Breast Cancer Classification with Capsules and Convolutional Neural Nets

Marcell Veiner

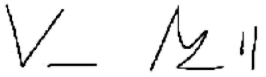
A dissertation submitted in partial fulfilment
of the requirements for the degree of
Bachelor of Science
of the
University of Aberdeen.

Department of Computing Science

2021

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed: 

Date: 2021

Abstract

Breast cancer is one of the most common type of cancer in the UK, and one of the leading causes of cancer related deaths worldwide. Diagnosis is often set up with the help of computer aided techniques, however, current approaches (convolutional neural nets), discard valuable low-level information which can be essential in distinguishing between types of tissue. In recent years a new approach has been proposed, called capsule networks, and their importance in medical imaging has soon become apparent.

In this study we have set out to implement and test several versions of capsule networks, and compare their performance with convolutional neural networks. We have used a two staged approach for this, where the first phase learns an appropriate representation of the patched histology images through classification, and the second phase classifies patches using the obtained models to downsize patches. The label of the whole image is then reconstructed through the votes of the individual patch labels.

Our approach achieved competing performance on the binary BreakHis dataset, and got close to the competition on BACH. We have also established baselines on a novel 3 class dataset (Databiox), for which there are no published results as of this day. Our results showed that capsules can outperform convolutional nets in this setting, and can compete with transfer learning approaches.

Acknowledgements

As for any project, this one as well turned out to be harder than it seemed at the first glance, and finishing it would not have been possible without the help and support of some people. First and foremost, I would like to thank Dr Georgios Leontidis for his guidance and supervision. He helped me deepen my knowledge about deep learning in general, and motivated me to do further work in this field. I really hope that at some point we will have a chance to work on something together again.

I am also grateful to Eszter Csorba for her patience while listening to the endless stream of my complaints, when I could not get things to work. She, as well as many of my peers in the computing science department, are the reasons I managed to finish this project during these difficult times.

Lastly, I would like to thank Professor Nir Oren and Dr Mingjun Zhong for their flexibility in scheduling the demo, and for their understanding towards an extension. I am sure that it did not help their already busy schedules, and I have tried to keep this in mind when putting down the final touches. Thank you all.

Terminology

- Convolutional Neural Networks: CNNs or ConvNets are a type of deep neural nets, used on data that is known to have a grid like structure, most typically images. The fundamental building blocks of CNNs are pooling layers, convