpose of this thesis, to automatically estimate the percentage of tumoural cells with respect to non-tumoural ones and facilitate the physician's job.

Analysing human tissue is a challenging task. The first part of the process consists of extracting the tissue from the part of the body that is relevant to the patient's condition. If the patient is alive the extraction is typically done using a needle. Otherwise, if the organ is already removed then a cube of tissue can be sliced. Then, simply watching those slices through a microscope is not going to be enough. Cells are very small and their interior is difficult to observe even when looking through the lenses of a microscope. For that reason tissue is stained prior to observing it. There are different kind of staining, some of them highlight the cell nuclei, others the membrane and other stainings react with specific kind of cells. Another factor to take into account is the cost of the dye. Some of them make the task at hand easier but are too expensive to do for every patient. A trade-off is usually found where the more expensive one is used only when the cheaper is not enough for confidently diagnosing.

In the past, the surgically extracted slices were typically watched through the lenses of a microscope [9], what is termed as optical microscopy. With the development of new technologies, the field has become more and more digitalised [17], now being called digital pathology. The resulting digital images that originate from watching through the microscope the tissue are called Whole Slide Images (WSI). Between when a biopsy procedure is made and when the specialist watches it, there is now a period of time required to digitalise it. In other words, at the morning one specialist carries out the removal of the tissue. Afterwards, technicians digitalise the image in the afternoon. It is in the next morning the physician watches the result. Taking advantage of that interval between when the image is digitised and when the doctor watches it, other computational methods can be applied prior to the experts receiving the images. This will enhance the physician user experience while working without the need for them to wait for the algorithm to give the result at real time because it is already precomputed. Having a preliminary diagnosis can help reduce the workload and make the histologist more productive.

1.2 Why graphs?

This whole thesis was initially thought to be about lung tissue. Starting from a WSI we are interested in detecting which cells are tumoural and which cells are healthy. The purpose of the application is to rapidly detect tissue slices that contain a high amount of tumoural DNA so that it can be later on processed and analysed. Finding such WSI requires the histologist to look at several of them and deciding which one to choose. So we want to provide a ranking of images from more likely of having a high percentage of tumour to less likely. This way, on average, the physician would require to look at less images per patient, making it possible to analyse more patients' WSI in the same time. So, where does graphs come into play?

In a previous thesis, the same problem was tackled using a computer vision-only approach [28]. Initially, I was asked to improve on that method. After several months working on the problem I began to notice the principal flaw (which is also the principal feature) of convolutional neural networks: an inductive bias towards locality [6]. That means deep neural networks classify cells based on their immediate morphological properties. However, having met with pathologist Irene Sansano Valero, which is an expert histologist in the field of lung tumour, made it clear that cells were considered being tumoural or not depending on their surroundings. Visually identical cells may be classified differently if their neighbourhood is different. Here, I am always referring to lung tissue, we will later on analyse if this still holds true for other tissues. It was made clear then that a different approach was needed. Another kind of inductive bias was required, a relational inductive bias. One that classified cells based on their relationships with nearby cells and not only on their individual properties. This is the exact kind of inductive bias graph neural networks provide [5].

This is how the idea came into existence. The exact details of how the graph is constructed and which networks are employed are later discussed. But the key idea is that if relations between cells are considered important when classifying them, using graphs in some way may lead to better results. We expect this hypothesis to be specially true for lung tissue and, in fact, we come across high evidence that it is so. The question then is, is it true for other tissues? The answer is it depends. Apart from the lung dataset we repeat all the experiments for other three datasets obtaining mixed results. Sometimes it is a good fit, sometimes it is not and sometimes it works better but not because it is a graph but because it is a stacking classifier. The exact meaning of these words and which experiments are made to prove it will be later discussed.

Chapter 2

Problem Formulation and State Of The Art

2.1 Definition

In order to estimate the percentage of tumoural cells we decided to solve other problem. Instead of just predicting a number from a WSI, which would make the problem a logistic regression problem, we decided to segment and classify every cell to later count them and compute the percentage. There are two reasons why we chose to do so. The first one is because it makes models more interpretable, and the second one is because it makes the problem easier to solve.

Developing more on that first reason, we work in the medical field. Here, having a model that is statistically better than, say, a student is not enough to consider it really better than the student. The student can explain itself on why it made that diagnosis and so it provides more insight on how and why it makes mistakes. A seemingly black box model that simply outputs a number (the percentage of tumoural cells) gives no insight whatsoever of how and when you may expect it to fail. This is very dangerous in a medical setting. If we were at a factory classifying packages, we don't care so much about that. We can recover lost packages later on while still benefiting from the efficiency of using a automated process. However, we cannot recover dead patients. That makes interpretability a must. If our model predicts cells we can see which cells are causing more trouble. This helps know which data is needed to used for retraining the model and fixing those mistakes in the future. In the end, we want a model that can learn from its mistakes, looking at individual cell predictions makes it easier to recognise patterns in the mistakes it is doing and so it makes it easier to incrementally improve the model from its errors.

The second reason is that the regression problem is harder to solve. It may seem to be the other way around but it is not. Having worked with machine learning problems for years, my experience is that a regression problem where images are the input is almost never a good idea. Reframing the problem to solve something apparently harder in between has brought me better results in the past. Nonetheless, let's describe mathematically each problem in order to analyse each of them and come to an intuition of why it is a bad idea.

First, let's denote by $\mathcal{X}$ and $\mathcal{Y}$ the input and output space of the problem. We expect to find a function $f : \mathcal{X} \to \mathcal{Y}$ that effectively predicts the correct percentage given the WSI. Any WSI is no more than a collection of pixels, for that reason they can be viewed as very high dimensional vectors $\mathcal{X} = \mathbb{R}^N$, with $N > 10^9$ [27]. Similarly, the percentage is just a number between 0 and 1, so $\mathcal{Y} = [0, 1]$.

Now, the logistic regression approach consists of finding such function from a family of parametric models $\mathcal{F}_\theta = \{f_\theta | f_\theta : \mathcal{X} \to \mathcal{Y}\}$ while the segmentation approach tries to find f differently. It first divides the WSI into patches. Then, each patch is processed independently to obtain pixel-wise predictions which are then used to compute the percentage. So, the family of parametric models now in consideration is $\mathcal{G}_\theta = \{g_\theta | g_\theta : \mathcal{X}_s \to \mathcal{X}_s,\ \mathcal{X}_s \subset \mathcal{X}\}$ where $\mathcal{X}_s$ represents the space of images of 1024 by 1024 pixels, meaning $\dim(\mathcal{X}_s) \approx 3.1 \cdot 10^6$. Given any segmentation model g_θ we construct its regression counterpart f_θ as described in the following commutative diagram.

$$\begin{tikzcd}
\mathcal{X} \arrow[r, "f_\theta"] \arrow[d, "s"'] & \mathcal{Y} \\
\mathcal{X}_s^P \arrow[r, "\tilde{g}_\theta"] & \mathcal{X}_s^P \arrow[u, "c"']
\end{tikzcd}$$

Where we have extended g_θ to several patches by applying it independently to all of them.

$$\begin{aligned} \tilde{g}_\theta : \mathcal{X}_s^P &\to \mathcal{X}_s^P \\ (x_1, ..., x_P) &\mapsto (g_\theta(x_1), ..., g_\theta(x_P)) \end{aligned} \tag{2.1}$$

And the s and c functions refer to the split and count operations. Given a WSI it is split into several patches, for each patch, every cell is segmented and then all the tumoural and healthy cells are counted.

Without making any further assumptions about the families of functions we can derive some important insights about the advantages and disadvantages of each approach. First of all, it is clear that in the first approach the models have access to a wider context. Considering independent patches limits the ability to take into account global information. For instance, in lung tissue there is a structure called the cilium. It is a filamentous structure that appears near bronchioles. Its own existence is reason enough for considering the nearby cells as healthy. An example of such structure is depicted on Figure 4.15. The presence of cilium in the middle of a WSI cannot be taken into account when classifying the border of the tissue. But this disadvantage is not such a big deal. In a 1024 by 1024 image there is room enough for all the cells affected by the cilium, so limiting to such patches is enough. There is also another type of cell that interacts with their surrounding, the erythrocyte, also called red blood cell. All the cells glued to it are considered healthy. But that only applies to very close cells to it, so by using patches you will have no problem with this type of interactions. On the other hand, if we look at the dimensionality of input and output spaces we see a clear difference between the two approaches. For $\mathcal{F}_\theta$ the input space has a huge dimensionality while the output is unidimensional. In contrast, for $\mathcal{G}_\theta$ the input and output space both have the same dimensionality which is several orders of magnitude smaller than the dimensionality of $\mathcal{X}$. This makes the first approach quite prone to the curse of dimensionality [38][1]. Even worse, we have just one number per each WSI. Even a simple regression problem needs more than 30 samples to have a reasonable amount of uncertainty. Having such amount of WSI is a very difficult task.

One may also think that the regression problem can be split into patches too. That is, solving the problem for individual patches and then averaging the percentages. This way the difference in dimensionality of the input and output spaces is lower. Moreover, the data scarcity problem is solved. But it is not so easy. In my experience, even with 224 by 224 images, regression is quite unfeasible if done in a naive way. On the other hand, for the individual percentages of each patch to be enough we will need to label all the patches of a WSI. Otherwise the sampling may not be uniform enough since WSI are very different from one part of the image to another, in other words, they are highly non-stationary. Labelling all the patches of one WSI could mean labelling thousands of images, which is unfeasible.

All in all, we chose this way because we believed it was going to give better results and because the doctors preferred that solution over the other for its interpretability.

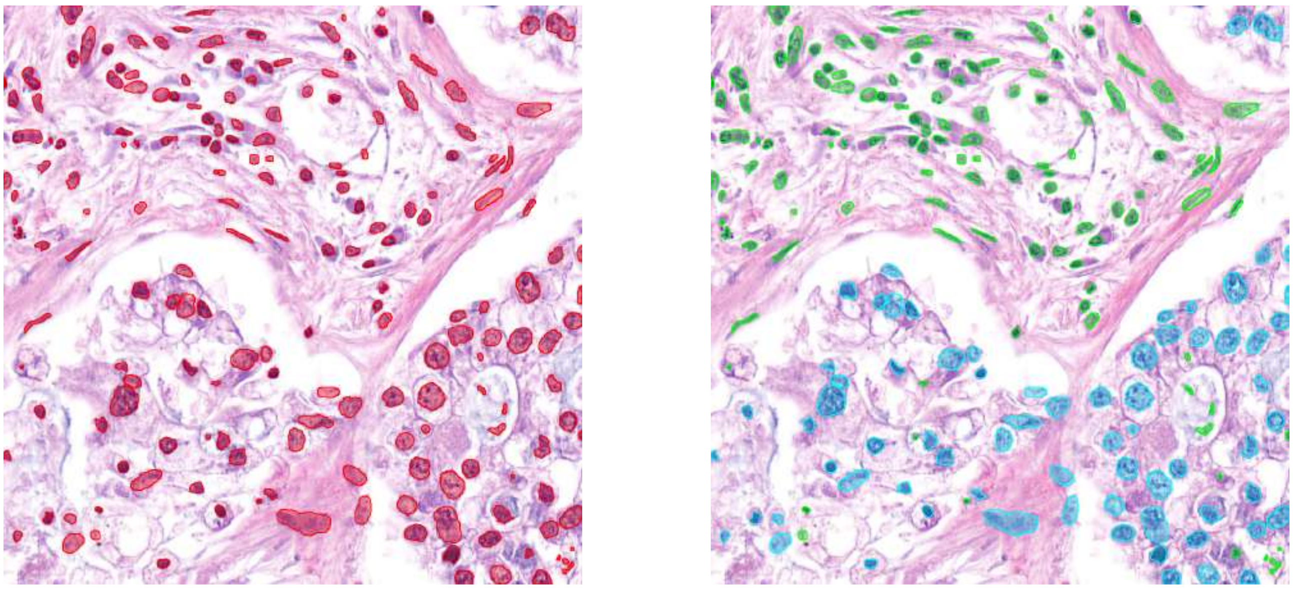
[1]Notice that this is not true for other regression problems, NeRFs [24] need to increase the dimensionality of the input to work properly.

2.2 Data

Now that we have described the problem as a segmentation and classification one, let's describe the datasets in which it is going to be applied and how they were obtained. The first one and the main focus of the thesis is the DigiPatics lung dataset, it is called like this because it was created under the DigiPatics project [33] and contains patches of lung WSI. Then, I'll describe another dataset also focused on tumour detection but of another organ, the DigiPatics breast dataset. And at the end I'll explain two more dataset: CoNSeP [12] and MoNuSAC [37]. They are two public datasets related to digital pathology and nuclei segmentation and classification as well.

2.2.1 DigiPatics lung dataset

This dataset contains 85 images of 1024 by 1024 pixels referring to Hematoxilin and Eoxin stained WSI. Every patch comes from a WSI of a patient with lung tumour and it is annotated pixel-wise, that is, every pixel is either 0, 1 or 2 depending on whether it belongs to the background, to a healthy cell or to a tumoural cell. To have a better understanding of the problem at hand, look at Figure 2.1 below.



(a) Segmentation (b) Classification

Figure 2.1: Visualisation of the kind of labels that are needed. For each cell it is required to have its contour, as seen in the left, together with their corresponding class as seen in the right. Blue means tumoural and green non-tumoural.

In order to annotate those images it is first needed to choose which images to label. One can extract hundreds of patches from surgical biopsies made by thoracotomy, and even in needle biopsies it is possible to obtain more than two hundred patches. But labelling those patches is very expensive and so they have to be chosen carefully. One important aspect to take into account is the number of patients at consideration. One thousand labelled images are of no use if they come from only one patient. There is a huge variance across WSI of different patients. For that reason we decided to annotate images from nine different patients. The selection within patient was made based on structures that seem very different from each other so that the dataset captures as much variance as possible.

Once the images were chosen, the annotation can begin. However, labelling manually each image can take days for each image. That is why we followed a semi-automated approach. To alleviate the amount of work required, an iterative procedure was designed. In a first step made by David Anglada and Feliu Formosa, the max-tree [31] was used to create rough segmentations of 24 patches that were later reviewed and improved by the students. Those initial labels were used to train Hovernet [12], explained in detail in section 2.3. Using that newly trained model, 20 more images were annotated and reviewed by me. Then, Hovernet was trained again using those 44 images and used to infer the GT of 41 more patches. Only after carefully correcting all the 85 patches, would the pathologist Irene Sansano Valero start reviewing the dataset. The whole process took around 100 hours of human labour, 20 hours of GPU computation and 15 hours of expert human labour.

As a side note, this dataset is enough for the research carried out here but is definitely not enough for a production setting. Nine patients is by no means a representative sample. And there are many structures not present in the patches selected. We made our best effort to annotate a dataset that would validate our approach. Having proved its validity it remains to extend the dataset and scale the models in order to achieve a production ready model.

2.2.2 DigiPatics breast dataset

Another tumour related dataset is provided by the DigiPatics group. This one was annotated by David Anglada and supervised by pathologist Teresa Soler. It contains 141 images from 4 different patients that had breast cancer. The biopsies were stained with HER2 staining. This time the dataset contains 6 different classes counting the background. I will not dive into much details of this dataset and what represents every class, the only fact I would like to mention is that domain experts on this field and the engineers collaborating with them do not think that group structure is as important here as it is in the case of lung tumours. In fact, experiments described in subsection 3.4.2 and subsection 3.4.3 show that this is the case. In Figure 2.2 you can see an example of the type of images that are in this dataset.

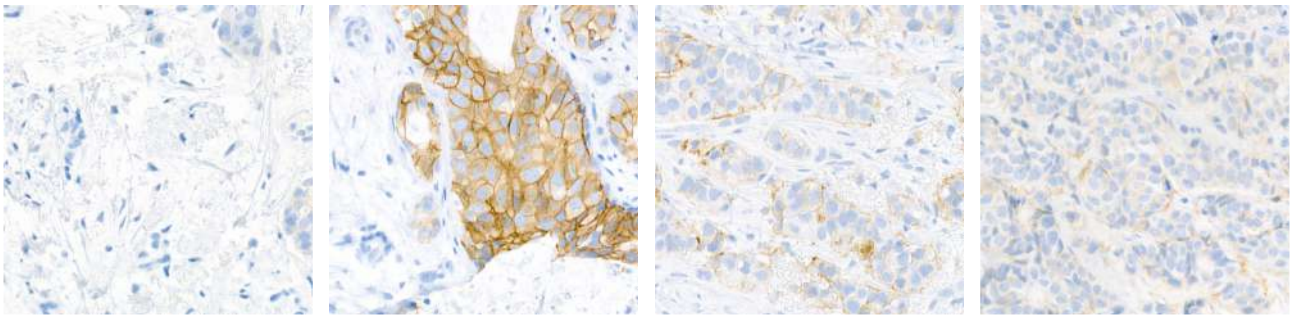

(a) Patient 1 (b) Patient 2 (c) Patient 3 (d) Patient 4

Figure 2.2: Patches of the four patients that are in this database. Names are omitted for data privacy. As you can observe, images are visually very different from patient to patient even though the same staining is employed in all of them.

2.2.3 CoNSeP dataset

The name means colorectal nuclear segmentation and phenotypes. In other words, it is about cell nuclei detection on colorectal adenocarcinoma WSIs. Yet another organ in which we are interested of automatically visualise the cells that are related to a tumour. It uses the same staining as the DigiPatics lung dataset: H&E. The size of the images is similar: 1000x1000. And it also has many patients: 16 in total. The main difference with the other datasets so far is that it only contains 41 images. Another difference with the lung dataset is that is has 7 classes, without counting the background. Instead of classifying cells as tumour vs non-tumour the authors here decided to provide a more detailed analysis of each cell. In Figure 2.3 you have an example of what are the images it contains. We decided to use this dataset because it was the one used in the Hovernet article although we don't have too much insight on this type of data. The results of the experiments on this dataset seem to prove that graphs neural network are not a good choice for colorectal tissue. Nonetheless, more research is needed to confirm that hypothesis. It remains to ask experts if the group structure is important here and other approaches need to be carried out, like binarising the classes. Since in this thesis I could only talk with lung experts, I couldn't carry those experiments myself. It is left as future research.

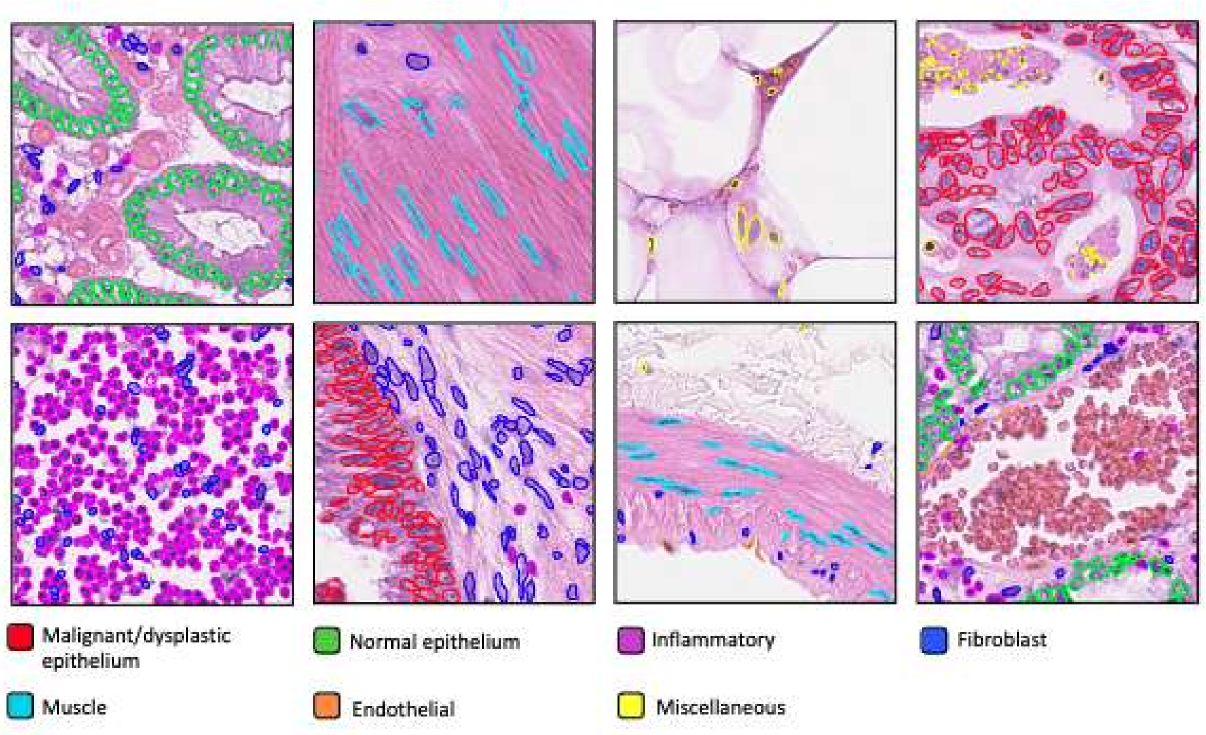


Figure 2.3: Different examples of the CoNSeP dataset illustrating all the different cells that it contains. Image was taken from the original article [12].

2.2.4 MoNuSAC dataset

As in previous subsection, the abbreviation has some meaning. It comes from Multi-Organ Nuclei Segmentation and Classification Challenge. This time it is not tumour related but it is still a nuclei segmentation and classification problem so we deemed it interesting to try our method here. The MoNuSAC database contains 294 images from 71 patients. There are several differences with respect to the other three datasets. The first one is the type of classes it has. They are not tumour related, instead cells are classified into four types: epithelial, lymphocite, macrophage and neutrophil. The second difference is the number of organs. In the previous datasets only one tumour was considered at a time. Here we have lung, breast, kidney and prostate. The third and last difference is that the WSI came from 37 hospitals instead of just one. Every WSI was selected from The Cancer Genome Data Portal[2] and were later labelled. Having such amount of data made this dataset quite interesting to use. It has more variety and many more images than all the datasets previously mentioned. Interestingly enough, using graph neural networks here gives better results than not using them. Experimental results will

[2]https://portal.gdc.cancer.gov/ Accessed 15th May 2023

be discussed in more detailed on section 4.1. One technical detail to remark is that images from this dataset have very different aspect ratios and sizes. To simplify the process we just upsampled or downsampled every image into having size 1024x1024 using Lanczos interpolation. An example of the images from this dataset is in Figure 2.4.

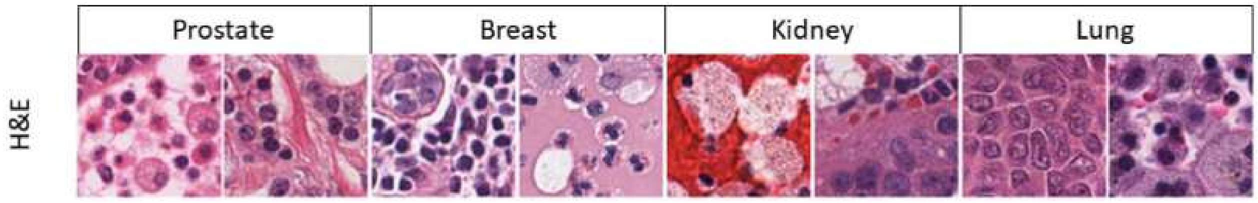


Figure 2.4: Image extracted from the MoNuSAC article [37]. It depicts various images from each of the four organs used in the creation of it.

2.3 Computer vision algorithms

In this section I will provide a brief survey about the state of the art in segmentations problems and a detailed explanation of Hovernet [12] which is the model used in the first phase of our method. Prior to diving into specific neural network architectures I will make a quick recap into what a convolutional neural network is. The forward pass of a single neuron from a layer of a neural network can be described as

$$a_j = f(\mathbf{w}^{(j)}\mathbf{x} + b_j) = f\Big(\big(\sum_{i=1}^{n} w_i^{(j)} x_i\big) + b_j)\Big) \tag{2.2}$$

where w^j is the j-th row of the weight matrix of that layer W, x_i is the i-th entry of the input and b_j is the j-th entry of the bias of that layer[3]. Here f is any non-linearity to give the network expressivity. Now, a convolutional layer is similar in that it also has neurons that are made of a linear operator applied to the input plus a bias and a non-linearity. However, the operator is different. For a normal neural network, also called multilayer perceptron, the lineal operator can be expressed as $(W \cdot \mathbf{x})^{(j)}$ being $\cdot$ the matrix multiplication and $(\cdot)^{(j)}$ denotes the j-th entry of the vector. For a convolutional neural network, the operator is the convolution $(W * \mathbf{x})^{(j)}$. In one dimension, if $W = (w_1, w_2, w_3)^T$ is a kernel of dimension 3, then the convolution can be expressed as a matrix multiplication like so[4]

$$W * \mathbf{x} = \begin{bmatrix} w_1 & w_2 & w_3 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & w_1 & w_2 & w_3 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_k \\ \vdots \\ x_n \end{bmatrix} \tag{2.3}$$

To extend it to 2D data like images, we could still use this matrix definition but it would get quite cumbersome so, instead, a simple formula can be used to describe 2D convolutions.

[3]https://atcold.github.io/pytorch-Deep-Learning/en/week02/02-3/ Accessed 15th May 2023
[4]https://atcold.github.io/pytorch-Deep-Learning/en/week04/04-1/ Accessed 15th May 2023

$$(W * \mathbf{x})^{(m,n)} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} w^{(m-i,n-j)} \cdot x^{(i,j)} \tag{2.4}$$

where the superindex notation denotes pixel coordinates and the weight matrix is centred at $(0, 0)$ and has compact support, same as the image, which also has compact support meaning it is only different than zero for a finite amount of coordinates. As you may have noticed, convolutions return images as output. Therefore, we can visualise what a convolutional neural network is doing by looking at the neuron activations, which in this context are called the channels instead of neurons. In Figure 2.5 there is an example of two possible filters that may be learned by a convolutional neural network.

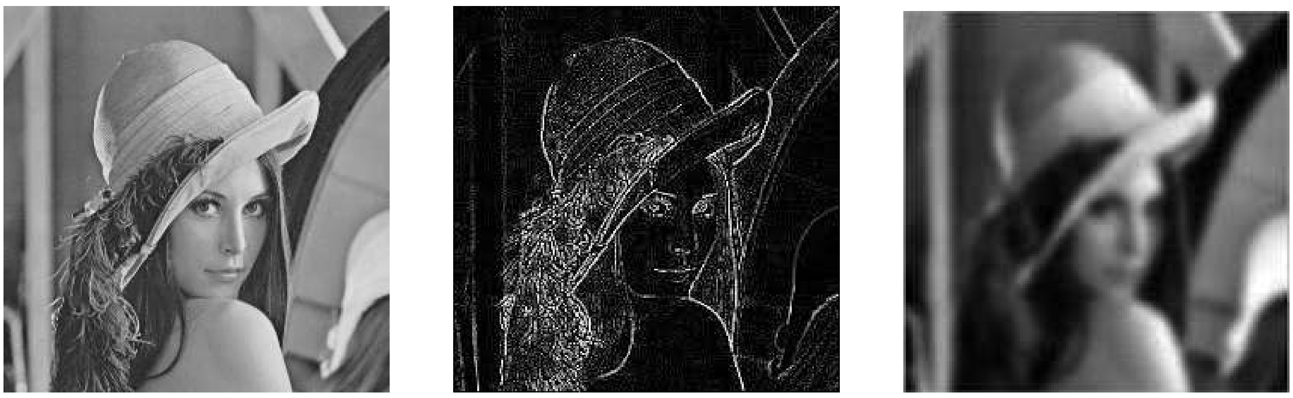

(a) Original (b) Laplacian (c) Gaussian

Figure 2.5: Example of convolution filters. At the left we have a very classical image, called Lena. On the center there is the laplacian filter applied to it using a convolution and on the right there is the gaussian filter applied to it.

A single convolution is not expressive enough to solve segmentation problems. Normally, several layers are required, but having too many layers has the problem of vanishing and exploding gradient which is why residual connections are needed. That was one of the ideas behind the first deep learning attempt at biomedical segmentation made by Olaf Ronneberger et al. [30]. They proposed an encoder-decoder architecture as shown in Figure 2.6.

That architecture was improved recently with the development of transformers [35]. In 2021 Jieneng Chen et al. invented TransUNet [7] and later on in 2022 Jeya Maria Jose Valanarasu et al. created UNeXt [34]. I will not dive into the specifics of those architectures but rather comment on their limitations and why we couldn't use them. The key limitation is their sample efficiency. Even though transformers are more sample efficient in reinforcement learning than previous deep learning methods [23], they still require a fair amount of data. In Jeya Maria Jose Valanarasu et al. [34] they used two datasets of 2594 and 647 images respectively. That is at least an order of magnitude more than what we could obtain. The other option, TransUNet [7], was tried in a dataset with 3779 computer tomography (CT) images, yet too much for us. Apart from that, the problem they tackled was organ segmentation, which is quite different from cell segmentation. In order to achieve better sample efficiency, a method with specific inductive biases is needed.

As explained in section 2.2, the max-tree [31] can give good results at detecting cell contours while using no data at all. The algorithm used morphological properties of the cells to distinguish them from the background. That set a precedent, morphological algorithms could help us reduce the amount of data needed. Another important morphological algorithm is the watershed [22]. It is known for creating accurate contours if the energy landscape is properly

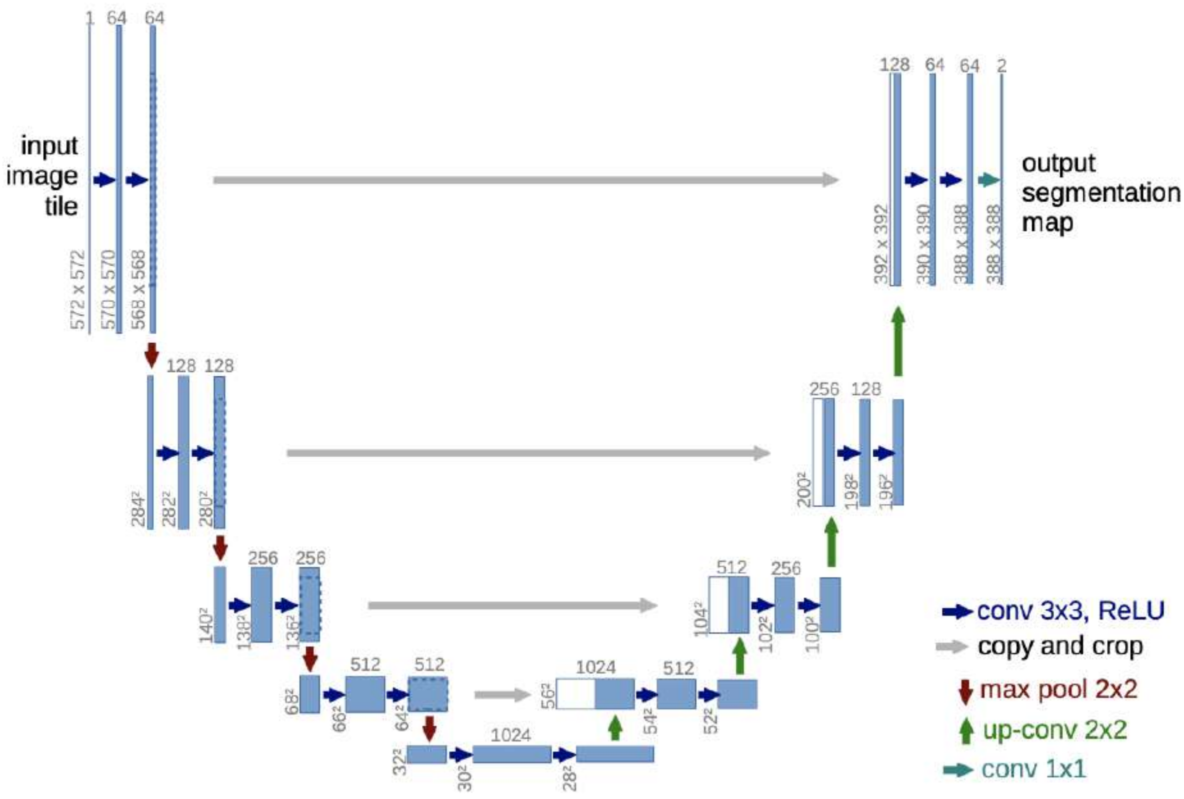


Figure 2.6: Original U-net architecture. It was composed by convolutional layers, pooling layers and residual connections.

defined. Hovernet [12] combines both the U-net architecture with the watershed algorithm to produce cell segmentations and classify them. An overview is on Figure 2.7.

Hovernet employs the same encoder-decoder architecture as U-net but it combines three different decoders with only one shared encoder. Each of the three decoders is trained to infer a different property from the GT. The NP branch separates the cells from the background, ignoring their class. The HV branch predicts horizontal and vertical distances from each pixel to the nuclei of the nearest cell. And the TP branch predicts the GT as is. Everything is trained end-to-end under one single loss function, which is shown below

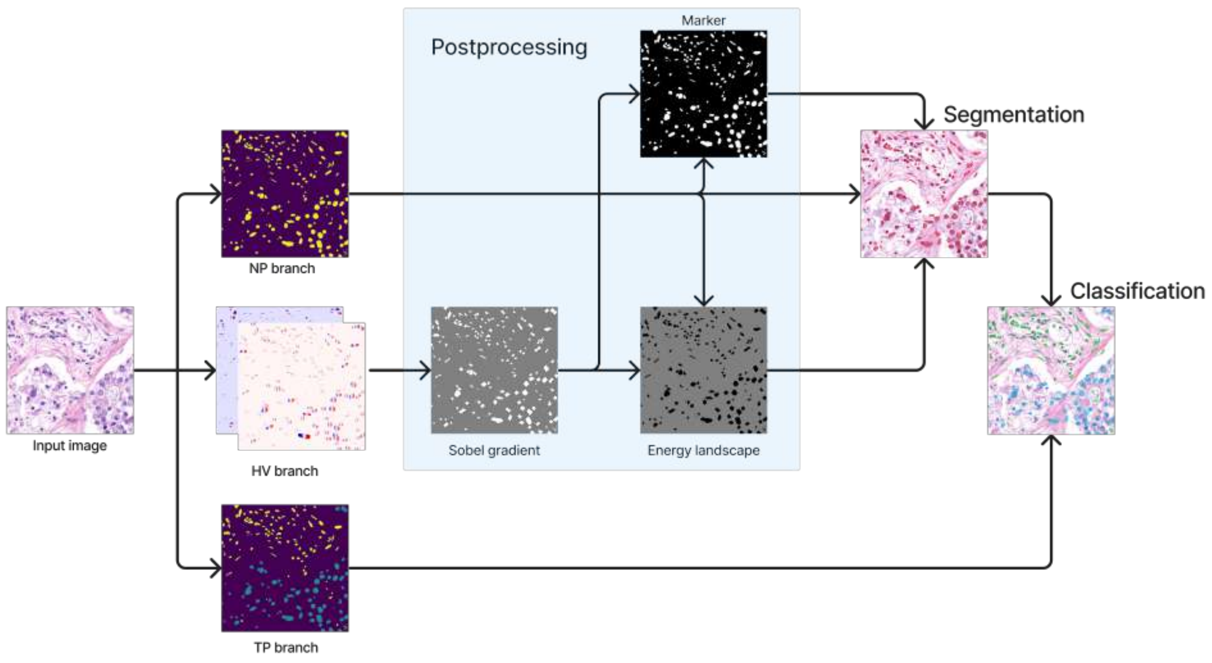


Figure 2.7: Overview of the Hovernet method. It has three branches that predict different maps derived from the GT. In a post-processing step all the maps are combined using the watershed algorithm with carefully designed energy landscapes and markers.

$$\begin{aligned}
\mathcal{L} &= \lambda_{mse}^{HV}\mathcal{L}_{mse}^{HV} + \lambda_{msge}^{HV}\mathcal{L}_{msge}^{HV} + \lambda_{bce}^{NP}\mathcal{L}_{bce}^{NP} + \lambda_{dice}^{NP}\mathcal{L}_{dice}^{NP} + \lambda_{bce}^{TP}\mathcal{L}_{bce}^{TP} + \lambda_{dice}^{TP}\mathcal{L}_{dice}^{TP} && (2.5)\\
&= \frac{\lambda_{mse}^{HV}}{B}\left(\sum_{i=0}^{B}\|H(x_i)-h_i\|_2^2 + \sum_{i=0}^{B}\|V(x_i)-v_i\|_2^2\right) && (2.6)\\
&+ \frac{\lambda_{msge}^{HV}}{B}\left(\sum_{i=0}^{B}\|\nabla H(x_i)-gh_i\|_2^2 + \sum_{i=0}^{B}\|\nabla V(x_i)-gv_i\|_2^2\right) && (2.7)\\
&+ \frac{\lambda_{bce}^{NP}}{B}\sum_{i=0}^{B}\sum_{j=0}^{D}(y_i^{NP})_j\log(NP(x_i)_j) && (2.8)\\
&+ \frac{\lambda_{dice}^{NP}}{B}\sum_{i=0}^{B}\left(1-\frac{\sum_{j=0}^{D}NP(x_i)_j(y_i^{NP})_j}{\sum_{j=0}^{D}(NP(x_i)_j)^2+\sum_{j=0}^{D}((y_i^{NP})_j)^2}\right) && (2.9)\\
&+ \frac{\lambda_{bce}^{TP}}{B}\sum_{i=0}^{B}\sum_{j=0}^{D}(y_i^{TP})_j\log(TP(x_i)_j) && (2.10)\\
&+ \frac{\lambda_{dice}^{TP}}{B}\sum_{i=0}^{B}\left(1-\frac{\sum_{j=0}^{D}TP(x_i)_j(y_i^{TP})_j}{\sum_{j=0}^{D}(TP(x_i)_j)^2+\sum_{j=0}^{D}((y_i^{TP})_j)^2}\right) && (2.11)
\end{aligned}$$

where all the λ are hyperparameters, the letter y denotes GT in any form, h and v are horizontal and vertical GT maps, gh and gv are the gradients of horizontal and vertical maps, D is the number of pixels in any image, B is the batch size, $\|\cdot\|_2^2$ is the L_2 norm, $H(\cdot)$ and $V(\cdot)$ are the outputs of the HV branch, $NP(\cdot)$ the output of the NP branch and $TP(\cdot)$ the output of the TP branch. This loss is particularly interesting because it combines multiple ideas. It is a mix of classification, regression and segmentation losses. Moreover, it can be considered as a second order optimisation since it is using the mean square error over the gradients. On the other side it combines the cross-entropy which is designed specially for classification problems while optimising the Dice loss which is more or less like maximising the intersection of predicted and real cells, more on that on subsection 3.3.9. It is expected that optimising all the different objectives while using a single encoder is going to make that encoder extract features useful for a variety of tasks, thus generalising better.

After the model is trained, it can be used for inference in addition with a post-processing phase which consists of the watershed algorithm. This particular watershed requires an energy landscape which defines the space where the flooding is made and a marker that contains the starting points to start the flooding. Both are defined below

$$E = (1 - \mathcal{S}_m(\mathbf{X})) \odot NP(\mathbf{X}) \tag{2.12}$$

$$M = \mathrm{ReLU}(NP(\mathbf{X}) - \mathcal{S}_m(\mathbf{X})) \tag{2.13}$$

being E the energy and M the marker. In those equations $\mathbf{X}$ refers to the input image, ReLU is the rectified linear unit [2], $\odot$ is the element-wise multiplication, also referred to as Hadamard product, and $\mathcal{S}_m(\mathbf{X})$ is the thresholded gradient of the HV branch as expressed here

$$\mathcal{S}_m(\mathbf{X}) = \max(S_x * H(\mathbf{X}), S_y * V(\mathbf{X})) \tag{2.14}$$

where S_x and S_y are Sobel filters [32] and $*$ is the convolution operation. The whole process can be visualised in Figure 2.7. The reason for using gradient filters over the HV branch is that if the prediction is perfect, then such gradients are exactly the NP branch. Therefore, if we apply a watershed of one over the other it is expected that one branch fills the errors of the other. That is not so simple in practice, and for this post processing algorithm to work both the HV branch and the NP branch need to be expanded and contracted using a thresholding function over a threshold that was carefully selected by the authors based on empirical results.

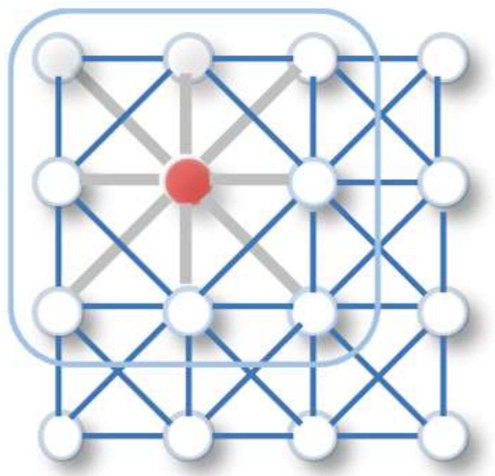
2.4 Graph neural networks

Having described the state of the art for computer vision, let's introduce the state of the art of graph neural networks as well. Graph neural networks are very similar to neural networks but the key difference is that the computational graph is different for every node and it varies depending on which nodes it is connected to. GNNs as used in this thesis generate an embedding which decodes all the relevant information of that node. Most of the techniques used with neural networks can be extended to GNNs as well, like dropout [13], batch normalisation [14] or pooling layers [41]. In fact, there are more than 50 different possible architectures [41] and more than 300.000 possible configurations [42]. However I will focus mainly on two of the most popular layers: graph convolution and graph attention. Both can be used for node classification which is what we are interested about since we are going to treat cells as nodes. More on that description in section 3.1.

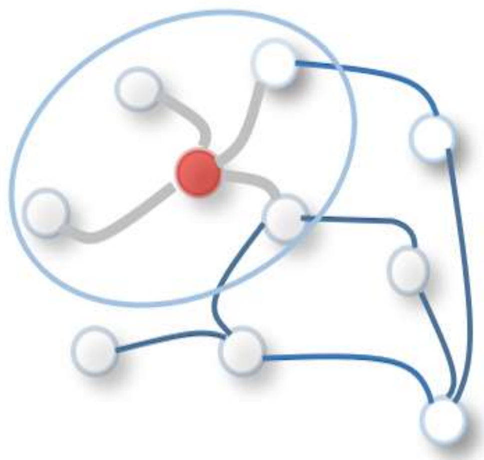
2.4.1 Graph convolution

This architecture was proposed by Thomas N. Kipf et al. [15] in 2016 and has been cited almost ten thousands times as of this date. The main idea is to adapt the notion of convolution from images to graphs. An illustration of the concept can be seen in Figure 2.8.

Mathematically, the graph convolution operation as expressed in [15] can be defined as shown below



(a) 2D Convolution where each pixel can be considered a node connected to all its adjacent pixels. This operation returns the weighted average of adjacent pixels for each node.



(b) Graph Convolution where the weighted average is taken with respect to adjacent nodes. There is no notion of pixels and each node has no absolute spatial coordinates.

Figure 2.8: Visualisation of 2D Convolution vs Graph Convolution taken from [41].

$$\mathbf{h}_j^{(l+1)} = \sigma\left(\mathbf{b}^{(l)} + \sum_{k\in\mathcal{N}_j} \frac{1}{c_{jk}} \mathbf{W}^{(l)}\mathbf{h}_k^{(l)}\right) \tag{2.15}$$

where $\mathbf{b}^{(l)} \in \mathbb{R}^d, \mathbf{W}^{(l)} \in \mathbb{R}^{d\times d}$ are the bias and weights of the layer, $\mathcal{N}_j$ is the set of neighbours of node j, $c_{jk} = \sqrt{|\mathcal{N}_j|\cdot|\mathcal{N}_k|}$ is a normalisation factor and σ is an activation function. The vectors $\mathbf{h}_k^{(l)}$ are the hidden embeddings of the network for each layer, being $\mathbf{h}_k^{(0)}$ an initial vector containing any relevant information about the node. That information can be the area of the cell, the average colour, or even a prior distribution for the class label. In the last layer, the weight matrix is of dimensions $C \times d$, where C is the number of classes or 1 if $C = 2$ and the activation function is either the sigmoid for a binary problem or the softmax [11] for a multi-class problem.

2.4.2 Graph attention

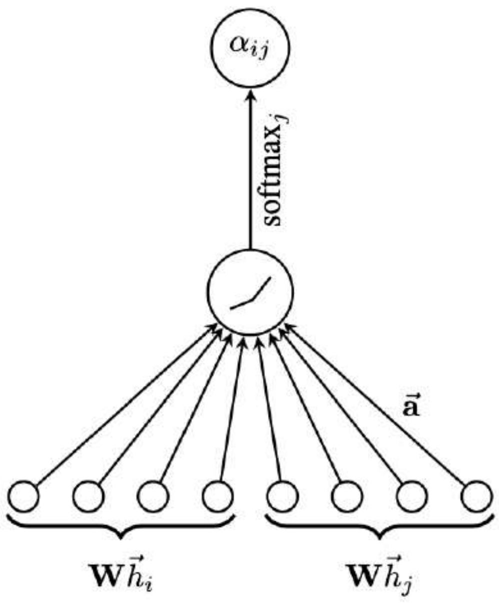
As an improvement over simply doing the average, one year after the publication of the graph convolution, Petar Veličković et al. [36] proposed the idea of including the attention mechanism [3] to compute a weighted average instead. This idea, which has been cited over eight thousands times, is visualised in Figure 2.9.

More formally, the computation can be described as follows

$$\mathbf{h}_j^{(l+1)} = \sigma\left(\sum_{k\in\mathcal{N}_j} \alpha_{jk}\mathbf{W}^{(l)}\mathbf{h}_k^{(l)}\right) \tag{2.16}$$

where $\mathbf{W}^l \in \mathbb{R}^{d\times d}$ are the layer weights and $\alpha_{jk} \in \mathbb{R}$ are the attention weights which are defined by the following formula

$$\alpha_{jk} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}\cdot[\mathbf{W}\mathbf{h}_j||\mathbf{W}\mathbf{h}_k]))}{\sum_{r\in\mathcal{N}_j}\exp(\text{LeakyReLU}(\mathbf{a}\cdot[\mathbf{W}\mathbf{h}_j||\mathbf{W}\mathbf{h}_r]))} \tag{2.17}$$



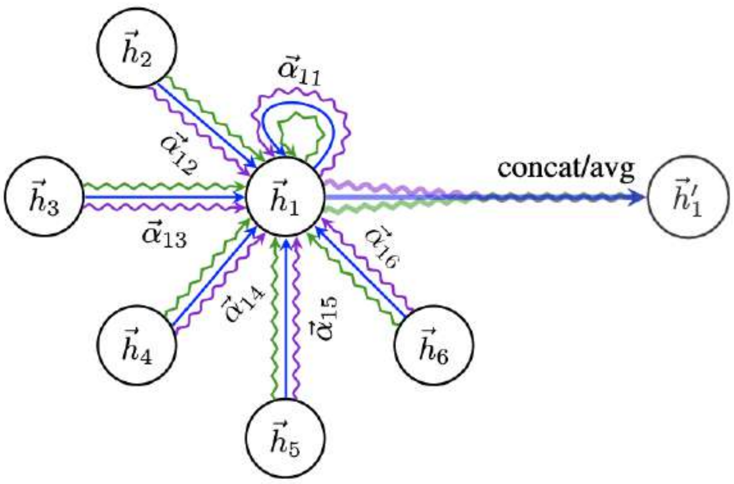


Figure 2.9: On the left is the overview of the computation of the attention weights. For each adjacent node an attention weight is computed based on the similarity of their embeddings. On the right there is a visualisation about multi-head attention, which consists of concatenating the result of several attention mechanisms. The figures are taken from the original article [36].

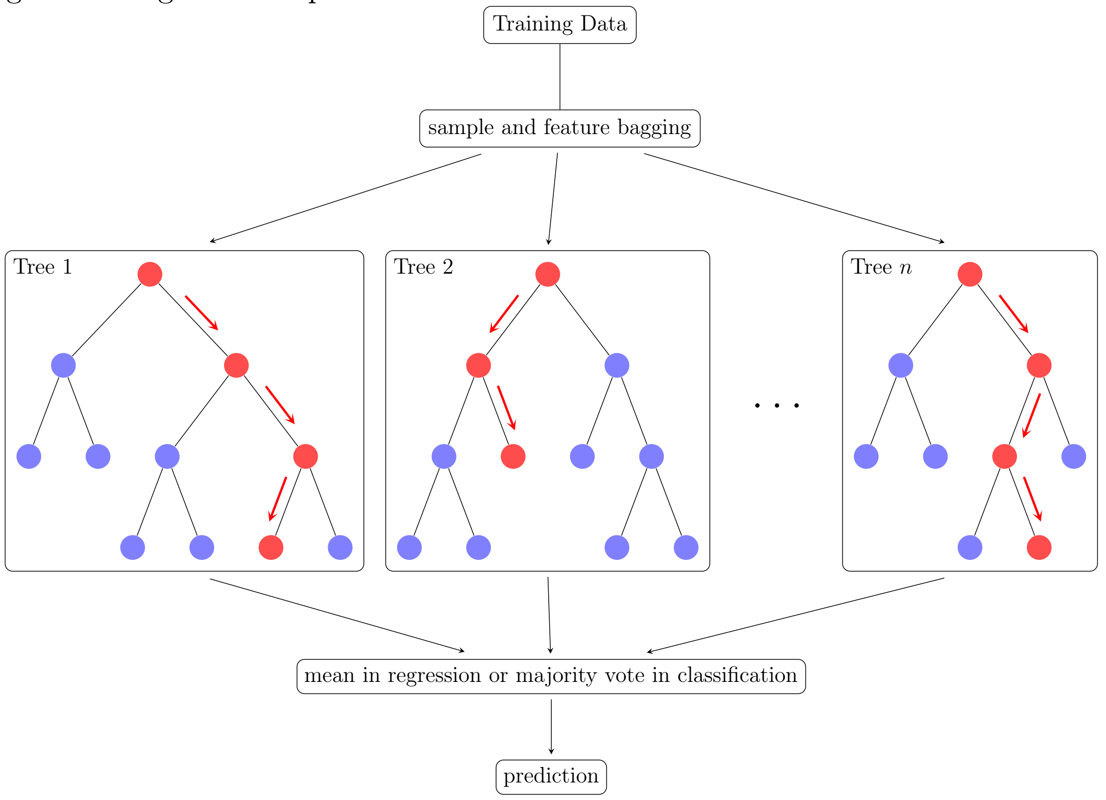
being $\mathbf{a} \in \mathbb{R}^{2d'}, \mathbf{W} \in \mathbb{R}^{d' \times d}$ two learnable projection matrices, LeakyReLU is the leaky rectified linear unit [19] and $||$ the concatenation operation. Inspired by the multi-head attention mechanism proposed in [35], the previous attention mechanism can be extended to H heads

$$\mathbf{h}_j^{(l+1)} = \Big\Vert_{h=1}^{H} \sigma \left(\sum_{k \in \mathcal{N}_j} \alpha_{jkh} \mathbf{W}_h^{(l)} \mathbf{h}_k^{(l)} \right) \tag{2.18}$$

where now $\mathbf{W}_h^{(l)} \in \mathbb{R}^{d \times Hd}$, the attention weights are different for each head and sum up to one in each head $\sum_{k \in \mathcal{N}_j} \alpha_{jkh} = 1, \ \forall h$ and in the final layer heads are averaged instead of concatenated as explained in [36]. Notice that there is not bias as with the convolution. This is because the attention scores are thought to be similarities between nodes. The attention mechanism is just a way of averaging embeddings based on their similarity to the target node. There is no need for bias under that interpretation.

2.5 XGBoost

The last section of this chapter is devoted to give a very brief explanation of an algorithm that is going to be tangentially used in the experiments as a way to perform an ablation study to see if the graphs are really being useful or not. XGBoost [8] is an algorithm to perform either regression or classification over tabular data. In our case, it is going to be used over the extracted features of each cell (perimeter, area, ...) to predict its class. The XGBoost algorithm is in fact a gradient boosted tree, the distinction here comes because XGBoost is a faster implementation of such idea. To understand what a gradient boosted tree we first need to explain what a random forest is. It is a set of tree classifiers that are ensembled together using the average of their predictions. A visualisation of it is here below[5].


Training Data
sample and feature bagging
Tree 1
Tree 2
Tree n
. . .
mean in regression or majority vote in classification
prediction

Now, to pass from a random forest to a gradient boosted tree we have to substitute the idea of bagging (doing the average) to boosting. Boosting consists of incrementally add models to the ensemble that each of them predicts the error of the previous ensemble. This way every model you add operates on a smaller target variable. For classification problems, the residual is taken from the log likelihood so that it is treated as a regression problem over the real line. The process is represented on Figure 2.10[6].

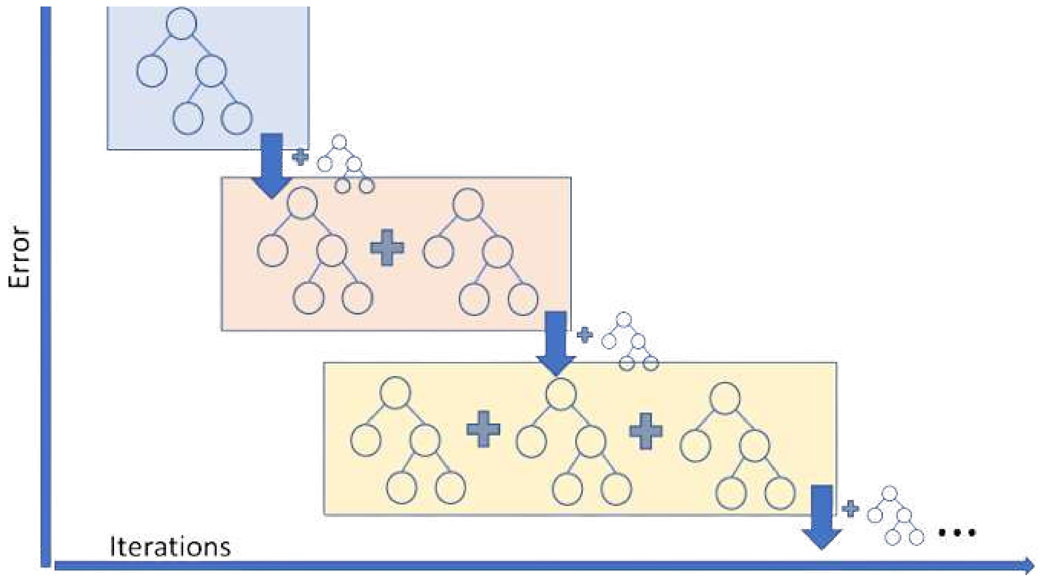


Figure 2.10: Visualisation of the concept of boosting for tree models. Image taken from a medium post.

[5]http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/ Accessed 25th January 2023

[6]Image taken from https://medium.com/analytics-vidhya/what-is-gradient-boosting-how-is-it-different-from-ada-boost-2d5ff5767cb2 Accessed 15th May 2023

Chapter 3

Problem Solving

Having described the problem at hand as well as relevant deep learning methods it is time to join it all into one single method. In this chapter we will cover a detailed explanation of the method proposed, the metrics that will be used to compare across architectures and hyperparameters and the experiments carried out to show its usefulness.

3.1 Method description

I have described how the cell classification problem was tackled with the help of convolutional neural networks. I have also explained two algorithms used for node classification. It is time to merge both fields. For that, we need to describe the cell classification problem as a node classification problem. What are going to be our nodes? The individual cells extracted from Hovernet. Using that computer algorithm we are going to infer the location of the cells in a patch by considering the contours in the predicted segmentation. Those contours are used just to extract what we call morphological features of the cell and its coordinates in the image. It is left to define the edges. We are going to consider two nodes (cells) to be related if they are sufficiently close. By sufficiently close it is meant that their euclidean distance is less than some previously defined amount. Apart from that, to have manageable graphs, the degree of each node is limited by only considering a small amount of nearby nodes as possible connections. This way we ensure the number of edges increases linearly with the number of nodes making our method more scalable. An example of such graph is on Figure 3.1.

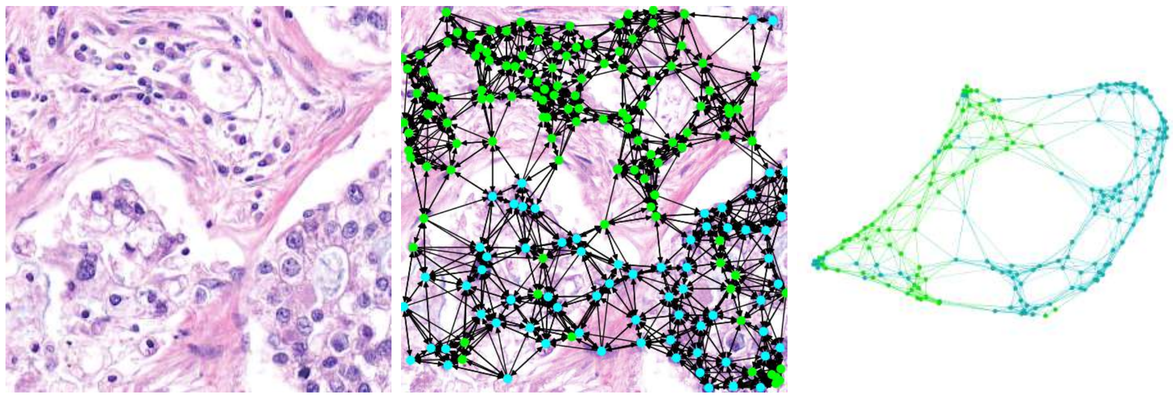


Figure 3.1: Example of an image and its associated graph. At the middle we have the nodes located at their corresponding centroids. However, a graph is an abstraction, so it may also be viewed as in the right image, since it only encodes relationships, not absolute positions. The visualization of the graph was made using Gephi [4].

Given nodes and edges we can still give more information to the graph neural network. In subsection 2.4.1 we showed that the network can be given an initial set of features $\mathbf{h}_k^{(0)}$. Those features can be anything that gives information about the cell. We decided to use the following set of descriptive features:

- The area and perimeter of the cell measured in pixels. Those values should give information about the shape of the cell.

- The standard deviation of the values of the pixels in gray format. This magnitude is supposed to bring insights about the luminosity of the cell.
- The histogram of the red, green and blue colour channels. We expect it to summarise the information about the colour of the cell.
- A prior distribution of the class. It is computed using the output of Hovernet. Each class probability is inferred as the number of pixels of that class predicted by Hovernet divided by the total number of pixels of the cell.

Later on, in section 3.4 an experiment is described to discover how relevant the selected features are.

3.2 Hyperparameter tuning

In our first step, Hovernet, we did not try to optimize any hyperparameter for two reasons. The first, the compute power needed is simply prohibitive. One configuration requires several hours to train. Doing cross-validation on it or trying more than 10 configurations is going to last for weeks for every dataset involved. The other reason is that the method is quite stable. Looking at the loss function during training for the train and validation dataset, we observed that after a few tens of epochs the curve converged for both datasets. So it seems to indicate that changing hyperparameters is not going to give a significant difference for the Hovernet model.

For the graph networks we will be tuning 4 variables. The first one is the number of layers. We fixed the number of neurons on its layer to 100 and try architectures from 1 layer up to 15 layers. The second hyperparameter we tune is the dropout rate. We apply dropout after every graph layer. The third hyperparameter is batch normalisation. We consider models with and without it. And the last hyperparameter is the type of the graph layer, either convolution or attention. Having defined those, we perform a grid search over them. We train all the different configurations and evaluate them in a validation dataset. The configuration that gives the best validation score is then evaluated on the test set, and those are the metrics that are reported. Since we are optimising the type of graph layer used, I will refer to the model as GNN and not as GCN (convolution) of GAT (attention) since it could be any of those.

In the case of XGBoost the tuning is done a bit differently. We optimize over the learning rate, the maximum depth of each tree and the percentage of features visible to each tree. The number of trees is fixed at 500 for every configuration. Since XGBoost is so efficient we could perform cross validation to estimate the validation score. Concretely, we used 10 fold cross validation. Then, the configuration with the best cross validation score was tested on the test set, and those are the metrics reported. For a more detailed analysis of which combination of hyperparameters was the optimal in each case, please refer to the Appendix E.

3.3 Evaluation metrics

This section is going to provide a review of the most common metrics for any classification problem, either binary of multiclass.

3.3.1 Confusion Matrix

Prior to defining any metric we have to define the concept of confusion matrix. Most of the metrics described can be expressed in terms of it. The confusion matrix is a way of measuring how precise is any method. For a binary classification problem one has positive (1) and negative (0) classes. If the model correctly predicts the positive or negative class it is called true positive and true negative. Then, if the model incorrectly predicts positive it is called false positive and if it infers negative wrongly it is denoted by false negative. Confusion matrices are typically expressed as shown below.

Table 3.1: Binary confusion matrix.

		Predicted	
		Positive	Negative
Actual	Positive	TP	FP
	Negative	FN	TN

On our specific problem the true positives are the tumoural cells that are correctly predicted as tumoural, true negatives are healthy cells rightly classified as so and false positives or negatives are misclassifications of cells in one way or another. However, there is one nuance needed to take into account. It may be possible that one cell is not predicted or that the model predicts more cells than there really are. For that reason we will only be considering those cells that are in the ground truth and in the prediction at the same time. This means that it is needed to create a 1-1 correspondence between real and predicted cells. This correspondence is created by distance. Two cells are considered a pairing if the former is the closest to the latter, and the latter is the closest to the former. Ignoring cells that do not belong to such pairings is not a problem because we are interested in improving the classification of already predicted cells, not on improving the segmentation itself. Nonetheless, we can also evaluate the performance on the missing cells by including the background as a new class.

To deal with such multiclass scenarios an adaptation is needed. Instead of adding more rows and columns the matrix is built considering one class against all the others. In that case several confusion matrices are needed. Of course, one can also create a bigger confusion matrix as follows

Table 3.2: Multi-class confusion matrix. Here the terms true positive, negative and false positive, negative lack any meaning unless you consider one class against the others.

		Predicted				
		Class 1	Class 2	Class 3	Class 4	Class 5
	Class 1					
	Class 2					
Actual	Class 3					
	Class 4					
	Class 5					

3.3.2 Accuracy

The first metric we will be defining is the most intuitive one. It is basically the percentage of correct predictions. Using the terminology from Table 3.1 it can be expressed as

$$\frac{TP+TN}{TP+FP+TN+FN} \tag{3.1}$$

The main disadvantage of the accuracy comes when dealing with imbalanced datasets. By predicting the class that appears the most a high accuracy can be easily achieved in those cases.

The accuracy is a binary classification metric, later on in subsection 3.3.8 an adaptation to multi-class problems is described.

3.3.3 Precision

The accuracy requires to know how many true negatives there are. But in some problems like object detection that is not always possible. Due to how labels are constructed in some cases it is impossible to know how many true negatives can be considered, although in the case of object detection the trend seems to be changing these days [16]. In those cases it makes sense to define the percentage of correct predictions only within the positive class. Mathematically, precision is defined as

$$P = \frac{TP}{TP + FP} \tag{3.2}$$

This metric, however, can be easily fooled. In an image with hundreds of cells, by only predicting one true tumoural cell you achieve a precision of 100%. But that is a useless value since you would be missing on most relevant cells.

3.3.4 Recall

As opposed to precision, recall focuses more on what relevant values are retrieved rather than them being correct. It is defined like this

$$R = \frac{TP}{TP + FN} \tag{3.3}$$

Again, this metric can also be fooled. By predicting everything as positive you achieve 100%. Since this metric ignores false positives you are left with a biased metric against the negative label.

3.3.5 F_1 Score

In order to take the best from precision and recall, the F-measure was proposed in 1992 at the Proceedings of the 4th conference on Message understanding [26]. The F-measure is defined as the harmonic mean between precision and recall.

$$\frac{2 \cdot P \cdot R}{P + R} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{3.4}$$

Nowadays it is called F_1 score because that measure has been extended to what is called the F_β score.

$$F_\beta = \frac{(1+\beta^2) \cdot P \cdot R}{\beta^2 \cdot P + R} = \frac{(1+\beta^2) \cdot TP}{(1+\beta^2) \cdot TP + \beta^2 \cdot FP + FN} \tag{3.5}$$

By taking $\beta = 1$ the original F-measure is obtained. This metric achieves 100% when all the relevant positive samples are retrieved and only the relevant samples, therefore it is not so easily fooled. Moreover, it is less prone to suffer from class imbalance as the accuracy does.

There are three ways of extending the F_1 score to a multi-class classification problem. All of them involve some kind of averaging the individual F_1 scores computed by considering one class against all the others.

3.3.6 Macro F_1 Score

The first way of averaging individual F_1 scores is by simply taking the arithmetic mean. If we have n classes, and we call F_1^i the F_1 score of the class i against the others, then the macro F_1 score is

$$\frac{1}{n}\sum_{i=1}^{n} F_1^i \tag{3.6}$$

The main drawback of only considering this metric is that classes can be imbalanced. In fact, for multi-class problems that is the rule rather than the exception. Giving equal weights to all of the classes harms the less represented labels.

3.3.7 Weighted F_1 Score

As a way of solving the main drawback of the macro F_1 score one can deal with the weighted F_1 score that averages individual scores based on their support, that is, based on the number of true instances by class. Let's call n_i the number of true instances of class i, then the weighted F_1 score is

$$\sum_{i=1}^{n} \frac{n_i \cdot F_1^i}{n_1 + \cdots + n_n} \tag{3.7}$$

3.3.8 Micro F_1 Score

Another way of averaging the individual F_1 scores is by micro-averaging. When macro-averaging, true positives, false negatives and false positives are computed per-class prior to averaging all the scores computed as defined in subsection 3.3.5. Micro-averaging changes the order. True positives, false negatives and false positives are first aggregated among all the classes and then the micro F_1 score is computed using the formula in subsection 3.3.5. To illustrate the computation let's consider the matrix from Table 3.2 and let's also fill it with false positives / negatives and true positives, giving the matrix in Table 3.3. Now, this terminology only makes sense when splitting by class. For that reason we will denote by TP_i the true positives of class i, and FP_i, FN_i the false positives and negatives of same class i. Notice that what is a false positive for one class can be a false negative for another.

Table 3.3: Multi-class confusion matrix. The values in the diagonal are all true positives when considering one class agains the others. Depending on which class you are considering, the values considered false positives and false negatives could be interchanged.

		Predicted				
		Class 1	Class 2	Class 3	Class 4	Class 5
	Class 1	TP_1	FP_1, FN_2	FP_1, FN_3	FP_1, FN_4	FP_1, FN_5
	Class 2	FP_2, FN_1	TP_2	FP_2, FN_3	FP_2, FN_4	FP_2, FN_5
Actual	Class 3	FP_3, FN_1	FP_3, FN_2	TP_3	FP_3, FN_4	FP_3, FN_5
	Class 4	FP_4, FN_1	FP_4, FN_2	FP_4, FN_3	TP_4	FP_4, FN_5
	Class 5	FP_5, FN_1	FP_5, FN_2	FP_5, FN_3	FP_5, FN_4	TP_5

The micro-average is then the sum of the diagonal divided by the sum of all the entries in the matrix. It is quite similar to how the accuracy is computed. For that reason sometimes this metric is referred to as the accuracy in multi-class problems.

3.3.9 Dice's coefficient

The Dice's coefficient can be viewed as a generalisation of the F_1 score. Given two sets X and Y the Dice's coefficient is defined as

$$\frac{2 \cdot |X \cap Y|}{|X| + |Y|} \tag{3.8}$$

If we consider X as the set of relevant items and Y as the set of retrieved elements, we obtain the F_1 score. To show that, let's see what are the sets of retrieved and relevant objects. The relevant items are the sum of true positives and false negatives. The retrieved ones are the sum of true positives and false positives. The intersection is clearly just the true positives, so $|X \cap Y| = TP$ and also $|X| + |Y| = 2 \cdot TP + FN + FP$. Substituting into the formula for the Dice's coefficient the formula for the F_1 score appears.

But the Dice's coefficient can be used for more than that. It can be used as a metric for image segmentation problems. By defining X as the set of pixels that belongs to a class in the ground truth and Y as the set of pixels of the same class but in the predictions, the Dice's coefficient can be used for evaluating the performance of a segmentation model.

Furthermore, it is possible to extend that measure to a loss function. All the metrics presented so far require a thresholding function at the end. That function has a discontinuity at 0.5 but worse than that, are completely flat in the rest of the $[0, 1]$ interval, which means the gradient is zero. A null gradient stops any deep learning method from using them as loss functions. The adaptation of the Dice's coefficient to a loss was made by Milletari et al. [25]. The idea is to take advantage from the fact that pixel class probabilities range from 0 to 1. The Dice's coefficient can be seen as a boolean operation. Therefore, the Dice's loss is a function that when given just 0s and 1s is that same boolean operation, but is also defined in the rest of the $[0, 1]$ interval and not just in the extremes. To be consistent with the original notation, let's call p_i to the predicted probabilities and g_i to the ground truth probabilities, where i ranges from 1 to N being N the total number of pixels. Thus, the Dice's loss is

$$D = \frac{2 \sum_{i=1}^{N} p_i g_i}{\sum_{i=1}^{N} p_i^2 + \sum_{i=1}^{N} g_i^2} \tag{3.9}$$

It is clear that when $p_i \in \{0, 1\} \forall i$ and $g_i \in \{0, 1\} \forall i$ the result is the same as the Dice's coefficient. But in this new version, the gradient can be computed with respect to any pixel with this formula.

$$\frac{\partial D}{\partial p_j} = 2 \cdot \frac{g_j \left(\sum_{i=1}^{N} p_i^2 + \sum_{i=1}^{N} g_i^2 \right) - 2 p_j \sum_{i=1}^{N} p_i g_i}{\left(\sum_{i=1}^{N} p_i^2 + \sum_{i=1}^{N} g_i^2 \right)^2} \tag{3.10}$$

3.3.10 ROC AUC

Another metric that avoids thresholding the probabilities is the Receiver Operating Characteristic Area Under the Curve. It evaluates the ordering of the probabilities instead of the actual predictions. Before diving into the details of the ROC curve, we need to first define the false positive rate.

$$FPR = \frac{FP}{FP + TN} \tag{3.11}$$

False positives and true negatives depend on the threshold selected. Using 0.5 gives some predictions while using -1 give everything as positive and using 2 returns all negatives. By changing the threshold different FPR are obtained. Moreover, the Recall, also known as true positive rate (TPR), changes when using different thresholds too. There is a balance between both of them, similar to what happened with precision and recall. Both metrics can be fooled, but not at the same time. So by changing the threshold one can measure which metric is being fooled the most. The ROC curve plots the TPR in the y axis and the FPR in the x axis. An example is on Figure 3.2.

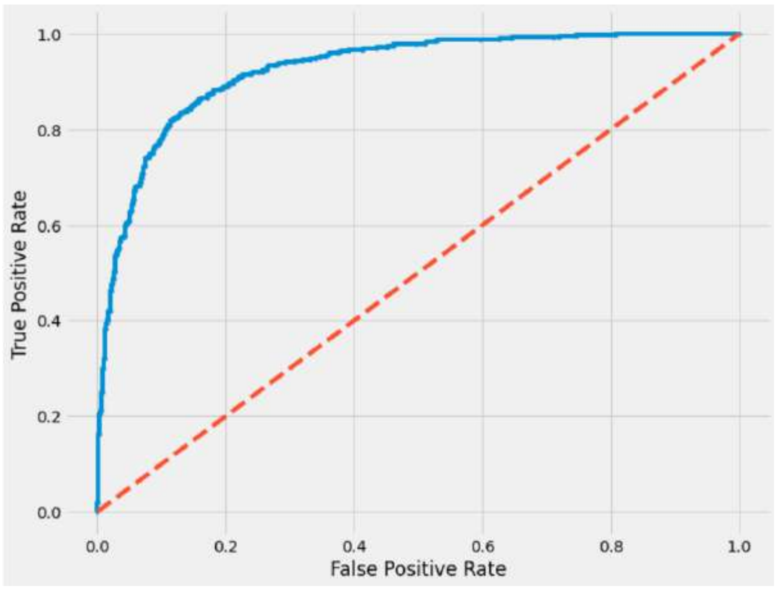


Figure 3.2: Example of a ROC curve.

This curve always starts in the $(0, 0)$ and finishes in the $(1, 1)$ corresponding to thresholds 2 and -1 respectively (or any other threshold that returns all negatives and all positives). Given that, the ROC AUC is, as its name states, the area under that curve. A random classifier would yield the orange line in Figure 3.2 that goes from $(0, 0)$ to $(1, 1)$, while a perfect classifier would go from $(0, 0)$ to $(0, 1)$ and then to $(1, 1)$. Therefore, any classifier has an AUC roughly between 0.5 and 1. Notice that an AUC of 0 corresponds to an adversarial classifier, which is a perfect classifier but changes negatives with positives.

3.3.11 Calibration

To finish this section, I'll cover a different way of evaluating classifiers. Normally, the effectiveness of a classifier is measured depending on how well it classifies samples. But that has its flaws too. It may be interesting to have a way of measuring the uncertainty of a prediction. If I am going to be diagnosed with cancer I want to know how likely that is wrong. Having a 51% probability of dying is not the same as having a 99.9% probability. None of the previous metrics evaluates the quality of the uncertainty provided by the probabilities. Deep learning methods oftentimes suffer from overconfidence [40, 21, 20]. Neural networks typically provide probabilities that are close to 1 even when the available information is not enough to be so sure about that prediction. To analyse that phenomena, reliability diagrams were invented. An example is provided in Figure 3.3. On the x axis there is the predicted probability while in the y axis is an estimation of the real probability.

More specifically, predicted probability is quantised. The x axis represents the probability distribution of the model, but we only have a finite collection of samples, so that distribution is estimated using the histogram. This means that the points in Figure 3.3 in fact represents a set of samples that are predicted with very similar probabilities. The magnitude in the y axis is how many of them really are positive. As an example, suppose we have 4 samples predicted

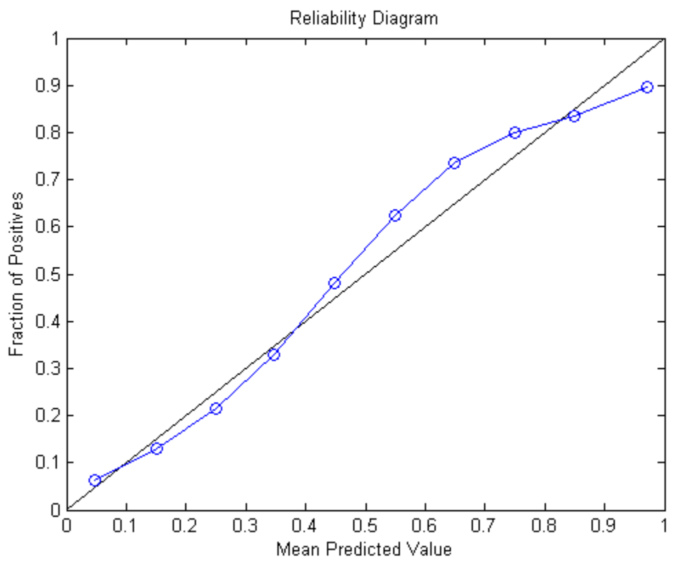


Figure 3.3: Example of a reliability diagram.

with probabilities $0.74, 0.75, 0.75, 0.76$ and only three of them are real positives. The mean predicted probability of that group is 0.75, and the real probability is also 0.75 since 3 out of 4 are positives. This would give a point in the black line, a perfectly calibrated point. On the other hand, consider the following example. Four samples, 3 with probability 1 and 1 with probability 0. From the first three, one is negative and two are positive. The sample with probability 0 is negative. In this case, we have a point at $(0, 0)$ and another at $(1, \frac{2}{3})$. This time, we have a less calibrated prediction. Nonetheless, in both examples we have an accuracy of 75%. This illustrates the fact that calibration is independent from how well you classify samples. An example of an uncalibrated model is on Figure 3.4.

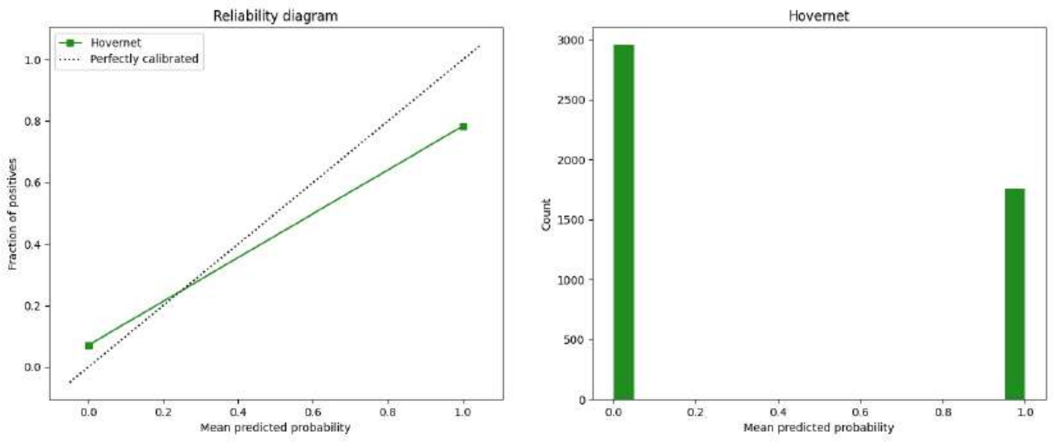


Figure 3.4: Reliability diagram of Hovernet from one of our experiments. At the right is the histogram of the predicted probabilities, clearly not a uniform distribution.

Reliability diagrams can be converted into a metric in the same way the ROC was made a metric by using the area under it. Here, instead of the area under the curve, the area between the model curve and a perfectly calibrated curve is taken. Furthermore, since points in that diagram can represent sets of arbitrary size the area of each bin is weighted by the number of samples in it. Where now we are calling the points bins, since in fact, each point has a width that represents the range of probabilities it takes into account. A more appropriate visualisation is on Figure 3.5[1] since it also shows the differences between the real and predicted probability which is what is used for computing the metric. The mean of the absolute value of the differences showed in that figure is called Expected Calibration Error (ECE).

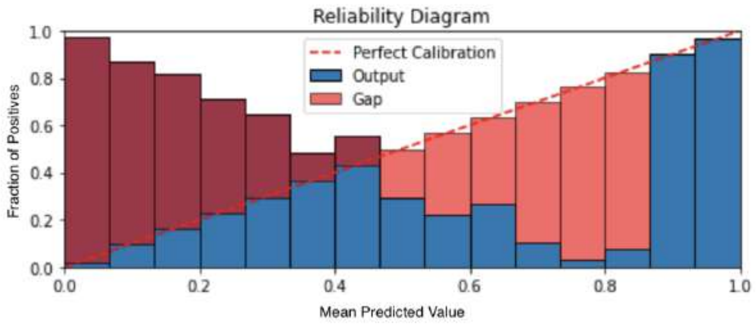


Figure 3.5: Another way of representing a reliability diagram. The blue colour is the real probability of a class in that group. The two red colours denote the difference with respect to the optimally calibrated classifier. The difference can be by lack or by excess, that is why there are two colours. The darker one means the real probability is higher than the predicted one. And the lighter one means the predicted probability is lower than what it should be.

3.3.12 Extending metrics

I have depicted a clear way of measuring uncertainty calibration in the previous section but it has at least one downside, it is only for binary problems. I'll devote this subsection to a way in which one can extend any binary metric to a multiclass metric. Suppose we have a metric called M. Now, given a multiclass setup one can compute M^k, the same metric but considering the class k against the others. By grouping all the other classes into one, the problem is now binary and so we can compute M. By averaging all the M^k we get the extension of M. For instance, the macro F_1 score correspond to doing this to the F_1 score with equal weights to all classes. While the weighted F_1 score is the weighted average of the individual F_1 scores. However, the micro F_1 score, which is also called accuracy, does not correspond to the extension of the accuracy, although it is quite close. The extension of the accuracy is in fact $\frac{2}{N}\text{Micro} + \frac{N-2}{N}$ being N the number of classes. The proof is straightforward. If we call D the diagonal of the confusion matrix and T the total sum then

$$Acc^k = \frac{a_{kk} + \sum_{i,j\neq k} a_{ij}}{\sum_{i,j} a_{ij}} \tag{3.12}$$

$$Acc_{extended} = \frac{1}{N}\sum_k Acc^k = \frac{\sum_k a_{kk} + \sum_k \sum_{i,j\neq k} a_{ij}}{N\sum_{i,j} a_{ij}} \tag{3.13}$$

$$= \frac{2D + (N-2)T}{NT} = \frac{2}{N}\text{Micro} + \frac{N-2}{N} \tag{3.14}$$

where the intermediate step can be deduced by counting how many times does each term appears. The diagonal terms appear $N-1$ times while the rest of terms appear $N-2$ times, therefore $\sum_k \sum_{i,j\neq k} a_{ij} = D + (N-1)T$, concluding the proof. Surprisingly enough, the extension of the accuracy approaches 1 as the number of classes increases so it is quite a bad metric and is not used in practice, that is why the Micro F_1 is always preferred. In the

[1]Image taken from https://github.com/EFS-OpenSource/calibration-framework/issues/17 Accessed 15th May 2023

experiments below, we will be using the extension of the ECE and call it just ECE for simplicity in the multiclass datasets.

3.4 Experiments

Two experiments were carried out to show the usefulness of graph neural networks in the problem of cell detection and another two experiments were made to provide insights about the models involved. Results are in section 4.1, this section is only to describe the experiments themselves.

3.4.1 GNN vs CNN

The main experiment and the one that supports the principal thesis of this work compares convolutional neural networks with graph neural networks. In order to state that GNNs are actually useful they must outperform CNNs (Hovernet) in every metric defined in section 3.3. But outperforming in just one dataset is not enough. For that reason the comparison will be made using all the datasets from the original Hovernet article [12, 10, 37], and using several internal datasets of the DigiPatics project [33]. More concretely, the methods will be evaluated in the HER2 stained breast dataset and the H&E stained lung dataset. For the multi-class problems, multi-class metrics will be taken into account and for binary problems, binary metrics.

3.4.2 GNN vs XGBoost

Outperforming CNNs may not be due to the graph structure. It is possible that the extracted features are sufficient to improve CNN results. It may be that the edges do not contain any useful information. To account for that, GNN must be compared with node-only methods. In this case we selected XGBoost [8] for comparison since it has been in the leaderboard of several Kaggle competitions[2]. By comparing node-only methods with GNNs we can know if the value of GNNs reside on the extracted features or on the graph structure itself.

3.4.3 Void GNNs

Finally, the last experiment is about knowing if the extracted features are relevant or not. We distinguish two types of features: morphological and probabilities. The first group is independent from the Hovernet output while the second is not. To see which of them are more important we train the same GNN models with different features. One with all the features, one only with morphological features and another only with probabilities. By comparing the metrics obtained we can discern which set of features is more relevant.

3.4.4 Scaling CNNs

The original Hovernet article restricted their images to having 270x270 pixels. That is a very narrow view of the cells. It seems intuitive that increasing the receptive field of the model increases its performance too. But science does not understand intuition, only facts. So an experiment needs to be carried out in order to prove if increasing the receptive field is really beneficial. Apart from that, since the original weights of Hovernet are open source, we can initialise our weights with them. Therefore, there are four different models that we will call 270, 270FT, 518 and 518FT. The number indicates the resolution of the images and FT means

[2]https://dataaspirant.com/xgboost-algorithm/

fine-tuned. The models with FT in the name initialise their weights with the ones from the original Hovernet article. The other models initialise their weights at random. The datasets used in this experiment are only our own lung and breast datasets.

Chapter 4

Results

4.1 Quantitative analysis

In what follows there will be four tables per section showing the results of each experiment for the four different datasets described in section 2.2. The order will also be the same, with the lung dataset first since it is the main focus of the thesis. Also, at every table the best results will be marked in bold.

4.1.1 GNN vs CNN

This is the main experiment of my thesis. It shows whether using graphs improves over not using them. We observe that for the case of lung, putting a graph on top of Hovernet gives better metrics. The last metric is of special interest. It means that overall the predicted percentage of tumoural cells is just three points away from the real percentage. We strongly believe that this value is more than enough to consider the model a good fit. Physician on their day to day approximate their percentages to the nearest 5% to be faster and because they are estimating areas of tumour. We believe the human eye cannot approximate areas any better. It would be interesting to perform an experiment if that the case. Giving people the labelled images and asking them to estimate the percentage given the real labels. I predict that the error in percentage prediction is going to be around that 5%.

On the other datasets, for two out of the three multiclass datasets it does improve. Moreover it also lowers the ECE, showing it is not only predicting better but is better calibrated. Nonetheless, the remaining dataset (CoNSeP) poses a question. Why is not working there? Probably because the structure is not so useful when working with colorectal cancer. It may also be that since there are seven classes, the relationship between them are more complex. Maybe if we considered just two classes the GNNs would again win. We would have liked to perform such experiment, but we lack the expertise on colorectal cancer to decide how to properly label the cells to achieve such results. Another remarkable property is that even though GNNs are worse in the CoNSeP dataset overall, they have a lower ECE, meaning they are still better calibrated.

In the datasets that GNNs do improve over CNNs, it is left to discern the real cause why that is happening. When talking to experts, we expected GNNs to be a good fit for the lung dataset but not for the breast one. In the lung dataset they pay special attention to the way cells are grouped together, while in the breast dataset it was not so much the case. So, why does using graphs give better results in both cases? The GNN vs XGBoost and the Void GNN experiments will throw light in this matter. Also, we are here referring to GNNs in an abstract manner for simplicity, the specific architecture that performed best on each case is reported in Appendix E.

Table 4.1: Result of the GNN vs CNN experiment.

	Accuracy (↑)	F_1 score (↑)	ROC AUC (↑)	ECE (↓)	%Err (↓)
CNN	82.39%	57.69%	75.20%	0.1653	11.89%
GNN	**83.27%**	**66.53%**	**86.84%**	**0.0884**	**3.52%**

Table 4.2: DigiPatics lung dataset.

	Micro F_1 score (↑)	Macro F_1 score (↑)	Weighted F_1 score (↑)	ECE (↓)
CNN	65.12%	40.55%	66.38%	0.3243
GNN	**70.47%**	**42.53%**	**71.13%**	**0.2501**

Table 4.3: DigiPatics breast dataset.

	Micro F_1 score (↑)	Macro F_1 score (↑)	Weighted F_1 score (↑)	ECE (↓)
CNN	**71.11%**	**54.06%**	**70.39%**	0.0680
GNN	64.44%	47.87%	61.42%	**0.0539**

Table 4.4: CoNSeP dataset.

	Micro F_1 score (↑)	Macro F_1 score (↑)	Weighted F_1 score (↑)	ECE (↓)
CNN	82.06 %	68.76%	82.34%	0.0774
GNN	**88.71%**	**69.72%**	**89.05%**	**0.0251**

Table 4.5: MoNuSAC dataset.

4.1.2 GNN vs XGBoost

In the first experiment it was shown that GNNs outperform CNNs in most of the scenarios. But why? Is it due to the relations among cells or due to stacking another classifier on top of the CNN? In the former we should expect GNN to also outperform a node-only method like XGBoost. If it is the latter, then XGBoost should win or give similar results. This experiment can be viewed as an ablation study over the edges. We thought of using the same GNN without edges, but that is simply a multilayer perceptron which, based on my own experience, for tabular data is almost never a good approach. We wanted the node-only alternative to be as good as possible, that is why we chose XGBoost, because it is the state of the art for tabular data. If GNNs win XGBoost, then the edges are clearly relevant to the task, which is what we wanted to show.

So, are edges neccessary? As we can see here, the answer is not crystal clear. In some cases it is better to use GNNs and in others it is not. In order to further elucidate when is the case in advance to training the models, the qualitative analysis will help give some insight. Looking at individual images will provide some keys about when graphs are a good fit. Moreover, we find that for those datasets where GNNs perform better, in the Void GNN experiment the models trained without probabilities also perform better. While for those that XGBoost is the winner here, the models with probabilities are the winner there. Showcasing that for MoNuSAC and Digipatic lung datasets, the graph structure is indeed important and the performance boost is not due to stacking a classifier on top. This is also consistent with our prior knowledge of the breast dataset. We didn't expect graphs to be a good fit because they indeed aren't. Their performance boost is because of the overall model being a stack classifier.

Table 4.6: Result of the GNN vs XGBoost experiment.

	Accuracy (↑)	F_1 score (↑)	ROC AUC (↑)	ECE (↓)	%Err (↓)
CNN	82.39%	57.69%	75.20%	0.1653	11.89%
XGB	**83.77%**	62.64%	84.20%	0.1007	7.85%
GNN	83.27%	**66.53%**	**86.84%**	**0.0884**	**3.52%**

Table 4.7: DigiPatics lung dataset.

	Micro F_1 score (↑)	Macro F_1 score (↑)	Weighted F_1 score (↑)	ECE (↓)
CNN	65.12%	40.55%	66.38%	0.3243
XGB	**78.51%**	**47.36%**	**79.78%**	0.2502
GNN	70.47%	42.53%	71.13%	**0.2501**

Table 4.8: DigiPatics breast dataset.

	Micro F_1 score (↑)	Macro F_1 score (↑)	Weighted F_1 score (↑)	ECE (↓)
CNN	71.11%	**54.06%**	**70.39%**	0.0680
XGB	**71.54%**	51.64%	69.87%	**0.0320**
GNN	64.44%	47.87%	61.42%	0.0539

Table 4.9: CoNSeP dataset.

	Micro F_1 score (↑)	Macro F_1 score (↑)	Weighted F_1 score (↑)	ECE (↓)
CNN	82.06 %	68.76%	82.34%	0.0774
XGB	85.23%	**76.09%**	85.20%	0.0449
GNN	**88.71%**	69.72%	**89.05%**	**0.0251**

Table 4.10: MoNuSAC dataset.

4.1.3 Void GNNs

Another way of seeing if the good performance of graphs is due to the fact that we are considering the existence of edges or it is explained by the attributes of the nodes, is to train the graph models with and without the attributes. If there is no performance difference, then edges are relevant. Otherwise, it is less relevant. Moreover, there is a set of attributes that depends on the layers behind, the probabilities. To discern if GNNs are working cause they are stacked above or because there is a graph structure we also train the models with and without the probabilities. By training we refer to the full hyperparameter process as described in section 3.2 where we optimise parameters on training set and hyperparameters on validation set prior to evaluating on the test set. Specific details of the values that were selected are in Appendix E.

We observe that in the datasets that XGBoost did not outperform GNNs the GNN trained with no information about the prior probability given by Hovernet does in fact perform better than the model with those probabilities. However, in the two datasets that XGBoost did give better results we can see that using probabilities gives a benefit over using other types of features. This tells us that MoNuSAC and DigiPatics lung datasets have more structure than CoNSeP and DigiPatics breast datasets. In DigiPatics breast GNNs gave better results than CNN but it was due to stacking. For DigiPatics lung it was because the graph is indeed a good way of modelling the problem.

Another conclusion that derives from this tables is that some kind of features are needed, no matter how simple they are. Training the graph networks without any feature at all gave very poor results. It also resulted in a more unstable training in general. Providing the model with some information about the individual cells is crucial for the proper functioning of the method.

Table 4.11: Result of the Void GNNs experiment.

	Accuracy (↑)	F_1 score (↑)	ROC AUC (↑)	ECE (↓)	%Err (↓)
Full	83.27%	66.53%	86.84%	0.0884	3.52%
Probabilities	82.87%	63.10%	87.26%	0.0470%	7.08%
Morphological	**83.81%**	**70.18%**	**88.88%**	**0.0431%**	**0.78%**
Void	73.24%	0.00%	54.66%	0.1250	26.76%

Table 4.12: DigiPatics lung dataset.

	Micro F_1 score (↑)	Macro F_1 score (↑)	Weighted F_1 score (↑)	ECE (↓)
Full	**70.47%**	**42.53%**	**71.13%**	0.2501
Probabilities	67.94%	40.83%	68.91%	0.2666
Morphological	63.05%	30.48%	61.26%	0.2676
Void	36.91%	11.86%	24.96%	**0.2413**

Table 4.13: DigiPatics breast dataset.

	Micro F_1 score (↑)	Macro F_1 score (↑)	Weighted F_1 score (↑)	ECE (↓)
Full	64.44%	**47.87%**	61.42%	0.0539
Probabilities	**64.93%**	47.37%	**61.63%**	**0.0494**
Morphological	52.11%	29.42%	48.19%	0.0582
Void	29.88%	8.08%	15.78%	0.0742

Table 4.14: CoNSeP dataset.

	Micro F_1 score (↑)	Macro F_1 score (↑)	Weighted F_1 score (↑)	ECE (↓)
Full	88.71%	69.72%	89.05%	**0.0251**
Probabilities	82.59%	**73.43%**	82.60%	0.0784
Morphological	**92.01%**	69.83%	**92.02%**	0.0405
Void	53.61%	26.65%	52.20%	0.0580

Table 4.15: MoNuSAC dataset.

4.1.4 Scaling CNNs

Deleting last layer weights and retraining can give better performance, as shown below in Table 4.19. 270FT performs better than 270 and 518FT better than 518 for the CoNSeP dataset. This aligns perfectly with the results from [43]. They observe that reinitialising weights and retraining can boost performance. In our setup this translates to fine-tuning a model in the same dataset it was pretrained. That is because to perform transfer learning from a dataset to another we reinitialised the last layer of the decoders prior to training. The main reason for that was to adapt the model to a different number of classes. The last layer is the only one dependent on the number of classes, so reusing it is impossible without reinitialising the weights in some way. Also, you may have noticed that the metrics for 518FT are slightly different than those in the GNN vs CNN experiment even though they are the same model. That is because we trained the same model twice converging to slightly different checkpoints. I provide both metrics to demonstrate that the ordering is the same using either of both metrics, thus proving training is sufficiently stable.

Same result is obtained for MoNuSAC dataset. Fine-tuning the checkpoint trained on CoNSeP helps obtain better results when applied as initialisation for the models trained on MoNuSAC dataset which comes from a totally different distribution. The features learned by the encoder seems to generalise well to this other dataset. However, the results in the breast dataset differ from the ones in the CoNSeP and MoNuSAC datasets. In that case using a pretrained checkpoint didn't perform well at neither resolution. This may be because the local minima found for the CoNSeP dataset is far away from the nearest minima in the DigiPatics breast dataset, making a random initialisation a better method. Another reason may be that the breast dataset has a colour distribution very different that the others, probably because it used a different staining method.

Another remarkable fact is that using a field of view of 518 pixels instead of 270 gives better metrics no matter the initialisation nor the dataset, except for lung. We would have liked to further try a bigger field of view. But training the 518 models required more than 20 GB of GPU VRAM. Scaling to 1030x1030 images would require near 80 GB of GPU VRAM, which is only feasible using A100 or H100, which cost more than 10000€. Using CPU offloading was also not an option since that technique trades memory for time, typically increasing by 100 the time required. The models required around 4 hours to train. This means one experiment may take up to 2 weeks with a bigger field of view, which is clearly prohibitive. With our resources, 518 was the maximum we could achieve.

The lung case is quite strange for two reasons. First, because a smaller field of view gives better results while on the other datasets it doesn't. And second, because in previous experiments we made when the labels were not reviewed by an expert revealed exactly the opposite pattern. The DigiPatics lung dataset went through an iterative process until it arrived at the state it is here. The scaling experiment consistently gave better results for 518FT at all the steps except at the last one, when the labels were all reviewed by an expert. Although we still use the 518FT backbone for the GNN, it doesn't invalidate the point since the two stages are independent. We are just showing that adding a graph neural network on top improves, that is independent from improving the segmentation model.

Table 4.16: Result of the Scaling CNNs experiment.

	Accuracy (↑)	F_1 score (↑)	ROC AUC (↑)
270	78.77%	67.17%	79.00%
270FT	**85.46%**	**71.65%**	**79.90%**
518	81.07%	56.16%	69.73%
518FT	82.22%	57.01%	70.13%

Table 4.17: DigiPatics lung dataset.

	Micro F_1 score (↑)	Macro F_1 score (↑)	Weighted F_1 score (↑)
270	70.04%	38.22%	68.43%
270FT	48.55%	21.57%	34.44%
518	**72.36%**	**45.64%**	**75.80%**
518FT	68.78%	43.27%	71.58%

Table 4.18: DigiPatics breast dataset.

	Micro F_1 score (↑)	Macro F_1 score (↑)	Weighted F_1 score (↑)
270	54.20%	36.21%	54.62%
270FT	66.73%	49.01%	65.96%
518	56.75%	37.76%	59.45%
518FT	**71.02%**	**53.83%**	**70.52%**

Table 4.19: CoNSeP dataset.

	Micro F_1 score (↑)	Macro F_1 score (↑)	Weighted F_1 score (↑)
270	76.46%	56.20%	77.04%
270FT	77.97%	62.84%	77.93%
518	78.57%	58.64%	79.61%
518FT	**82.02%**	**70.40%**	**82.09%**

Table 4.20: MoNuSAC dataset.

4.1.5 CNNs metrics in detail

In all the experiments above I provided metrics only for 1-1 matchings of cells as explained in subsection 3.3.1. We are only considering cells that are located in almost the same position in the ground truth and in the predicted segmentation. That is a fair way of comparing CNN to GNN since GNN can only improve predictions on 1-1 matchings. However, it does not show the full picture. The CNN can miss cells or predict cells that do not exist. For that reason I here provide the same metrics as above but adding one extra class: the background. It never has true positives, only false positives and false negatives. Thus, it will almost always be worse than the metrics above given. For a counter example on when it is not worse, look at the next paragraph. The bigger the gap, the more cells it is missing or incorrectly predicting. Nevertheless, such gap does not alter any of the conclusions. In this project we did not try to narrow this gap because we consider the classification problem to be inherently more difficult than the segmentation problem. We believe improving the segmentation problem is simply a matter of more data and bigger models. If you think about it, you can detect cells without formal knowledge, it is just a matter of geometry and color. But recognising where are the tumours located? That requires more than 10 years of training to humans, and even in that case experts still have doubts.

Therefore we directed our efforts toward improving the classification, not the segmentation.

You may have noticed that in the DigiPatics breast dataset the macro F_1 score did not worsen when adding the background. It seems counterintuitive since we are adding a class with no true positives, only false positives and false negatives. But, the key to why this happens is purely technical. This is the global confusion matrix for the test set in the DigiPatics breast dataset:

$$\begin{bmatrix} 0 & 95 & 151 & 116 & 9 & 425 \\ 26 & 100 & 14 & 1 & 0 & 64 \\ 518 & 752 & 1136 & 819 & 17 & 445 \\ 25 & 6.0 & 54 & 336 & 28 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 509 & 48 & 18 & 3 & 1 & 2672 \end{bmatrix}$$

Why is this matrix problematic? Well, there is one class which does not appear at all in the ground truth. The reason why this causes the problem is because the two macro F_1 scores are computed differently. The "With Background" metrics are computed using the confusion matrix and a custom function I designed. The "Without Background" metrics are computed using pairs of labels and the sklearn f1_score function. Both functions are coded properly, that is not the problem. But in my implementation of the metric, I coded a function that estimates the number of classes based on the classes with support in the ground truth, and also ignoring the zero class. However, the sklearn function estimates the number of classes as the class with the maximum label. This is making one metric being divided by 4 and another being divided by 5. If we adjust the sklearn given metric to consider 4 classes instead of 5 we get a macro F_1 score of 50.69% which is bigger than 44.56% as expected. The reason I don't adjust this metric in the other tables is because the GNN methods are also evaluated using the sklearn function so the comparison is fair as it is. I just left the metrics below unchanged to showcase how a simple decision in design of an algorithm or method can affect the final conclusions, even creating mathematically impossible situations.

Metrics as presented here seems to put in a very bad place the Hovernet algorithm when dealing with the background. It seems as if it is missing a lot of cells and predicting non-existent cells. It is more complex than that. Let's look at an illustrative image from the breast dataset in which metrics are less than 60%.

Table 4.21: Hovernet evaluated with and without background in four different datasets.

	Accuracy	F_1 score	ROC AUC	Micro F_1	Macro F_1	Weighted F_1
With Background	NA	NA	NA	66.12%	66.06%	72.11%
Without Background	82.39%	57.69%	75.20%	NA	NA	NA

Table 4.22: DigiPatics lung dataset.

	Micro F_1 score	Macro F_1 score	Weighted F_1 score
With Background	50.57%	**44.56%**	57.89%
Without Background	**65.12%**	40.55%	**66.38%**

Table 4.23: DigiPatics breast dataset.

	Micro F_1 score	Macro F_1 score	Weighted F_1 score
With Background	48.38%	49.58%	57.84%
Without Background	**71.11%**	**54.06%**	**70.39%**

Table 4.24: CoNSeP dataset.

	Micro F_1 score	Macro F_1 score	Weighted F_1 score
With Background	53.29%	43.95%	67.67%
Without Background	**82.06%**	**68.76%**	**82.34%**

Table 4.25: MoNuSAC dataset.

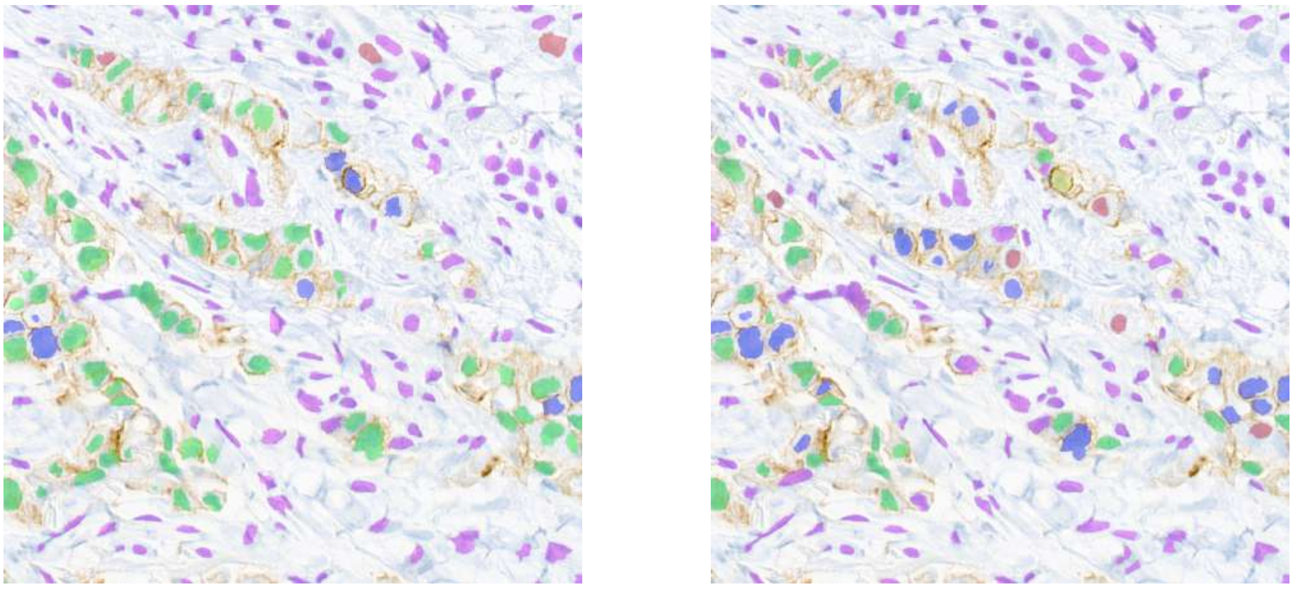

(a) Ground truth (b) Prediction

Figure 4.1: Comparison of the ground truth labels with the prediction from Hovernet.

By only looking at Figure 4.1 can you spot rapidly all the missing cells? Probably not. The segmentation seems accurate. But there are a lot of subtleties to take into account. Let's look now at Figure 4.2.

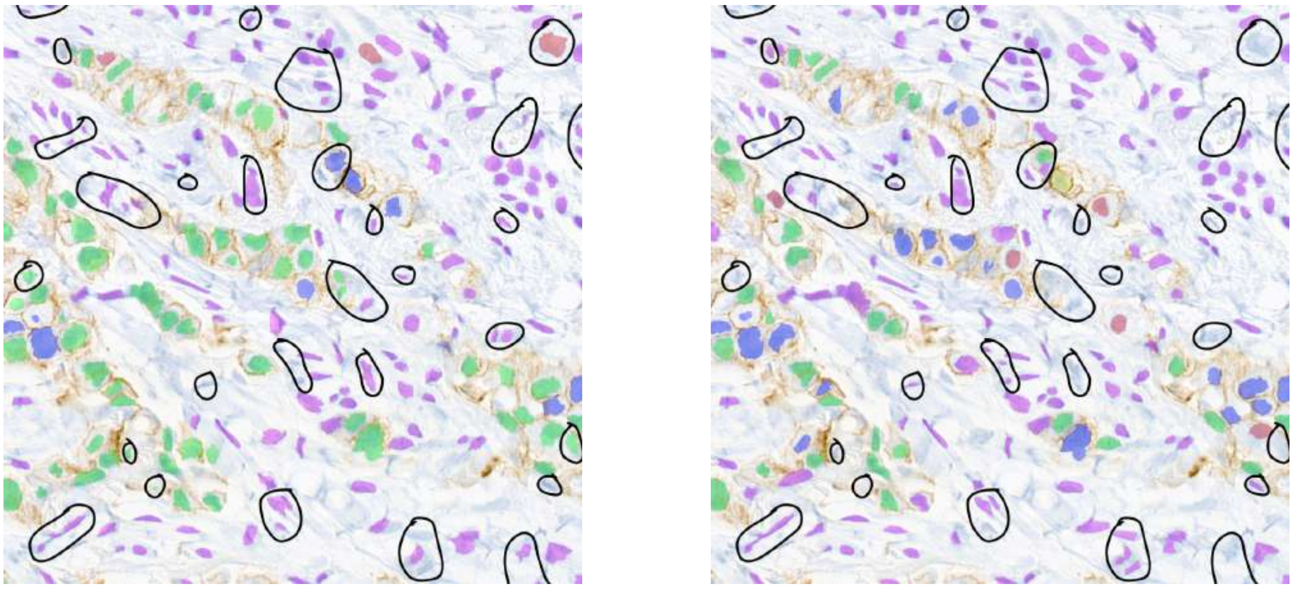

(a) Ground truth (b) Prediction

Figure 4.2: Comparison of the ground truth labels with the prediction from Hovernet with all the differences marked in black. We are only considering as a difference if the cells do not coincide. We are not looking at the classes of detected cells, just at the missing cells.

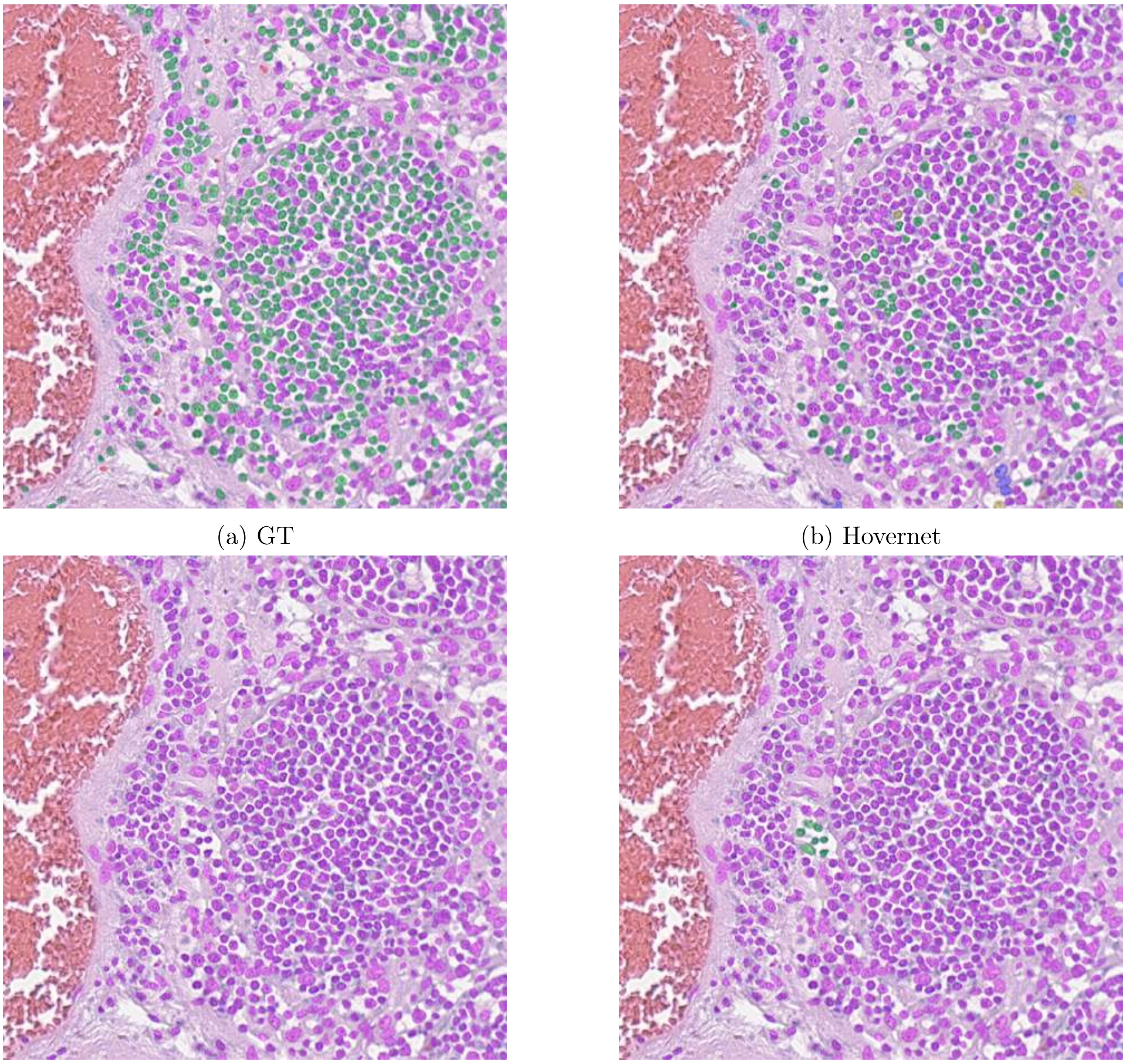
As you can see there are a lot of inconsistencies. But those inconsistencies are very difficult to detect. They can be divided into three groups: missing cells on GT, missing cells on prediction and cells which are split into parts. Most of the missing cells are very dubious. I cannot really tell if all the cells detected by Hovernet and by GT are really not in the GT, many seem to be plausible cells. And the case of cells which are split is difficult to say if it is really a problem. In some cases in can be post-processed as explained in Appendix B. And in any case, the cell is predicted, just in another way, so it is not such a big mistake. Another factor to highlight is that these mistakes occur typically on small cells. This means that if we only look at the area we will get high metrics, like the Dice loss reported on the Hovernet article. But since we are giving equal weight to every cell, and there are lots of small cells, they can make the metrics low even though you don't perceive such a high difference. The drop in performance when considering the background is around a quarter of the original metric. Looking at Figure 4.2 it seems plausible that around a quarter of the cells are detected inconsistently but overall, they do not represent a big area, so visually, the result seems good enough.

4.2 Qualitative analysis

In this section I will be providing insights as to why and when graph neural networks are a good regularizer. The overall effect of using GNNs can be described as finding groups. In some cases it is a good approach in others it isn't, it depends on the data. Throughout this sections I will be referring to all the classes by their colours. They all have a more profound meaning, like epithelial or inflammatory or tumoural. But for simplicity I will just refer to them by the colour they are painted with.

4.2.1 CoNSeP

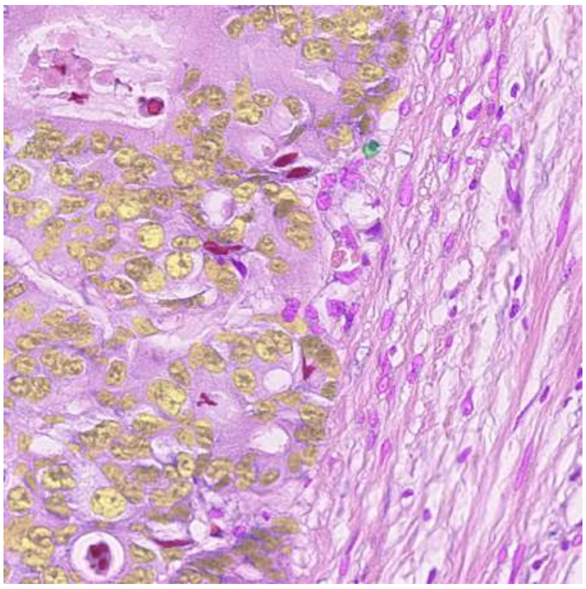
This dataset is the smallest of all four which makes a model with higher inductive bias like Hovernet perform better. Graphs are a more general structure, and that means they require more data to function properly. As a first example look at Figure 4.3. Hovernet tries to identify some of the green cells while the graph convolution just consider everything to be a big group. However, Hovernet also misclassifies some cells as blue or yellow when they are not, which is something the GCN does not do. In this aspect, and as will be seen with more examples, GCN normally play safe and assign the class with more cells as the main class of the group. GCNs do not typically try to find outliers which is something Hovernet does but sometimes, as in this case, it fails. Graph attention on the other hand may focus on smaller groups. In this case, GAT has managed to properly identify a small green nest of cells, just missing one of the cells in that nest.

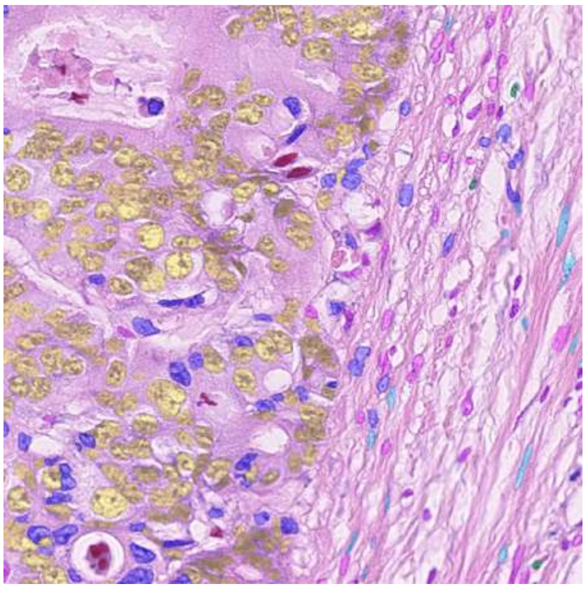

(a) GT (b) Hovernet

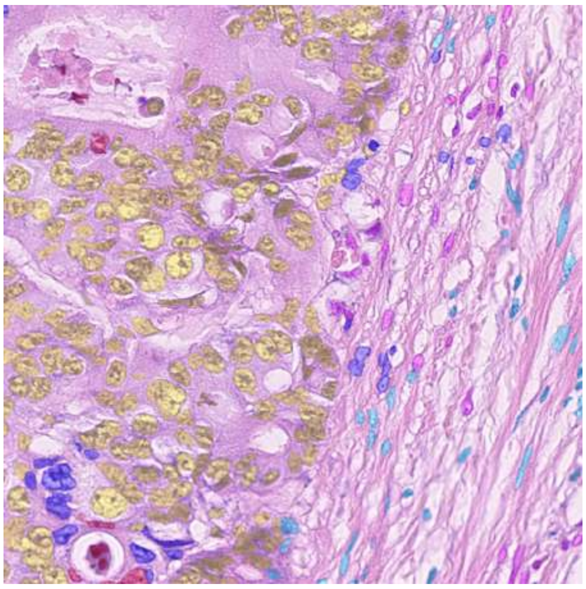
(c) GCN (d) GAT

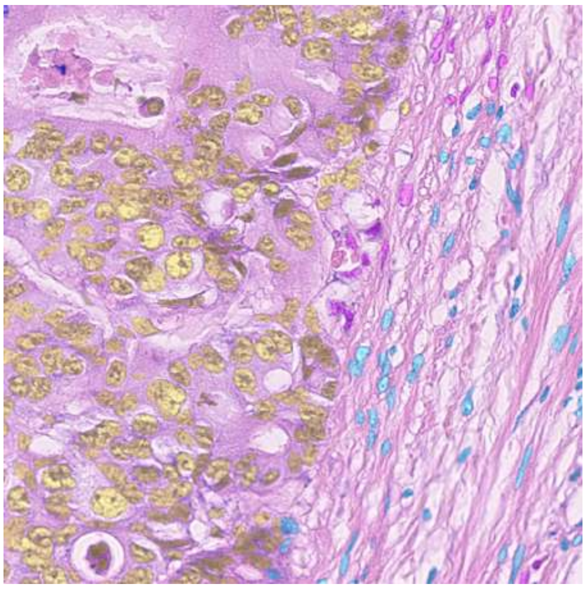
Figure 4.3: Comparison of the predictions of Hovernet, graph convolution and graph attention methods.

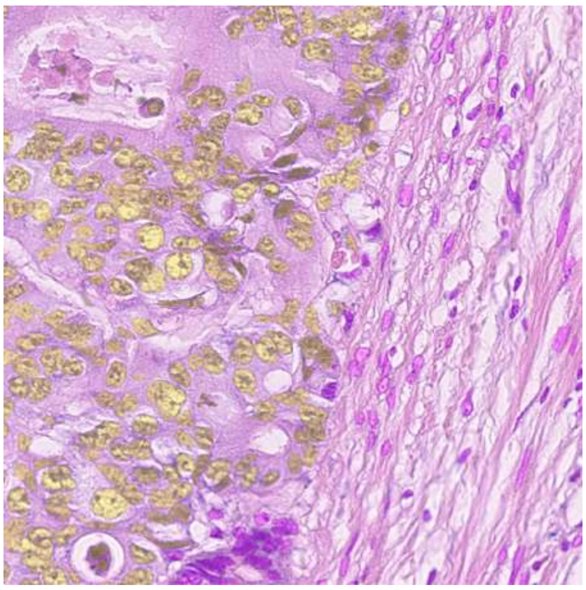
To illustrate how diverse is this concrete dataset, I will show another example where the sizes of the cells are different, the labels are different, and the way the cells are structured is also different. This example shows how difficult this dataset is. With so many classes, the number of possible ways of arranging cells is very high. Making less than 30 images not enough for the graph to learn something useful.


(a) GT


(b) Hovernet


(c) GCN


(d) Probabilities only


(e) Morphological features only

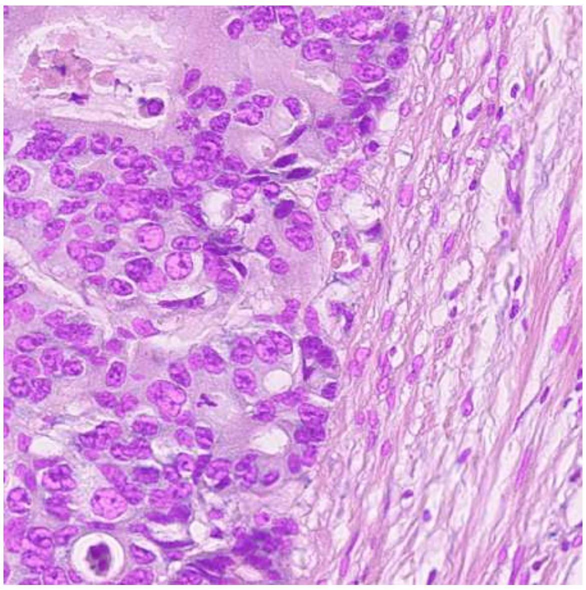

(f) Void

Figure 4.4: Illustration of the behaviour of the different models.

In Figure 4.4 we can see that Hovernet misses some of the magenta cells and classifies them as blue. Here, the effect of the GCN is positive in some aspects and negative in others. The positive side is that it recovers most of the yellow cells thanks to considering them as a whole. The negative aspect is that it expands the cyan cells which were initially misclassified by Hovernet. This is a common pattern. Since the GNN operates using the probabilities from Hovernet, it can propagate the errors from it. As further proof of that look at the same example, but now look at the output from the models trained without the probabilities of Hovernet and trained only with those probabilities. The model that was trained only with probabilities further propagates the cyan group while the model trained without them did not produce any cyan cells at all. There is a trade-off when using Hovernet probabilities. It can help fix groups where Hovernet misclassified a few of the inner cells. But relying too much on Hovernet probabilities can worsen the results. When using just morphological features, the GNN is more independent from Hovernet which in this case was beneficial. Nonetheless, using no features at all is a bad idea. In this specific example the model trained only with the graph but no features classified everything as magenta, wrongly forgetting the yellow cells. Visually it is quite clear that the yellow cells and the magenta ones are a separate group, but if you just consider nodes in a graph, without indicating the area or the perimeter, there is not enough information to detect two groups in this image.

To end with the analysis of this dataset, I will display another image where cells of different types are mixed together. In Figure 4.5 we can see how Hovernet predicts a more diverse set of cells than the GCN. It has yellow cells which are not surrounded by other yellow cells, it also has cyan, magenta and blue. In contrast, the GCN has mostly predicted the cyan as the dominant class. The effect of using GNNs is visible again. Hovernet takes risk and predicts lonely cells, cells with no neighbour of the same class. GNNs create neighbourhoods of cells. In this case neither of them are really a good fit since Hovernet predicts many classes that are not real and the GCN cannot really discern the mix of cyan and magenta that is in the middle of the image and predicts the whole middle group as cyan.

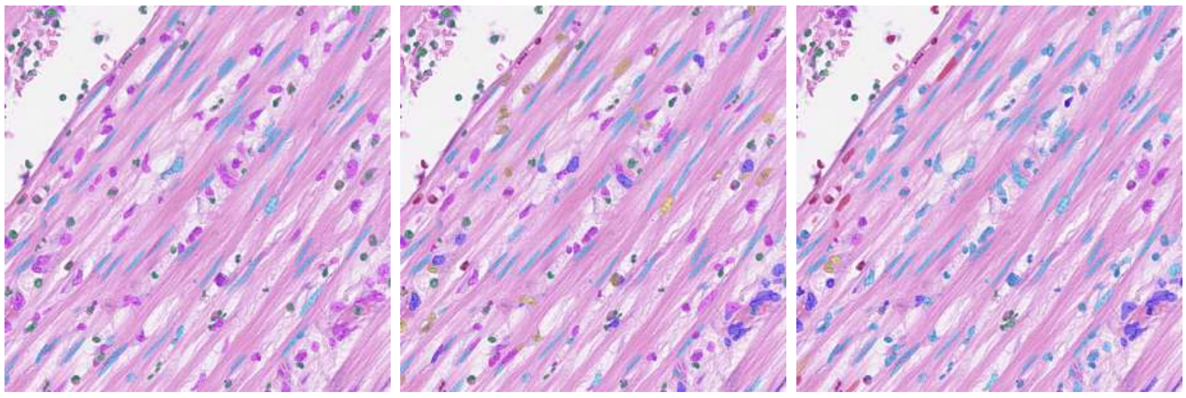

(a) GT (b) Hovernet (c) GCN

Figure 4.5: Last example of CoNSeP database. It mixes different cell types which makes it difficult for the GNN to work properly.

4.2.2 MoNuSAC

In previous sections I have stated that this dataset was a good example of where the graph neural networks are a good fit. With the CoNSeP dataset we have seen that those models tend to group cell togethers. When there is a mix of different classes it works poorly. In Figure 4.6 we can see three examples out of this dataset which are exactly the opposite. They contain well defined groups. It is difficult to appreciate but in the image on the left there are two well defined groups, one green and one red and in the image at the left there are also blue cells close to each other. Many of the images in this dataset are like the one in the middle, with only one class in the whole image. Having this setup it is expected to see Hovernet misclassify some cells since it is not considering the image as a whole while the GNNs solve that problem by merging everything into big groups.

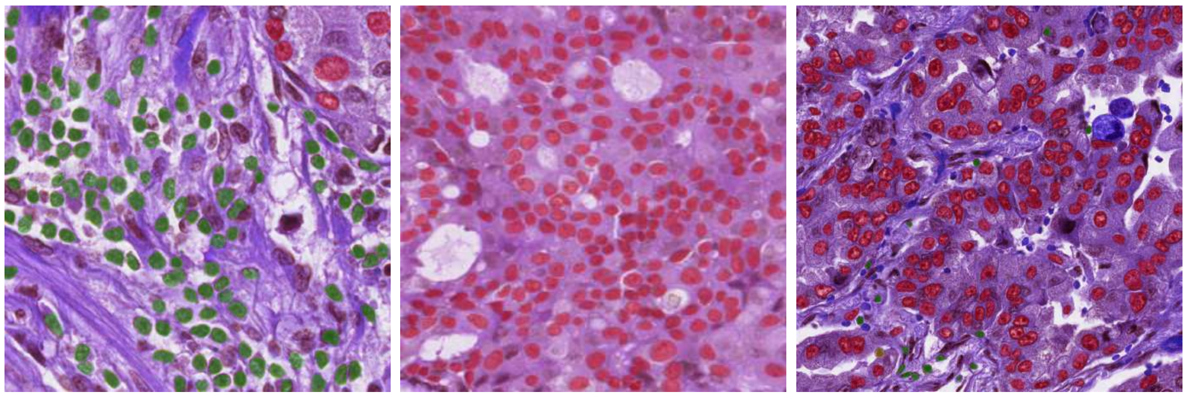
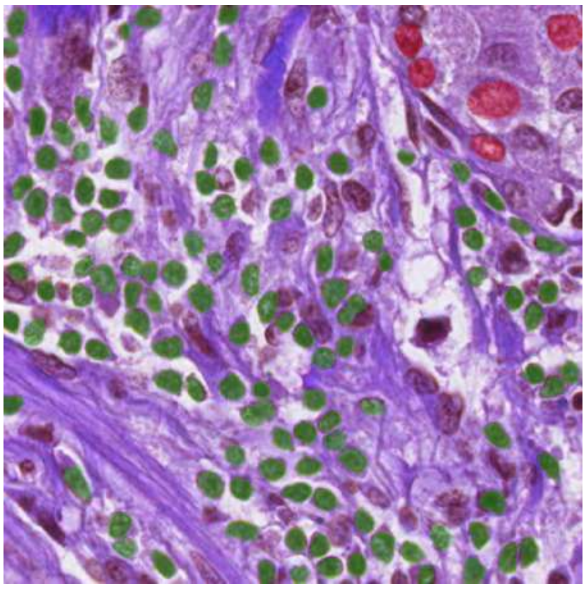
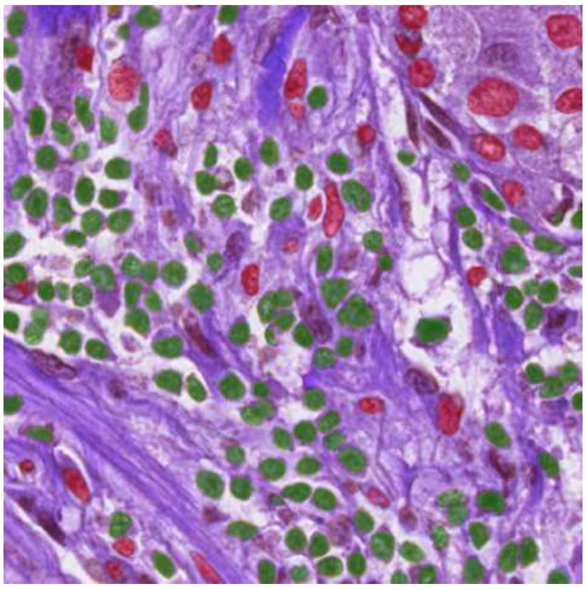


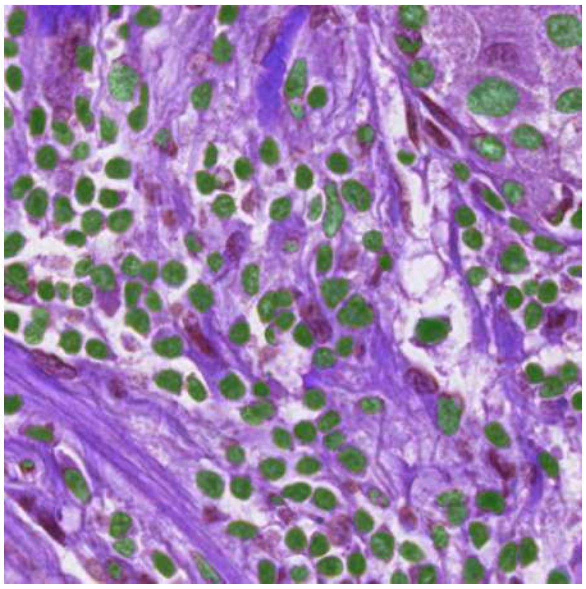
Figure 4.6: Three examples with their ground truth overlayed on top.

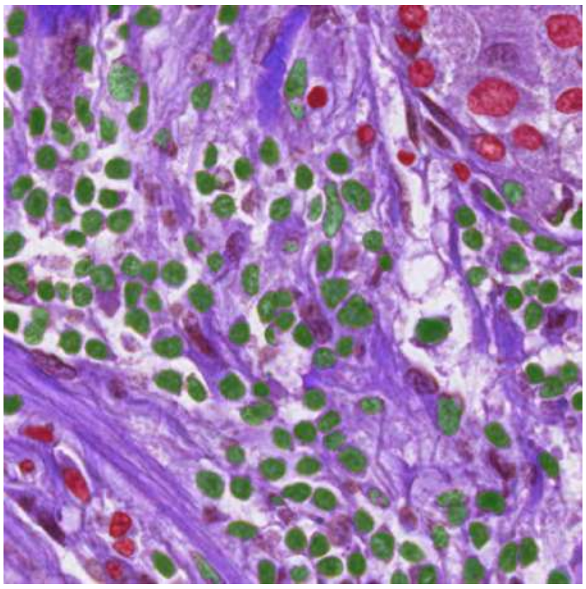
Let's dive into the details of what every model predicted with the image on the left. Same as what happened in CoNSeP, here in Figure 4.7 the Hovernet randomly misclassifies some green cells into red cells. The graph network that was trained only with Hovernet probabilities believes it and so it creates three nests. The green in the middle, the red at the top right and the red at the bottom left grouping and merging the result from the Hovernet. The network trained with morphological features is more correct in that the group at the bottom left is correctly classified as green while preserving the red group at the top right. The GCN trained with everything is confused and so it simply predicts everything as green. Finally, the GCN that only sees the graph and no features at all behaves a bit unpredictably, creating another group of red cells at the bottom right and at the top left.

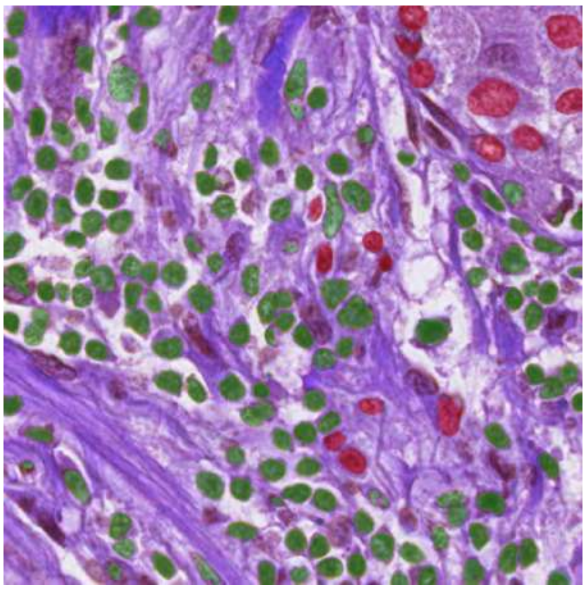
Similar conclusions can be drawn with the result obtained from the image on the right of Figure 4.6. The output is on Figure 4.8. The result with the remaining image at the center of Figure 4.6 is less informative. In there all the models correctly predicted every cell as red. Even though Hovernet sometimes randomly misclassifies little cells, there are cases like the one in the middle of Figure 4.6 that everything is correctly classified. When there is only one group to classify, the graph networks will be as right as Hovernet. After all, they are using the probabilities as inputs and creating clusters of cells. If there is only one cluster, there is not much work to do.


(a) GT


(b) Hovernet


(c) GCN


(d) Probabilities only


(e) Morphological features only

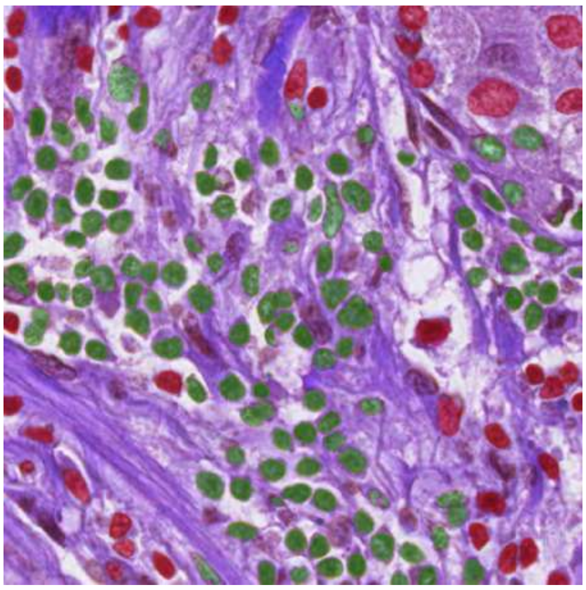
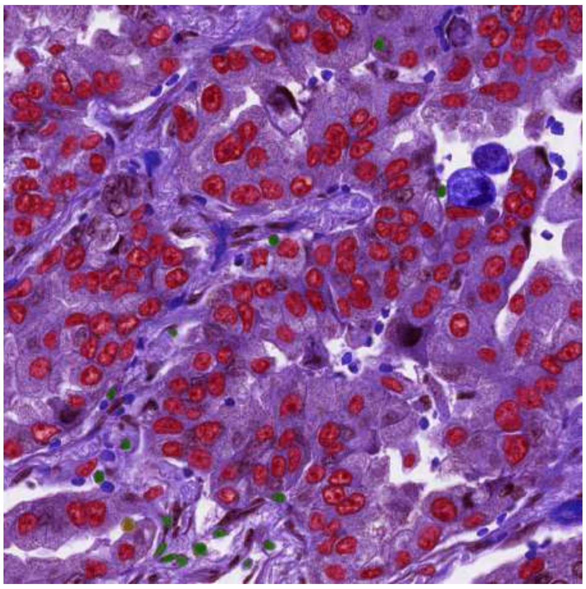
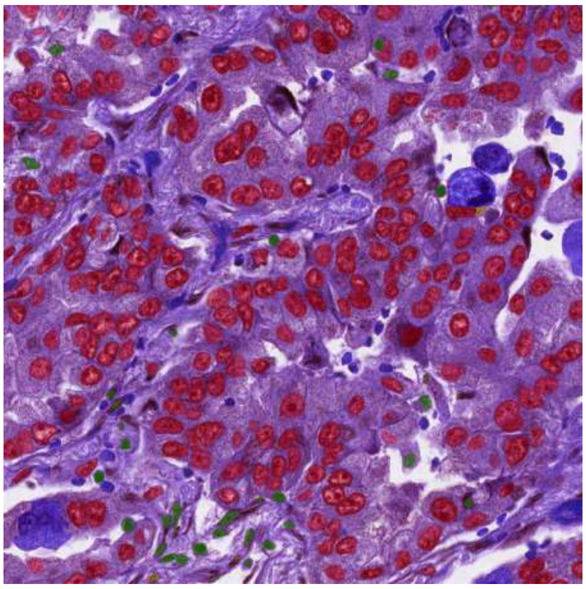
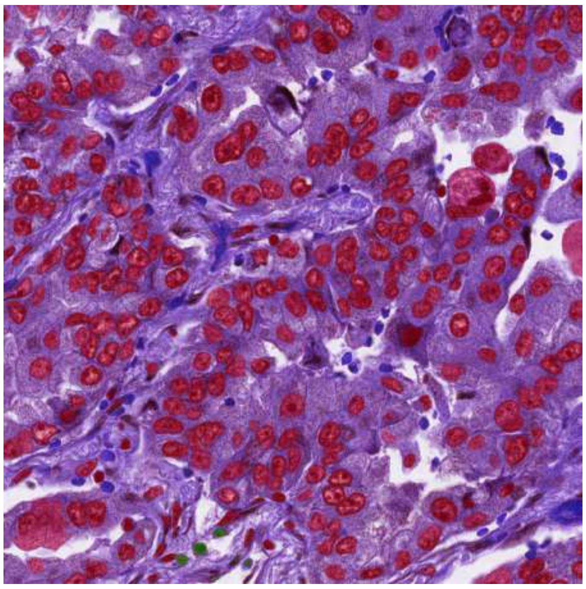
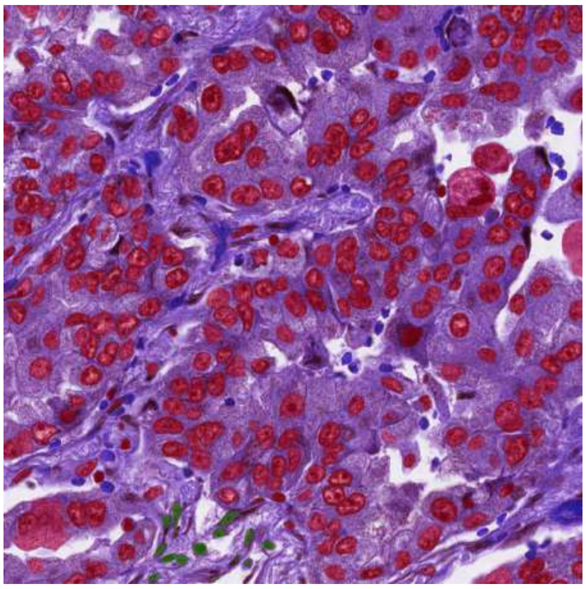

(f) Void

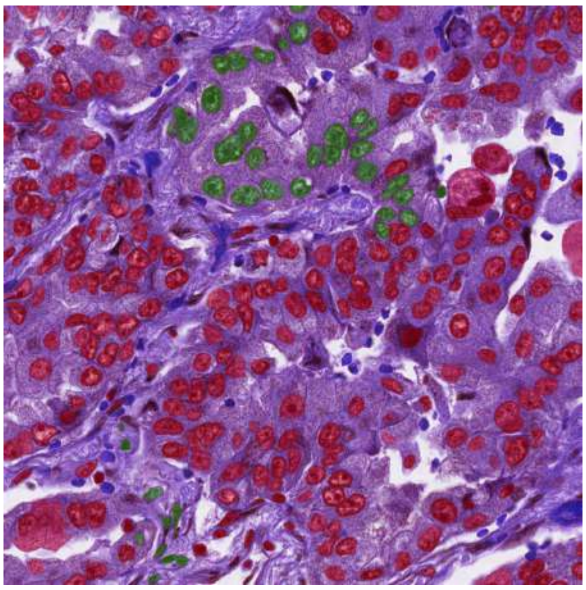
Figure 4.7: Illustration of the behaviour of the different models.


(a) GT


(b) Hovernet


(c) GCN


(d) Probabilities only


(e) Morphological features only

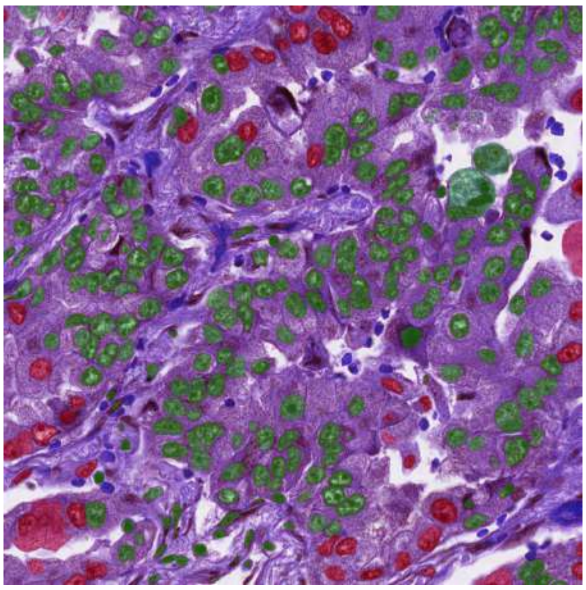

(f) Void

Figure 4.8: Another illustration of the behaviour of the different models.

4.2.3 DigiPatics breast

Let's start the qualitative analysis of this section with a question. Given the following three images with their ground truth on top. Which of them would you think is going to give less trouble to the graph neural networks? If you have read the previous two sections the answer should be obvious by now.

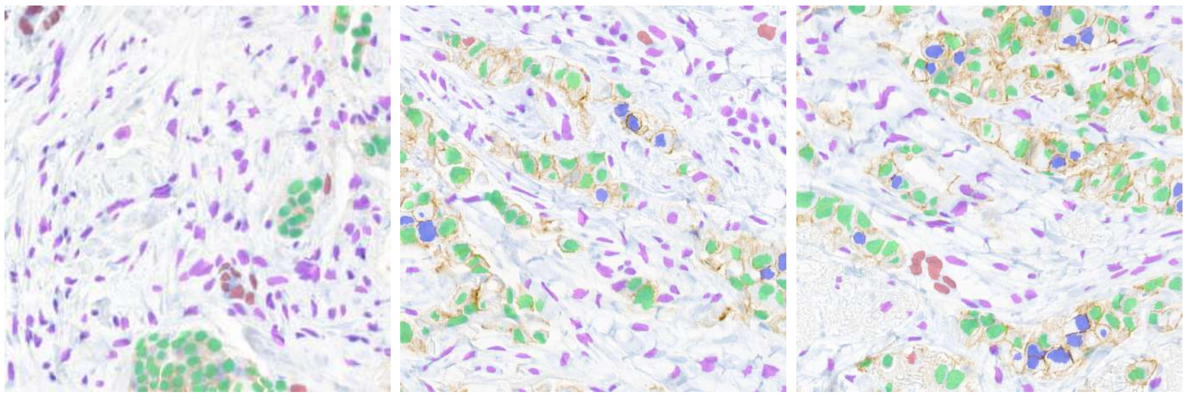


Figure 4.9: Three examples with their ground truth overlayed on top.

That's right, the left one. It has clearly identifiable groups. The other two also have nests of cells but in between there are cells of other classes. In the first image the nests are clearly defined and separated between each others. Now comes the second question. Watch the next image in Figure 4.10, which is the output of Hovernet, and try to predict which are going to be the groups detected by the GCN trained with all the features, with probabilities only, with morphological features only and with features at all.

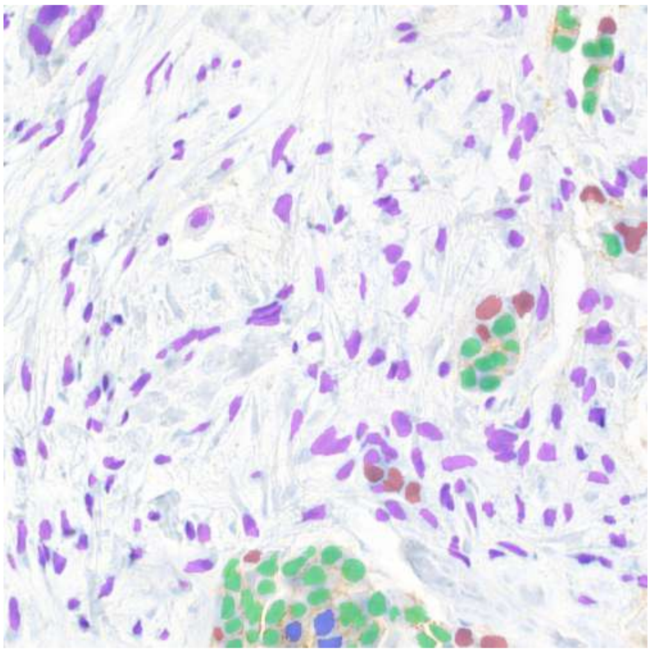
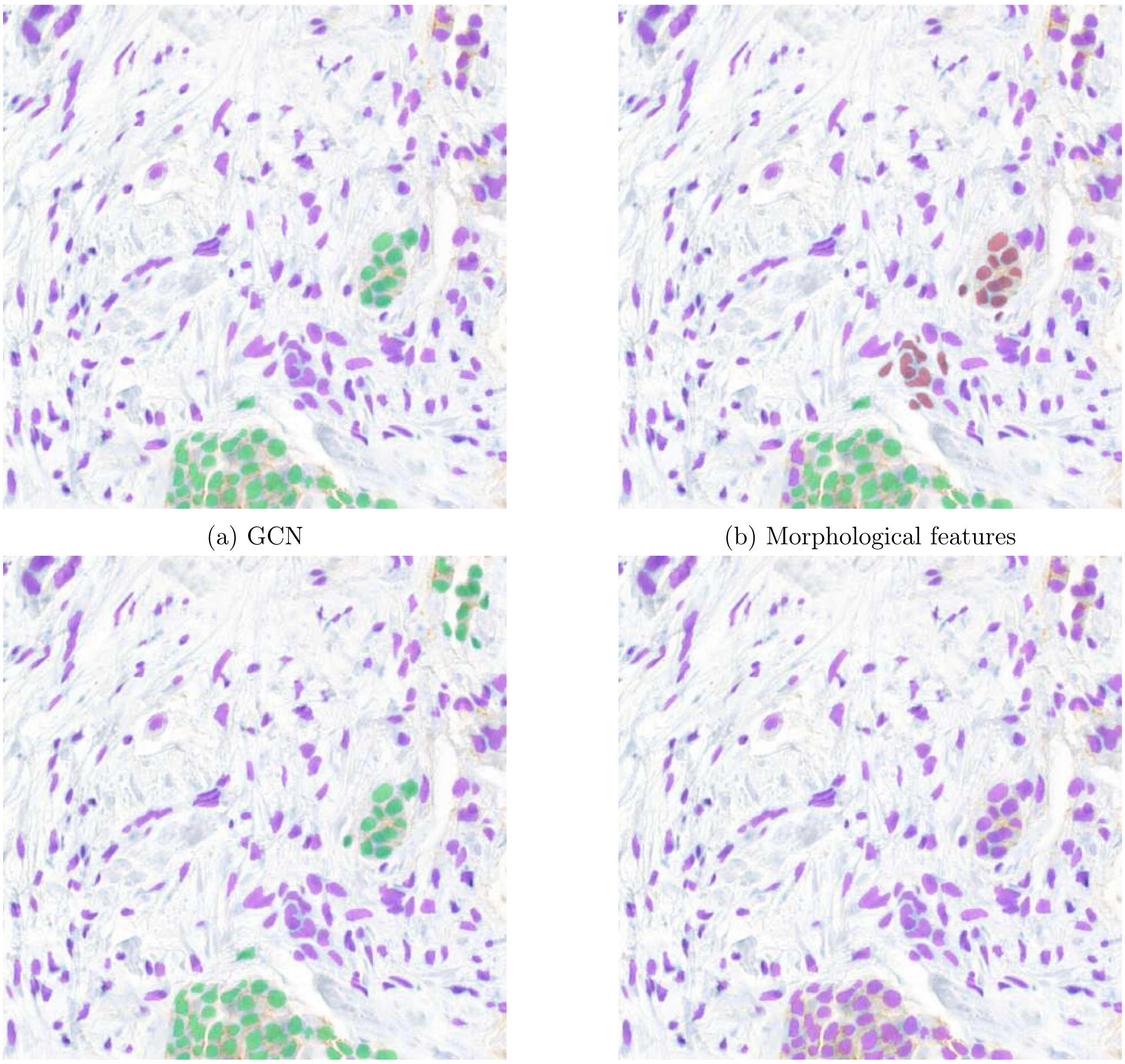


Figure 4.10: Output of Hovernet for the leftmost image of Figure 4.9.

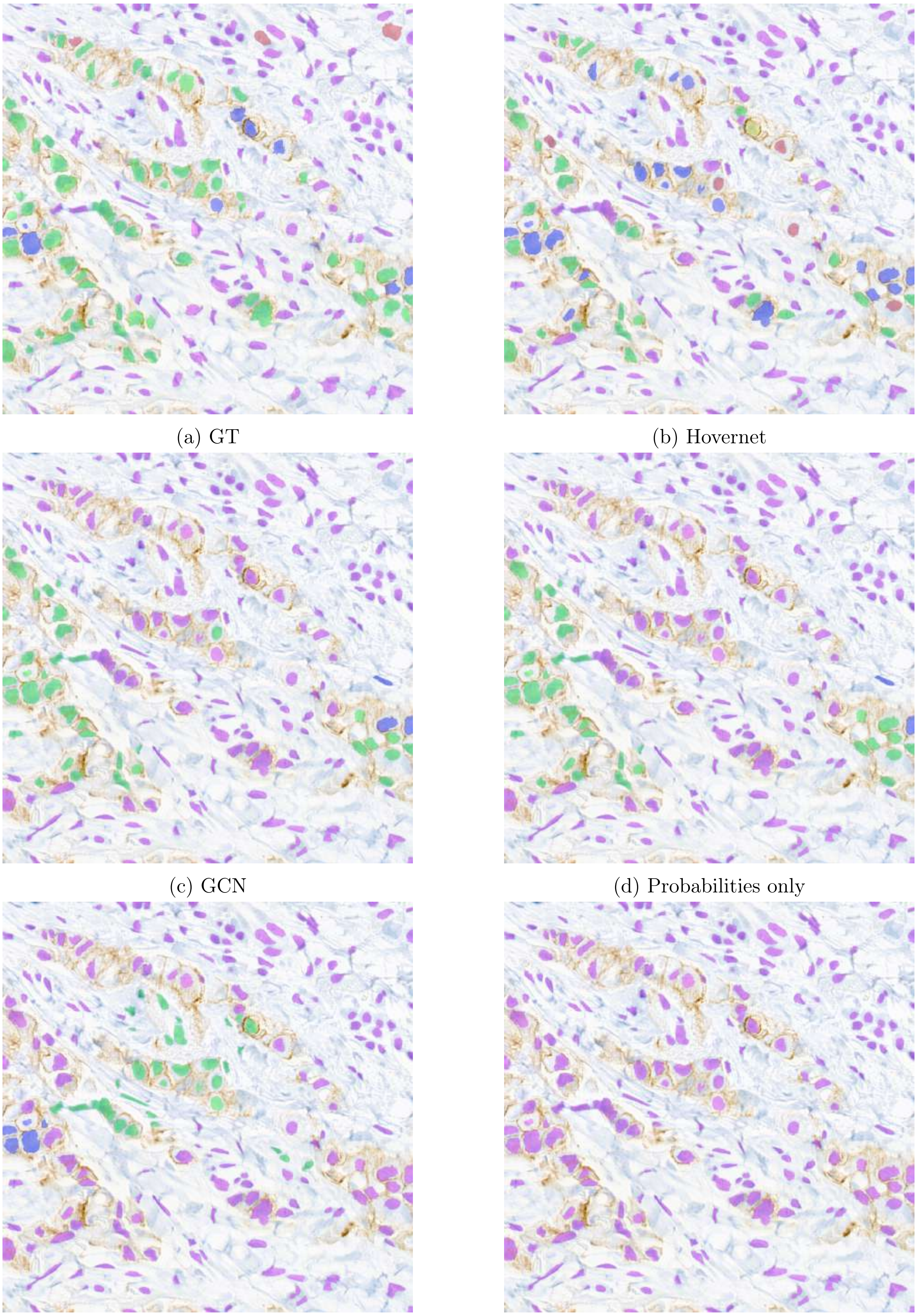
This question was more tricky. Although GNNs behave predictably, there is still margin for uncertainty. The GCN with all the features has made the green cells eat their neighbours although it missed the group at top right which is detected by the model trained only with probabilities. On the other hand, the model that does not have Hovernet probabilities has predicted some red cells where the other predicted red or magenta. And the void model has played the safest of them all by just classifying everything with the dominant class. Note that being safe does not mean being good. It is just a poetic license to state that it did not bother trying to identify minority groups.


(a) GCN (b) Morphological features

(c) Probabilities (d) Void

Figure 4.11: Output of the GCN trained with different sets of input features.

In Figure 4.12 and Figure 4.13 there are the outputs for the other two images from the beginning. Graph networks try to find clusters, sometimes they guess it right, sometimes they don't. Hovernet, as usual, tries to identify lonely cells correctly. Again, sometimes it works, sometimes it does not.



(a) GT

(b) Hovernet

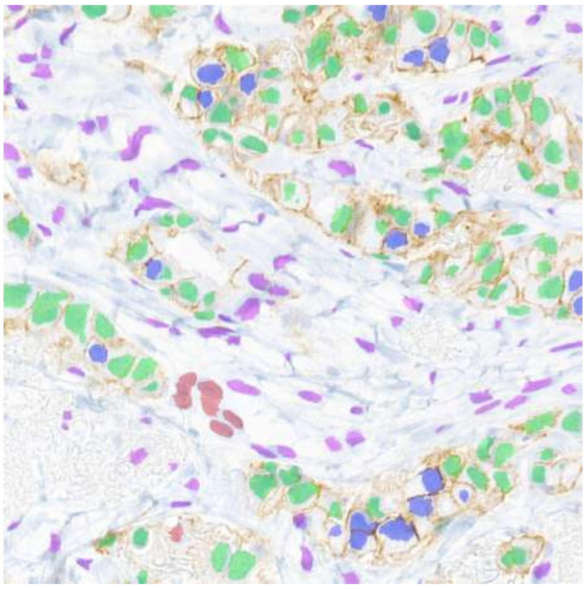
(c) GCN

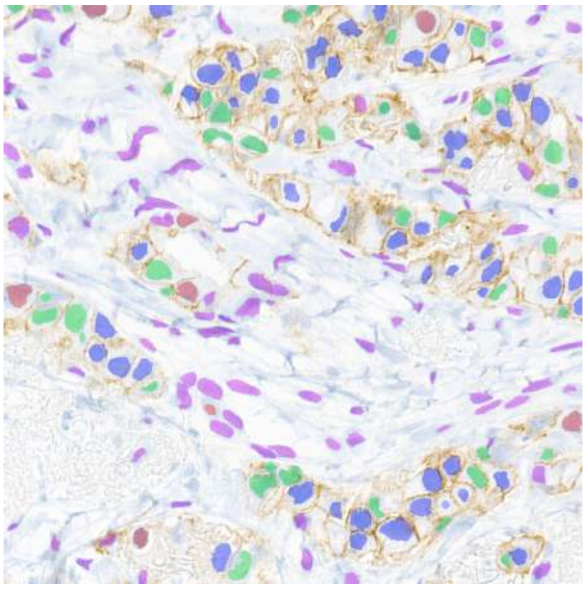
(d) Probabilities only

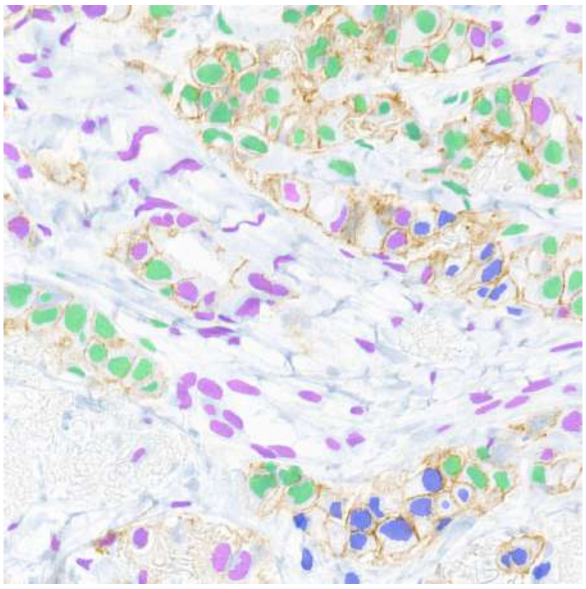
(e) Morphological features only

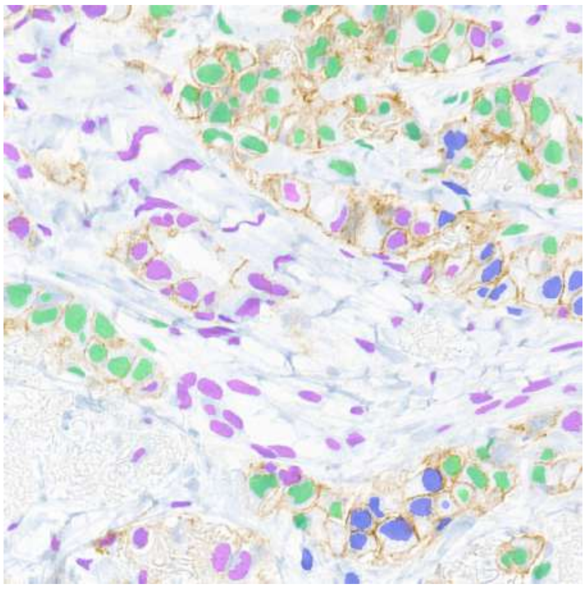
(f) Void

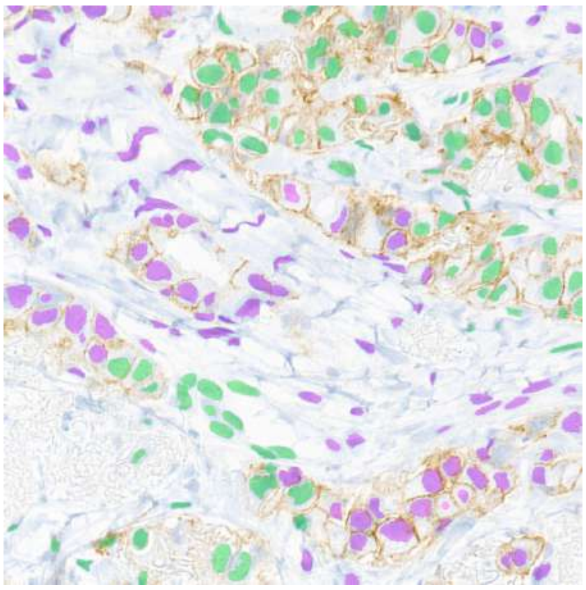
Figure 4.12: Illustration of the behaviour of the different models.


(a) GT


(b) Hovernet


(c) GCN


(d) Probabilities only


(e) Morphological features only

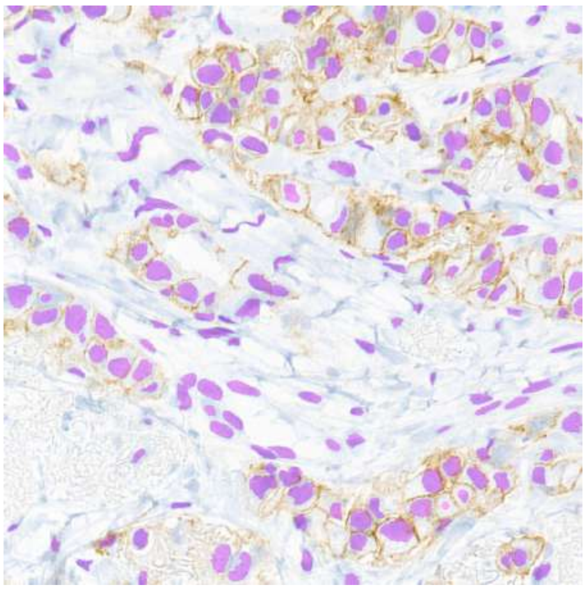
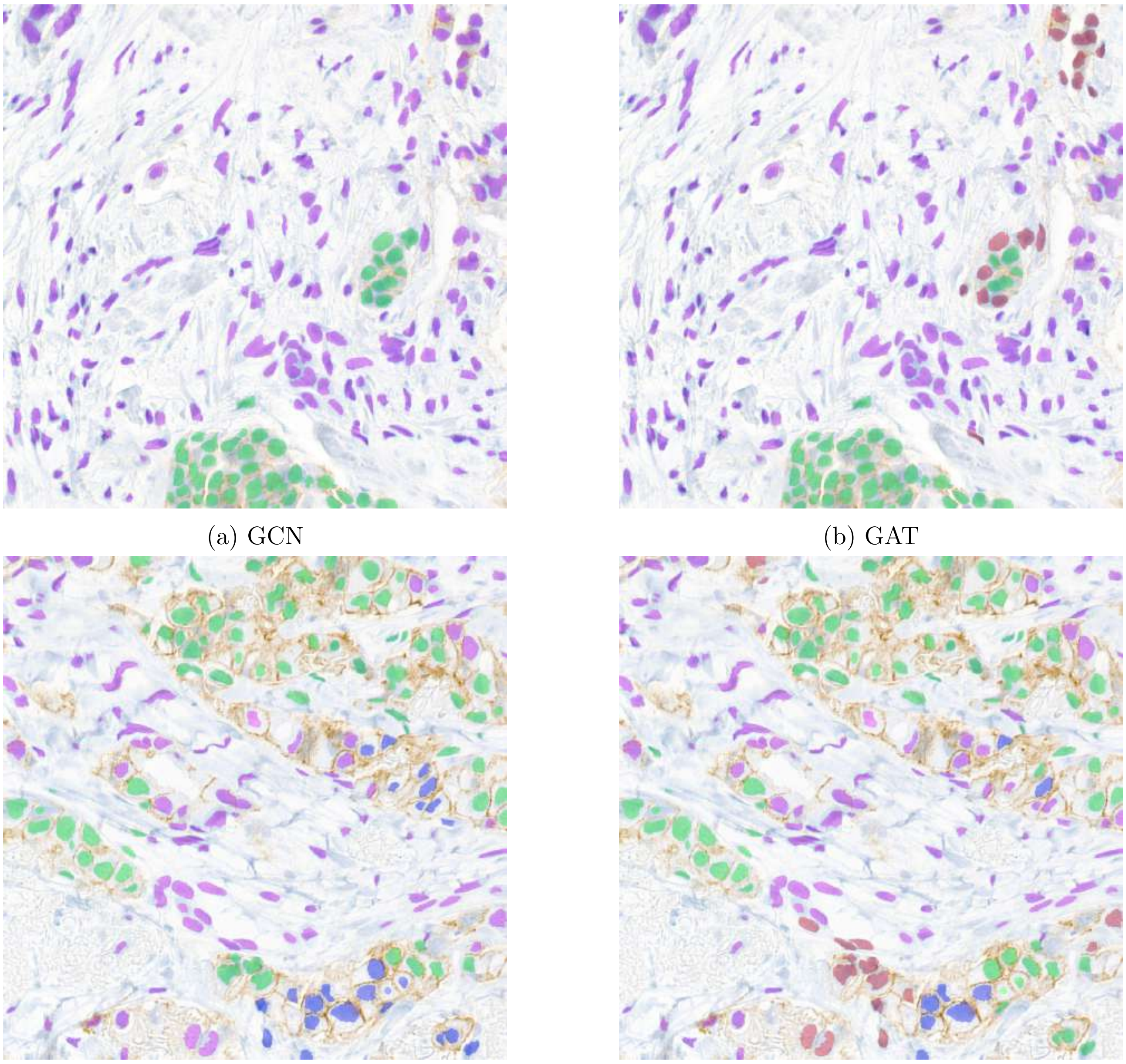

(f) Void

Figure 4.13: Yet another illustration of the behaviour of the different models.

To finish this section I will compare graph convolution to graph attention. Graph attention networks also create nests of cells. The difference now is that the attention network is trying to detect smaller groups and mixed groups. This may derive from the fact that the attention mechanism gives more flexibility in how labels are propagated through the graph. With the convolution everything gets spread equally likely. But with attention other patterns can emerge and, in fact, emerge. In Figure 4.14 we can see that in the first pair the graph attention has surrounded a green group by a red group. That is quite unlikely for convolution to do. With the convolution cells get eaten by the biggest group. In the second pair there are some blue cells that are alone in the GAT case and more group diversity. If we consider graph networks to act as regularisers, then GCN penalizes more the existence of lonely cells than GAT.


(a) GCN (b) GAT

(c) GCN (d) GAT

Figure 4.14: Comparison of graph convolution and graph attention predictions.

4.2.4 DigiPatics lung

I was personally involved in the creation of this dataset. This means that I know some information that is normally not documented in these databases. When labelling, the physician sometimes decided entirely based on information not present on the image itself. Or information present in the image but very subtle. One such case is the cilium. Cells near cilium are never tumoural but can be very similar to tumoural cells. In all the experiments we made, including those that are not documented here, the specific image in Figure 4.15 was always predicted as tumoural. However, our best performing graph model somehow solved that problem. It is possible that it has been pure luck. But it is still remarkable because we didn't have such luck with Hovernet.

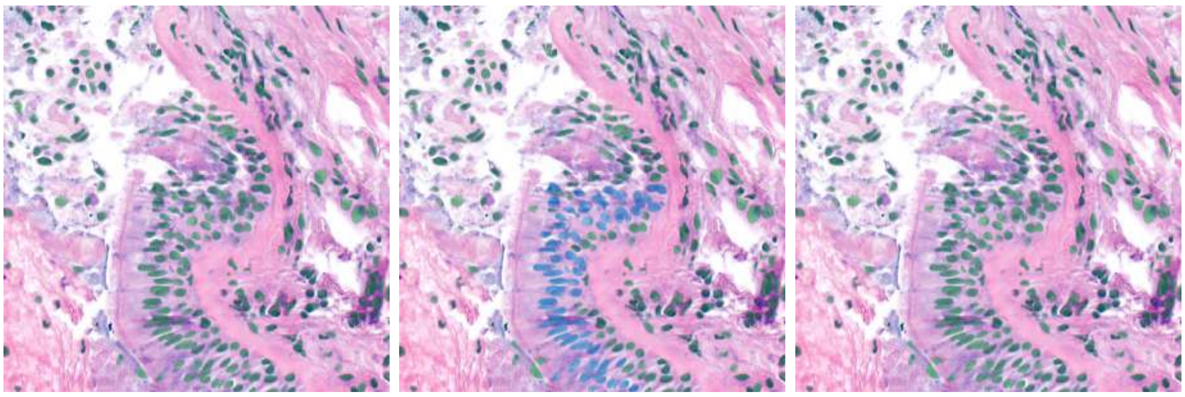

(a) GT (b) Hovernet (c) GCN without probabilities

Figure 4.15: Example of an image containing cilium together with Hovernet and GNN predictions.

Another special case is shown in Figure 4.16. When labelling this specific patch, Irene needed a few minutes to notice that it was not tumoural at all. And she discovered that by looking at the whole slide image. I was told that with that image alone it was not possible to be 100% sure that all the cells were not tumoural. And we can see that both Hovernet and the GNN fail in this case. The cells marked in blue look so similar to tumoural cells that the models classify them as that because they lack the information needed to discard them from being tumoural.

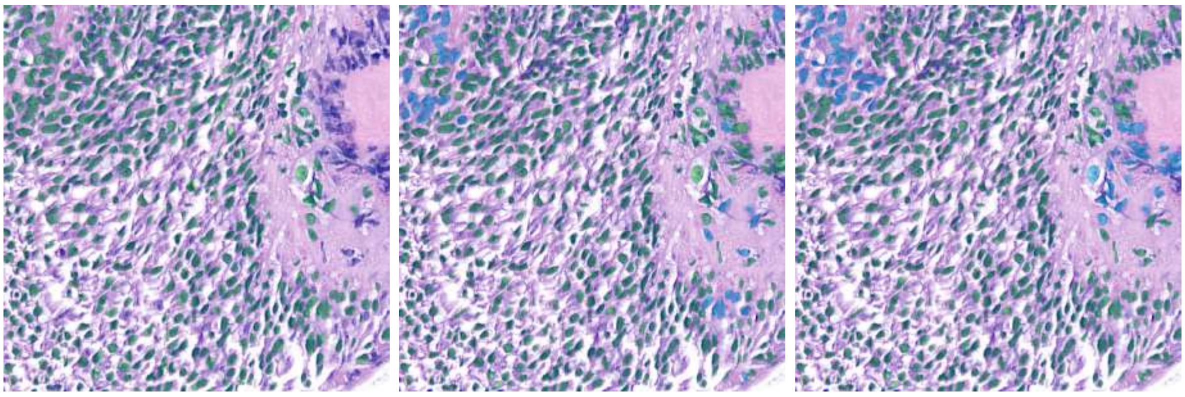

(a) GT (b) Hovernet (c) GCN without probabilities

Figure 4.16: Example of a patch that required extra context to label together with Hovernet and GNN predictions.

To finish this chapter I will leave a canonical example of when graphs are a good fit. The case depicted in Figure 4.17 contains three groups perfectly separated from each other. Two nests of tumoural cells at both sides and in the middle a group of healthy cells. Hovernet struggles to detect some of the tumoural cells from the left and right. Probably because they are small, round and with a low nuclei to cytoplasm ratio, which are the properties typical of non-tumoural cells. But they are indeed tumoural, in part because they are all so close together. The graph network, on the other hand, leverages the fact that they are grouped together and propagates the dominant class of each cluster to all the cells of that cluster, perfectly classifying the whole patch.

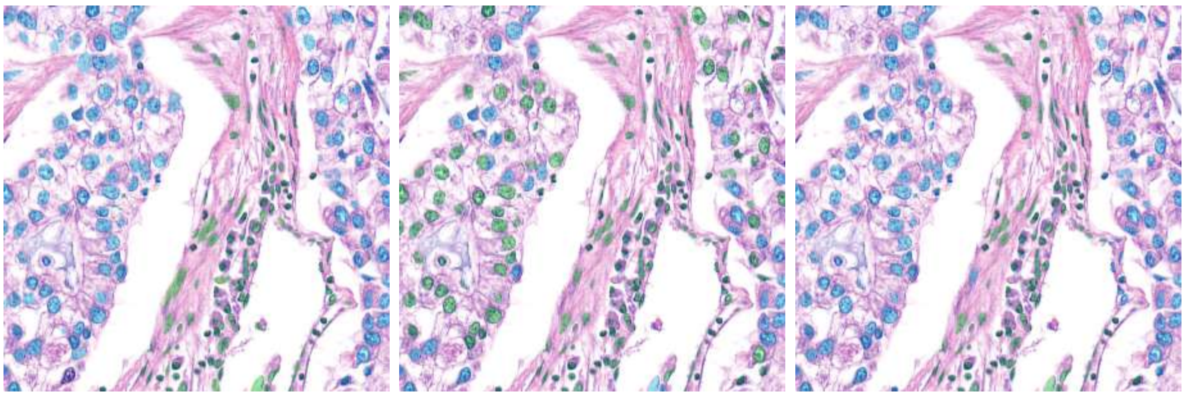

(a) GT (b) Hovernet (c) GCN without probabilities

Figure 4.17: Another example from the validation dataset where the graph network is clearly better than Hovernet.

Chapter 5

Conclusions

Digital pathology can be very difficult to tackle. Our experiments have showed that there is not a single method that works properly in every setup. Each organ and stain requires their own specific and carefully designed model. For lung and H&E stain the method proposed not only works better than previous state of the art models but our experiments have proven that it is properly leveraging extra information hidden to computer vision algorithms and tabular methods like XGBoost. Domain expertise was key for designing the algorithm. By talking to real experts on the topic I was able to discover how important the neighbourhood of a cell was to classify it as tumoural or not. Since in digital pathology data is so scarce, discovering such inductive biases can be very valuable. A simple vision transformer would easily solve this problem with billions of images. But we have hundreds. That is why the method presented in this thesis is so valuable, because of its sample efficiency.

Chapter 6

Future work

The obvious lines of research here are improving the CNN backbone, improving the GNN head or training everything end-to-end instead of having two-phases as in this work. All of them would bring better metrics and more robust classification methods. However, the obvious is not always the best. Progress is made by impressive discoveries that few people expected. That is why I want to propose a different way of advancing science in this field. The classification paradigm has served us well, but a new paradigm is emerging nowadays. Multimodal models are on the rise. Image captioning models like BLIP [18] or CLIP [29] have gained popularity in the last years. They provide a way of opening the black boxes that are neural networks. Captioning an image is just one step behind a model explaining itself and the reasoning about the result. Why not make a model explain cancer? Expert pathologist train new physicians showing them images and explaining them with voice or text. Why not train models the same way? Obviously those models won't be better than the experts, but they can outscale them. One person can only view a small amount of patients in a life. A model can be trained with billions of cases in months and can infer the result of millions of new cases in days given enough computational resources. So that is the non-obvious future work: Image captioning for medical image analysis.

I'll use the rest of the page to describe the roadmap to achieving the goal of medical images explaining themselves. First step: data. That is the most important part of the whole project. No data no models. Typically foundational models require billions of image-text pairs, or if only trained with text they require trillions of tokens. Such amount of medical data is not available yet, although it would be possible if experts were into it. But I'll assume physicians don't really want to spend time creating datasets (which is more or less my experience dealing with doctors). The good news is that one can fine-tune a foundational model with smaller amounts of data. With thousands of image-text pairs would be enough for a production ready model. To obtain such pairs pathologist would be required to think loud. That's it. We can automatically transcribe audio into text, so it is enough for them to record their voice while working. The next step is the model. In the previous paragraph I have shown two methods of jointly training a latent text-image space. But we can achieve much more than that, we want to speak to the images. Something like what is done in Mini-GPT4 [44]. Right now there is a wide variety of language models to be used as backbones. Stability AI recently released StableLM[1], and we have the LLaMa family of models too[2]. We are not running out of open source LLMs anytime soon. Using them is a huge boost in performance and sample efficiency. Moreover, there is also DINOv2[3] which can be used as image encoder to further reduce the burden of training. We just need to merge everything and finetune it with the appropiate data. Sounds easy, right?

[1]https://github.com/Stability-AI/StableLM
[2]https://ai.facebook.com/blog/large-language-model-llama-meta-ai/
[3]https://dinov2.metademolab.com/

References

[1] Sil (https://math.stackexchange.com/users/290240/sil). *Injectivity of given integer function.* Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/4567383 (version: 2022-11-02). eprint: https://math.stackexchange.com/q/4567383. URL: https://math.stackexchange.com/q/4567383.

[2] Abien Fred Agarap. *Deep Learning using Rectified Linear Units (ReLU).* 2019. arXiv: 1803.08375 [cs.NE].

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate.* 2014. DOI: 10.48550/ARXIV.1409.0473. URL: https://arxiv.org/abs/1409.0473.

[4] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. "Gephi: an open source software for exploring and manipulating networks". In: *Third international AAAI conference on weblogs and social media.* 2009.

[5] Michael M. Bronstein et al. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges". In: *CoRR* abs/2104.13478 (2021). arXiv: 2104.13478. URL: https://arxiv.org/abs/2104.13478.

[6] Nadav Cohen and Amnon Shashua. "Inductive Bias of Deep Convolutional Networks through Pooling Geometry". In: *CoRR* abs/1605.06743 (2016). arXiv: 1605.06743. URL: http://arxiv.org/abs/1605.06743.

[7] Jieneng Chen et al. *TransUNet: Transformers Make Strong Encoders for Medical Image Segmentation.* 2021. arXiv: 2102.04306 [cs.CV].

[8] Tianqi Chen and Carlos Guestrin. "XGBoost". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, Aug. 2016. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145%2F2939672.2939785.

[9] Xiaodong Chen, Bin Zheng, and Hong Liu. "Optical and Digital Microscopic Imaging Techniques and Applications in Pathology". In: *Analytical Cellular Pathology* 34.1-2 (2011), pp. 5–18. DOI: 10.1155/2011/150563. URL: https://doi.org/10.1155/2011/150563.

[10] Jevgenij Gamper et al. *PanNuke Dataset Extension, Insights and Baselines.* 2020. arXiv: 2003.10778 [eess.IV].

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. "Deep Learning". In: http://www.deeplearningbook.org. MIT Press, 2016. Chap. 6.2.2.3.

[12] Simon Graham et al. *HoVer-Net: Simultaneous Segmentation and Classification of Nuclei in Multi-Tissue Histology Images.* 2019. arXiv: 1812.06499 [cs.CV].

[13] Geoffrey E. Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *CoRR* abs/1207.0580 (2012). arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580.

[14] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[15] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2016. DOI: 10.48550/ARXIV.1609.02907. URL: https://arxiv.org/abs/1609.02907.

[16] Alexander Kirillov et al. *Segment Anything*. 2023. arXiv: 2304.02643 [cs.CV].

[17] Neeta Kumar, Ruchika Gupta, and Sanjay Gupta. “Whole Slide Imaging (WSI) in Pathology: Current Perspectives and Future Directions”. In: *Journal of Digital Imaging* 33.4 (May 2020), pp. 1034–1040. DOI: 10.1007/s10278-020-00351-z. URL: https://doi.org/10.1007/s10278-020-00351-z.

[18] Junnan Li et al. *BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation*. 2022. arXiv: 2201.12086 [cs.CV].

[19] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. “Rectifier Nonlinearities Improve Neural Network Acoustic Models”. In: *ICML Workshop on Deep Learning for Audio, Speech and Language Processing* (2013).

[20] Gledson Melotti et al. *Reducing Overconfidence Predictions for Autonomous Driving Perception*. 2022. arXiv: 2202.07825 [cs.CV].

[21] Lassi Meronen et al. *Fixing Overconfidence in Dynamic Neural Networks*. 2023. arXiv: 2302.06359 [cs.LG].

[22] Fernand Meyer. *The watershed concept and its use in segmentation : a brief history*. 2012. DOI: 10.48550/ARXIV.1202.0216. URL: https://arxiv.org/abs/1202.0216.

[23] Vincent Micheli, Eloi Alonso, and François Fleuret. *Transformers are Sample-Efficient World Models*. 2023. arXiv: 2209.00588 [cs.LG].

[24] Ben Mildenhall et al. *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. 2020. arXiv: 2003.08934 [cs.CV].

[25] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. *V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation*. 2016. arXiv: 1606.04797 [cs.CV].

[26] *MUC4 ’92: Proceedings of the 4th Conference on Message Understanding*. McLean, Virginia: Association for Computational Linguistics, 1992. ISBN: 1558602739.

[27] National Electrical Manufacturers Association. *Digital Imaging and Communications in Medicine (DICOM) Standard*. Technical Report. [Online; accessed 4-April-2023]. National Electrical Manufacturers Association. URL: https://www.dicomstandard.org/.

[28] Julia Sala Prat. *Cell detection and classification of breast cancer histology images using a deep learning approach based on the U-Net architecture*. [Online; accessed 2-April-2023]. June 2021. URL: https://upcommons.upc.edu/handle/2117/353765.

[29] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV].

[30] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[31] Philippe Salembier, Albert Oliveras, and Luis Garrido. “Antiextensive connected operators for image and sequence processing”. In: *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society* 7 (Feb. 1998), pp. 555–70. DOI: 10.1109/83.663500.

[32] Irwin Sobel. “An Isotropic 3x3 Image Gradient Operator”. In: *Presentation at Stanford A.I. Project 1968* (Feb. 2014).

[33] Jordi Temprana-Salvador et al. “DigiPatICS: Digital Pathology Transformation of the Catalan Health Institute Network of 8 Hospitals;Planification, Implementation, and Preliminary Results”. In: *Diagnostics* 12.4 (2022). ISSN: 2075-4418. DOI: 10.3390/diagnostics12040852. URL: https://www.mdpi.com/2075-4418/12/4/852.

[34] Jeya Maria Jose Valanarasu and Vishal M. Patel. *UNeXt: MLP-based Rapid Medical Image Segmentation Network*. 2022. arXiv: 2203.04967 [eess.IV].

[35] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[36] Petar Veličković et al. *Graph Attention Networks*. 2017. DOI: 10.48550/ARXIV.1710.10903. URL: https://arxiv.org/abs/1710.10903.

[37] Ruchika Verma et al. “MoNuSAC2020: A Multi-Organ Nuclei Segmentation and Classification Challenge”. In: *IEEE Transactions on Medical Imaging* 40.12 (2021), pp. 3413–3423. DOI: 10.1109/TMI.2021.3085712.

[38] Analytics Vidhya. *The Curse of Dimensionality in Machine Learning*. https://www.analyticsvidhya.com/blog/2021/04/the-curse-of-dimensionality-in-machine-learning/. [Online; accessed 2-April-2023]. 2021.

[39] Zifu Wang et al. *Dice Semimetric Losses: Optimizing the Dice Score with Soft Labels*. 2023. arXiv: 2303.16296 [cs.CV].

[40] Hongxin Wei et al. *Mitigating Neural Network Overconfidence with Logit Normalization*. 2022. arXiv: 2205.09310 [cs.LG].

[41] Zonghan Wu et al. “A Comprehensive Survey on Graph Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (Jan. 2021), pp. 4–24. DOI: 10.1109/tnnls.2020.2978386. URL: https://doi.org/10.1109/tnnls.2020.2978386.

[42] Jiaxuan You, Rex Ying, and Jure Leskovec. *Design Space for Graph Neural Networks*. 2021. arXiv: 2011.08843 [cs.LG].

[43] Hattie Zhou et al. *Fortuitous Forgetting in Connectionist Networks*. 2022. arXiv: 2202.00155 [cs.LG].

[44] Deyao Zhu et al. *MiniGPT-4: Enhancing Vision-Language Understanding with Advanced Large Language Models*. 2023. arXiv: 2304.10592 [cs.CV].

Appendix A

Sustainability and costs

This project was carried out by a single student, which means the cost of it was very low. However, a project that depends on students is not a sustainable one since students are highly underpaid. For that reason I would like to provide a quantitative analysis on the real costs of carrying out this work.

Let's start counting the hours of work required to just build the dataset. Even after using a semi-automatic procedure to label the data, many hours were needed to end up with 85 labelled images. Roughly 100 hours of human labour and 20 hours of compute power were employed to create segmentations. Another 15 hours of expert human labour were required to review the class annotations by the physician. Now, let's translate that into money. The first 100 hours required very low training, just the ability to paint circles. A fair salary for that can be 10€/h (if outsourced it could be less than 2€/h). That translates into a cost of 1000€. The compute power requires GPUs that consume at most 350W. Let's estimate the CPU, memory and GPU consumption by 300W on average, which means 4.5kWh were consumed. That means 0.93€ if the price of electricity is 0.15€/kWh. And the expert work is typically paid more, let's estimate 20€/h which means 300€ in total. The total cost of creating the database would be 1300.93€. If we divide by the number of images we have a total of 15.30€ per image.

Second, the salary of the technical workers. A project like this normally requires a software developer, a data engineer and a project manager. Depending on the experience the salaries may range. For simplicity, let's consider the software developer is junior (30K/year), the data engineer is senior (50K/year) and the project manager has nearly five years of experience (80K/year). Assuming the project is carried out in just one year, and taking into account the extra 30% that needs to be payed to the social security, the total cost in salaries would be 208000€.

Third, putting a machine learning model in production is more than just creating it. It needs to be maintained, and someone has to care about the drift. Models performs worse when time passes because the data distribution is not fixed. New patients means new training is needed. Hospitals would need to train their technicians to perform retrainings and physicians to relabel. The cost of relabelling is already estimated: 15.30€ per image. Remains to estimate the cost of training technicians. 385$ or 350€ per person is what a course on machine learning would cost[1].

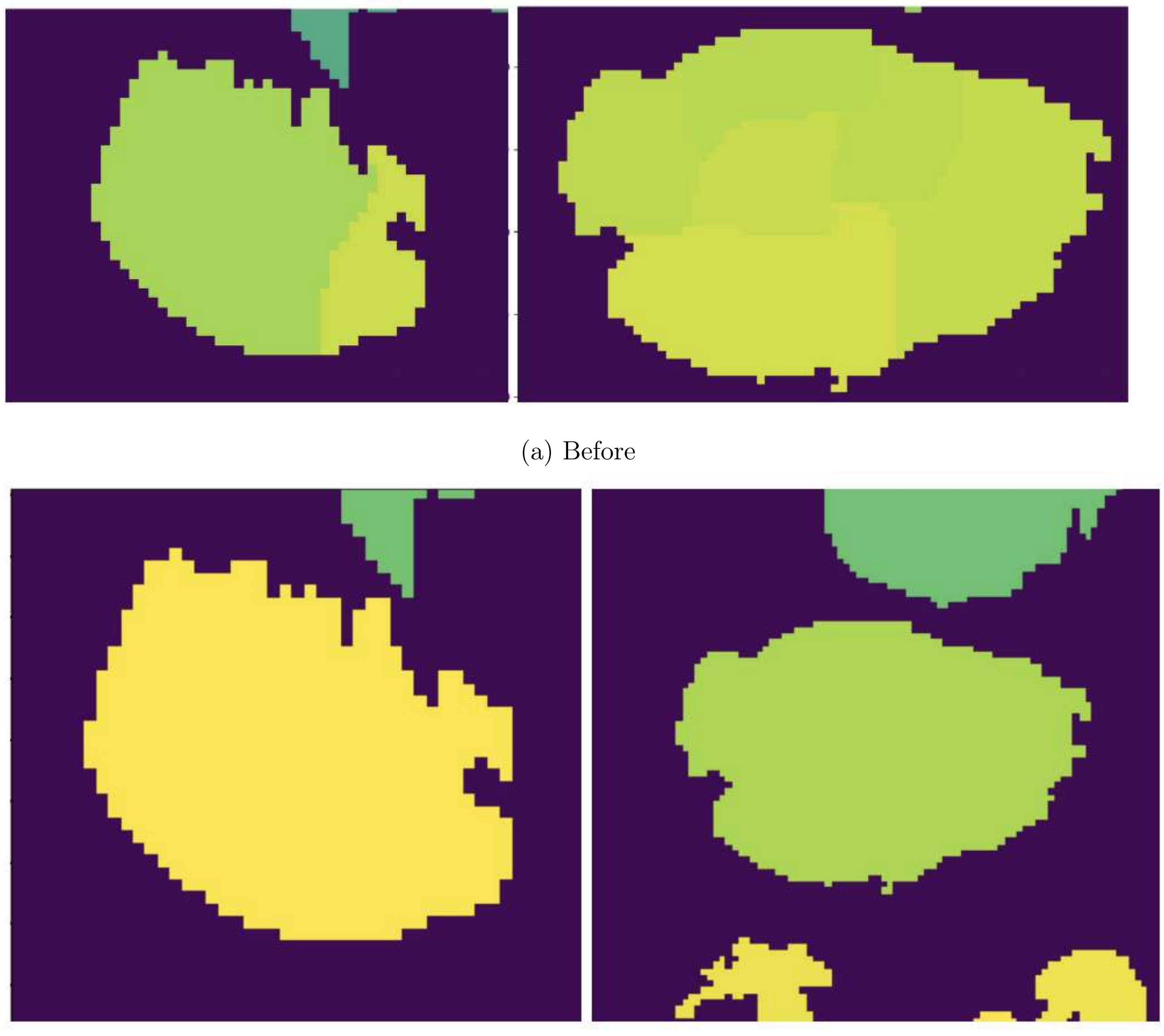
Finally, to show that the project is sustainable, it has to generate some value. I believe in open science and medicine. Putting a price in other people's lives is not ethical, so privatising this project is not an option. But someone has to pay the cost, in this case, the government. A project like this is only sustainable if public opinion is in favour of it. More data is needed to exactly estimate the social value this will bring. In my opinion, this product will reduce physician workloads, which is beneficial. By reducing the workload, the diagnostic process can be accelerated, and receiving faster diagnostics is something the public opinion will definitely be interested about.

[1]https://www.ml.school/c/start-here Accessed 8th April 2023

Appendix B

The problem of merging cells

One of the problems that appeared when using Hovernet was broken cells. In many cases, big cells were predicted as various small cells. It is visually appalling and that is why we designed an algorithm to merge broken cells. The result is on Figure B.1.



(a) Before

(b) After

Figure B.1: Example of cells that were not fully detected by Hovernet. After applying our algorithm all the smaller cells were merged into one.

The algorithm leverages techniques of mathematical morphology. If we call X_1 the predicted image and X_2 the image that identifies the background with the value zero and the rest with one, then, the image $(\delta(X_1) - X_1) \cdot X_2$ contains the frontiers of the cells, being δ a dilation. An illustration of the process is on Figure B.2.

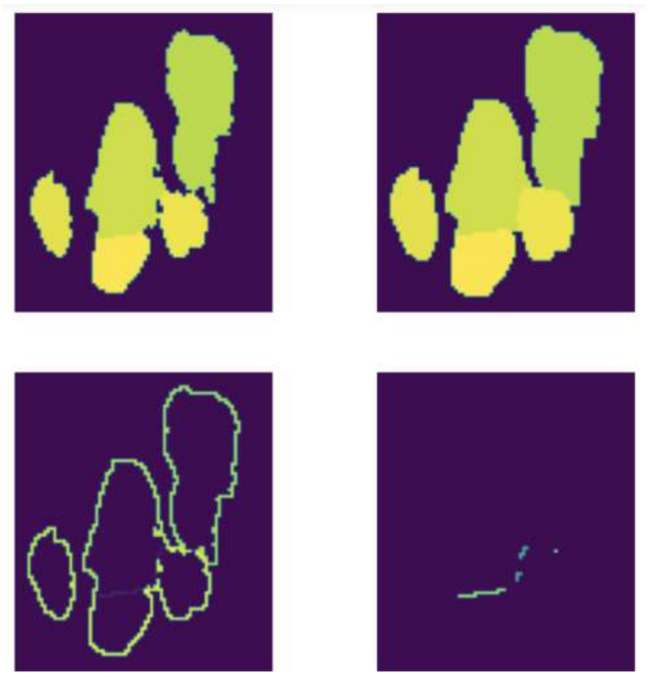


Figure B.2: Top-left is the original image, top-right is the dilation, bottom-left the gradient and bottom-right the masking which is the final result.

The problem here is that different frontiers may end up with the same identifier. X_1 contains the identifier of each cell in every pixel of such cell. But, $(\delta(X_1)-X_1)\cdot X_2$ contains the differences of identifiers in the position of the frontier. Differences of different pairs may end up with the same value. E.g.: $4 - 3 = 2 - 1$. To solve that, it is needed to properly set the identifiers so that each pair can be uniquely identified by the difference between the maximum and the minimum. One simple solution to that is to use powers of two. More concretely, applying the function $n \mapsto 2^n$ to the identifiers. This way $2^n - 2^m$ is a function that can be inverted back to the pair (n, m) where $n > m$. However, we can have up to 1500 cells, requiring identifiers up to 2^{1500} is clearly unfeasible and is not computationally optimal since to identify pairs we only need $\binom{1500}{2} = 1.124.250$ identifiers. The best function I have found so far is $n \mapsto n^5$, using other simple lower degree polynomials doesn't uniquely identify the difference for values up to 1500 [1].

Appendix C

TumourKit

The whole thesis dealt with the theory and the results of the experiments. Behind all that there was a lot of software needed to make it all work. I decided to build a python library that other people could use to replicate my experiments and design new ones. The library is called TumourKit. The code is available on GitHub[1] under the Affero GPLv3 license. It is tested on Ubuntu and Windows for python versions 3.8, 3.9 and 3.10. It has more than ten thousands lines of code, counting comments. For a more detailed explanation on how to use it and what is offered, please, read the docs[2].

[1] `https://github.com/Jerry-Master/lung-tumour-study`
[2] `https://lung-tumour-study.readthedocs.io/en/latest/`

Appendix D

Soft Labels

One of the problems we encountered when reviewing the lung dataset was that the expert was not sure which label to give to some cells. It was solved by simply ignoring those cells since in practice it didn't really matter because we are interested in percentages and a cell with no label can be removed from the numerator and denominator. However, we do preserve that information in the dataset because it could be used to train the models with soft labels. Even though we didn't try it, we expect it could help further calibrate the resulting probabilities. For that reason, in this appendix I will be explaining how to adapt all the methods presented in the thesis to soft labels.

The easiest ones to adapt are the graph neural networks. They are trained using the softmax, so they are already compatible with soft labels. It is only needed to substitute the one hot encoded vector of labels to a vector of probabilities and everything works exactly the same.

The XGBoost method requires a little trick. Although XGBoost also uses softmax for training, current implementations threshold the label so providing probabilities is of no use since the soft labels will be made into hard labels. Nonetheless, XGBoost supports a weight for each row in the dataset. This makes it possible to use the probabilities as weights. The trick consists of repeating rows one time for each class and assign that class probability to that repeated row weight. The idea was taken for an online forum[1] and the explanation behind how this works is that XGBoost multiplies the gradient and hessian by the weight and not by the label, as explained in another online forum[2].

The trickiest method to adapt is HoVerNet. Having soft labels does not affect neither the HV branch nor the NP branch but it does affect the NC branch. That branch is trained using the softmax and the dice loss. The first one is quite easy to adapt as described for the graph neural networks. However, the dice loss requires more thinking. In Zifu Wang et al. [39] they propose a way of extending the dice loss for soft labels with promising results. On a more technical and practical level, modifying the codebase to support this metric is too much of a hurdle given the expected benefits. But it is interesting on a theoretical perspective to know of the existence of such possibility. In the future, when it becomes a more refined technique with wider support on the main libraries, it may be a good way of training better models.

[1] `https://stackoverflow.com/a/66481600`

[2] `https://stats.stackexchange.com/a/365555/378093`

Appendix E

Hyperparameter study

Instead of showing the losses here which would be pretty useless since there are 32 configurations per dataset and for each configuration we track 5 or 6 metrics. That much information cannot be shown in a non-interactive way. Therefore, we provide public links to the tensorboard logs with all the training and validation metrics for every configuration. This way you can see how every configuration evolved in time and filter by hyperparameters. The naming convention is as follows. The first letters are either GCN or ATT meaning graph convolution of attention. Then, the next number is the number of layer, after that is the dropout value and then whether or not it used batch normalisation. So, ATT_10_0.3_bn means graph attention with 10 layers and 30% of dropout with batch normalisation. The links are below, if you do not see anything it may be because you need to select some configurations in the left bar.

- DigiPatics lung: https://tensorboard.dev/experiment/HVSvkl8LQyuNWIWw4dgNmg/
- DigiPatics breast: https://tensorboard.dev/experiment/cQoaRA5GRZSEdYh3ijuPmw/
- CoNSeP: https://tensorboard.dev/experiment/Zb6YbBWWSO6q17DdMEVtZw/
- MoNuSAC: https://tensorboard.dev/experiment/rfvi7c48Q7CEU7yNvdxEGg/

I am also providing links to the tensorboard logs of every void gnn experiment. For DigiPatics lung:

- Probabilities: https://tensorboard.dev/experiment/3Q9pt4KlSvWiv6xAeEE3cA/
- Morphological: https://tensorboard.dev/experiment/hgMqRfVbRr2XWs0xKmX2sQ/
- Void: https://tensorboard.dev/experiment/Ko52fAFqTRyKXwqXRQx4gQ/

For DigiPatics breast:

- Probabilities: https://tensorboard.dev/experiment/tVB2jfeWR4yWuolpKaEBFg/
- Morphological: https://tensorboard.dev/experiment/o1QXFlsMSPGo6zmVNY7f1g/
- Void: https://tensorboard.dev/experiment/IMC4U8y9R7KXzwaJMlm2Qg/

For CoNSeP:

- Probabilities: https://tensorboard.dev/experiment/uRu2tXp0Rr6HKVJ7bend0w/
- Morphological: https://tensorboard.dev/experiment/tBlaEjlqRJyTq5NfMGMWEg/
- Void: https://tensorboard.dev/experiment/aKsWfarRTYmw1CGyZY37RA/

For MoNuSAC:

- Probabilities: https://tensorboard.dev/experiment/EPvvFnj1SGC5AgE7geJmRA/

- Morphological: https://tensorboard.dev/experiment/gS9WVXkHQTapbAZNT21OjQ/
- Void: https://tensorboard.dev/experiment/e61LbjeZRwCO4Avz2Ar03A/

The best configuration of hyperparameters for each dataset can be found on Table E.1 and the test metrics of every configuration are presented in the remaining tables of this appendix. Keep in mind that these metrics are on the test set and provided only for curious readers. They were never used in the selection of the model nor configuration since we only used the validation set for that. In fact, you can see that the best metrics here are not always the same as the metrics presented. Also for curious readers, more tensorboard links are provided from the training of Hovernet below. Tensorboard dev does not support image visualisation so, even though I have image logs they cannot be shared easily. If you are interested on them, feel free to email me for it. Here, 00 is the first phase of the training where the encoder was frozen and 01 is the second phase of training which was done after 00 and had the encoder unfrozen.

- DigiPatics lung: https://tensorboard.dev/experiment/Ah3wt2t9Tl6ArIQ1qqGXDQ/
- DigiPatics breast: https://tensorboard.dev/experiment/W416onQiQbaAkWpplkm4TA/
- CoNSeP: https://tensorboard.dev/experiment/vcz3QWa2SouqvqYOGpVpXg/
- MoNuSAC: https://tensorboard.dev/experiment/3g6kp9xmTa6a6jKcxUAJcw/

The main takeaway from all this information is that there is no golden rule. Do not try to find any pattern, there is not any. Sometimes more layers is better, sometimes is not. Sometimes dropout works sometimes it does not. You always need to optimise for those hyperparameters since every problem is different and no intuition can be obtained from empirical data about the behaviour of those hyperparameters. If you want better results you are forced to use more compute power.

Table E.1: Best hyperparameter configurations.

Dataset	#Layers	Dropout	Batch Normalisation
DigiPatics lung	5	0.9	Yes
DigiPatics breast	1	0.6	Yes
CoNSeP	1	0.0	Yes
MoNuSAC	15	0.3	No

Table E.2: Graph convolution hyperparameter optimization on DigiPatics lung.

F1	Acc	AUC	%ERR	ECE	#Layers	Dropout	Norm type
62.77%	82.67%	84.94%	6.96%	0.0711	1	0.0	bn
69.44%	83.53%	90.06%	0.38%	0.0646	1	0.0	None
61.95%	82.55%	87.01%	7.64%	0.0761	1	0.3	bn
68.54%	84.13%	**90.44%**	3.06%	**0.0415**	1	0.3	None
63.00%	82.63%	88.05%	6.56%	0.0810	1	0.6	bn
68.04%	83.23%	89.97%	1.04%	0.0593	1	0.6	None
65.46%	83.15%	88.41%	4.72%	0.0544	1	0.9	bn
66.94%	83.75%	89.89%	4.36%	0.0504	1	0.9	None
62.83%	82.67%	86.84%	6.88%	0.0601	5	0.0	bn
64.78%	83.03%	87.29%	5.32%	0.0902	5	0.0	None
62.87%	82.63%	83.14%	6.72%	0.0746	5	0.3	bn
64.21%	83.09%	87.43%	6.26%	0.0870	5	0.3	None
63.74%	82.97%	85.23%	6.54%	0.0622	5	0.6	bn
64.87%	83.33%	87.56%	6.06%	0.0951	5	0.6	None
66.53%	83.27%	86.84%	3.52%	0.0884	5	0.9	bn
66.43%	83.29%	88.50%	3.74%	0.1117	5	0.9	None
56.93%	81.59%	86.89%	10.77%	0.0596	10	0.0	bn
66.75%	83.63%	87.10%	4.28%	0.0873	10	0.0	None
63.77%	82.81%	85.40%	6.06%	0.0451	10	0.3	bn
67.15%	83.67%	86.78%	3.80%	0.0907	10	0.3	None
66.48%	83.31%	86.34%	3.72%	0.0828	10	0.6	bn
65.55%	83.13%	86.28%	4.54%	0.0924	10	0.6	None
65.73%	83.35%	86.69%	4.92%	0.0846	10	0.9	bn
71.42%	84.27%	86.95%	1.52%	0.1190	10	0.9	None
66.06%	83.71%	89.08%	5.52%	0.0439	15	0.0	bn
71.31%	84.57%	86.82%	**0.26%**	0.0663	15	0.0	None
57.77%	81.83%	81.31%	10.49%	0.0563	15	0.3	bn
69.03%	84.27%	86.83%	2.72%	0.0612	15	0.3	None
67.36%	83.07%	85.58%	1.64%	0.0686	15	0.6	bn
68.74%	83.87%	86.79%	1.92%	0.0972	15	0.6	None
70.74%	**84.75%**	86.89%	1.40%	0.0842	15	0.9	bn
73.50%	84.59%	87.66%	4.64%	0.1351	15	0.9	None

Table E.3: Graph attention hyperparameter optimization on DigiPatics lung.

F1	Acc	AUC	%ERR	ECE	#Layers	Dropout	Norm type
56.81%	81.59%	84.26%	10.89%	0.0356	1	0.0	bn
61.03%	82.37%	90.23%	8.26%	0.0910	1	0.0	None
60.63%	82.43%	86.74%	8.89%	0.0795	1	0.3	bn
55.89%	81.49%	89.34%	11.55%	0.0982	1	0.3	None
64.28%	83.03%	88.51%	6.00%	0.1065	1	0.6	bn
65.02%	83.03%	89.35%	5.00%	0.0817	1	0.6	None
3.67%	73.74%	88.90%	26.26%	0.1446	1	0.9	bn
41.83%	78.63%	86.76%	16.77%	0.1009	1	0.9	None
59.73%	82.11%	85.68%	9.09%	0.0548	5	0.0	bn
63.85%	82.87%	87.43%	6.12%	0.1014	5	0.0	None
47.42%	80.25%	87.61%	15.95%	0.1143	5	0.3	bn
67.32%	83.97%	87.78%	4.46%	0.1376	5	0.3	None
67.84%	84.29%	**91.34%**	4.66%	0.1086	5	0.6	bn
68.88%	83.71%	89.21%	1.16%	0.1233	5	0.6	None
0.00%	73.24%	76.25%	26.76%	**0.0171**	5	0.9	bn
0.00%	73.24%	39.62%	26.76%	0.2574	5	0.9	None
53.42%	80.95%	89.55%	12.61%	0.1048	10	0.0	bn
55.01%	81.77%	86.10%	12.99%	0.1321	10	0.0	None
68.69%	84.09%	89.74%	2.70%	0.0499	10	0.3	bn
67.44%	83.63%	90.76%	3.24%	0.0791	10	0.3	None
0.00%	73.24%	89.37%	26.76%	0.0914	10	0.6	bn
0.00%	73.24%	31.83%	26.76%	0.2673	10	0.6	None
0.00%	73.24%	50.00%	26.76%	0.1027	10	0.9	bn
0.00%	73.24%	37.30%	26.76%	0.2674	10	0.9	None
71.79%	84.57%	89.85%	1.18%	0.0526	15	0.0	bn
71.71%	**84.87%**	89.95%	**0.04%**	0.0863	15	0.0	None
67.23%	84.51%	89.59%	6.24%	0.0727	15	0.3	bn
73.56%	84.69%	89.90%	4.38%	0.0750	15	0.3	None
0.00%	73.24%	75.87%	26.76%	0.0691	15	0.6	bn
11.18%	60.58%	39.72%	9.13%	0.2755	15	0.6	None
0.00%	73.24%	54.20%	26.76%	0.0490	15	0.9	bn
0.00%	73.24%	22.17%	26.76%	0.2676	15	0.9	None

Table E.4: Graph convolution hyperparameter optimization on DigiPatics breast.

Micro F1	Macro F1	Weighted F1	ECE	#Layers	Dropout	Norm type
70.56%	41.00%	70.63%	0.2542	1	0.0	bn
68.57%	41.16%	68.07%	0.2712	1	0.0	None
68.84%	40.92%	68.72%	0.2674	1	0.3	bn
70.62%	42.32%	70.52%	0.2565	1	0.3	None
70.47%	42.53%	71.13%	0.2501	1	0.6	bn
70.61%	**42.81%**	**71.87%**	0.2694	1	0.6	None
69.35%	41.50%	69.37%	0.2587	1	0.9	bn
70.04%	42.26%	70.13%	0.2571	1	0.9	None
61.34%	41.90%	59.75%	0.2816	5	0.0	bn
61.94%	41.87%	60.57%	0.2891	5	0.0	None
55.93%	34.51%	54.45%	0.3007	5	0.3	bn
55.42%	37.74%	53.37%	0.3143	5	0.3	None
62.31%	42.61%	62.76%	0.2798	5	0.6	bn
53.93%	34.59%	52.27%	0.3183	5	0.6	None
69.40%	34.25%	67.53%	**0.2423**	5	0.9	bn
61.47%	42.49%	61.90%	0.2881	5	0.9	None
51.30%	29.90%	46.98%	0.3230	10	0.0	bn
54.86%	34.87%	53.42%	0.3051	10	0.0	None
51.57%	35.26%	50.82%	0.3037	10	0.3	bn
55.64%	34.09%	54.02%	0.3004	10	0.3	None
56.13%	37.90%	56.80%	0.2877	10	0.6	bn
53.83%	32.87%	53.88%	0.3157	10	0.6	None
65.87%	35.59%	62.88%	0.2681	10	0.9	bn
65.58%	35.72%	62.66%	0.2938	10	0.9	None
54.70%	36.89%	52.84%	0.2831	15	0.0	bn
49.67%	31.46%	47.47%	0.3174	15	0.0	None
57.22%	36.64%	55.77%	0.2797	15	0.3	bn
54.38%	33.52%	54.51%	0.3062	15	0.3	None
49.15%	29.97%	46.72%	0.3226	15	0.6	bn
51.00%	31.14%	50.70%	0.3064	15	0.6	None
64.98%	34.08%	61.82%	0.2581	15	0.9	bn
64.20%	33.50%	60.95%	0.3143	15	0.9	None

Table E.5: Graph attention hyperparameter optimization on DigiPatics breast.

Micro F1	Macro F1	Weighted F1	ECE	#Layers	Dropout	Norm type
64.57%	38.40%	65.43%	0.2772	1	0.0	bn
61.67%	36.83%	60.40%	0.2867	1	0.0	None
71.38%	40.58%	70.53%	0.2536	1	0.3	bn
64.72%	38.37%	64.66%	0.2740	1	0.3	None
69.49%	**41.55%**	70.46%	0.2751	1	0.6	bn
70.59%	41.34%	70.82%	0.2574	1	0.6	None
44.75%	24.36%	31.87%	0.3307	1	0.9	bn
72.16%	39.09%	**71.57%**	0.2428	1	0.9	None
52.85%	38.19%	49.38%	0.3166	5	0.0	bn
55.56%	34.86%	52.16%	0.3158	5	0.0	None
64.40%	37.01%	64.90%	0.2472	5	0.3	bn
60.73%	38.45%	60.38%	0.2901	5	0.3	None
66.05%	32.22%	63.85%	0.2583	5	0.6	bn
59.78%	38.04%	55.77%	0.3445	5	0.6	None
42.08%	14.81%	24.92%	0.2713	5	0.9	bn
41.92%	14.77%	24.86%	0.3512	5	0.9	None
40.25%	18.55%	32.70%	0.3634	10	0.0	bn
46.88%	30.67%	44.74%	0.3398	10	0.0	None
50.99%	27.44%	52.42%	0.3007	10	0.3	bn
24.88%	22.21%	33.39%	0.4510	10	0.3	None
42.08%	14.81%	24.92%	0.2850	10	0.6	bn
48.18%	18.78%	35.70%	0.3670	10	0.6	None
42.08%	14.81%	24.92%	0.2588	10	0.9	bn
6.60%	3.09%	0.82%	0.5596	10	0.9	None
43.95%	28.14%	46.56%	0.3415	15	0.0	bn
45.56%	26.04%	41.67%	0.3567	15	0.0	None
58.08%	27.71%	55.01%	0.3047	15	0.3	bn
2.61%	1.96%	4.77%	**0.2060**	15	0.3	None
42.08%	14.81%	24.92%	0.2838	15	0.6	bn
25.31%	11.16%	23.68%	0.2399	15	0.6	None
42.08%	14.81%	24.92%	0.2670	15	0.9	bn
47.19%	19.39%	43.47%	0.2700	15	0.9	None

Table E.6: Graph convolution hyperparameter optimization on CoNSeP.

Micro F1	Macro F1	Weighted F1	ECE	#Layers	Dropout	Norm type
64.44%	47.87%	61.42%	0.0539	1	0.0	bn
63.67%	45.25%	60.02%	0.0419	1	0.0	None
64.17%	47.01%	61.46%	0.0541	1	0.3	bn
63.75%	46.39%	60.78%	0.0394	1	0.3	None
61.74%	46.78%	59.18%	0.0637	1	0.6	bn
64.67%	**48.24%**	**61.82%**	0.0415	1	0.6	None
57.46%	26.66%	51.03%	0.0681	1	0.9	bn
58.98%	26.46%	52.75%	**0.0376**	1	0.9	None
62.66%	42.05%	59.30%	0.0544	5	0.0	bn
58.77%	39.85%	56.42%	0.0589	5	0.0	None
58.08%	32.41%	51.75%	0.0759	5	0.3	bn
64.01%	46.02%	60.56%	0.0654	5	0.3	None
46.21%	18.21%	34.93%	0.0879	5	0.6	bn
49.08%	32.92%	46.46%	0.1057	5	0.6	None
31.58%	6.86%	15.15%	0.0960	5	0.9	bn
35.13%	15.62%	28.62%	0.0683	5	0.9	None
45.16%	35.88%	43.11%	0.1235	10	0.0	bn
57.76%	38.56%	55.33%	0.0884	10	0.0	None
61.63%	43.54%	58.23%	0.0562	10	0.3	bn
61.84%	42.40%	57.42%	0.0704	10	0.3	None
21.27%	5.01%	7.46%	0.0758	10	0.6	bn
15.65%	5.56%	11.84%	0.1898	10	0.6	None
21.27%	5.01%	7.46%	0.0812	10	0.9	bn
25.86%	9.86%	17.46%	0.1449	10	0.9	None
29.28%	21.74%	29.48%	0.1663	15	0.0	bn
51.08%	38.71%	47.74%	0.1195	15	0.0	None
55.27%	32.16%	48.99%	0.1047	15	0.3	bn
21.66%	12.53%	16.38%	0.1571	15	0.3	None
21.27%	5.01%	7.46%	0.0660	15	0.6	bn
29.18%	6.50%	13.40%	0.1931	15	0.6	None
31.58%	6.86%	15.15%	0.0883	15	0.9	bn
11.90%	5.04%	8.82%	0.2094	15	0.9	None

Table E.7: Graph convolution hyperparameter optimization on MoNuSAC.

Micro F1	Macro F1	Weighted F1	ECE	#Layers	Dropout	Norm type
83.60%	72.26%	83.67%	0.0549	1	0.0	bn
85.48%	75.53%	85.49%	0.0436	1	0.0	None
84.54%	74.18%	84.62%	0.0509	1	0.3	bn
86.67%	74.19%	86.75%	0.0386	1	0.3	None
84.46%	75.41%	84.42%	0.0577	1	0.6	bn
87.21%	75.98%	87.28%	0.0394	1	0.6	None
83.89%	74.60%	83.92%	0.0503	1	0.9	bn
87.32%	76.17%	87.42%	0.0429	1	0.9	None
86.38%	**76.62%**	86.36%	0.0344	5	0.0	bn
84.63%	73.59%	84.66%	0.0436	5	0.0	None
83.59%	71.47%	83.59%	0.0604	5	0.3	bn
84.49%	73.06%	84.56%	0.0436	5	0.3	None
83.53%	71.09%	83.46%	0.0533	5	0.6	bn
84.41%	73.95%	84.44%	0.0446	5	0.6	None
83.92%	45.29%	82.79%	0.0663	5	0.9	bn
84.88%	69.20%	84.76%	0.0511	5	0.9	None
87.11%	72.21%	87.32%	0.0418	10	0.0	bn
87.07%	73.01%	87.26%	**0.0244**	10	0.0	None
85.16%	74.56%	85.23%	0.0537	10	0.3	bn
85.44%	75.31%	85.44%	0.0427	10	0.3	None
84.66%	70.08%	84.67%	0.0484	10	0.6	bn
85.95%	73.55%	86.04%	0.0422	10	0.6	None
84.00%	42.56%	82.75%	0.0719	10	0.9	bn
87.90%	44.56%	86.67%	0.0440	10	0.9	None
82.32%	68.89%	82.44%	0.0689	15	0.0	bn
87.11%	72.75%	87.32%	0.0364	15	0.0	None
79.96%	63.06%	80.48%	0.0907	15	0.3	bn
88.71%	69.72%	**89.05%**	0.0251	15	0.3	None
85.39%	70.00%	85.40%	0.0283	15	0.6	bn
88.37%	75.00%	88.41%	0.0400	15	0.6	None
84.25%	42.69%	82.99%	0.0689	15	0.9	bn
87.16%	44.19%	85.97%	0.1254	15	0.9	None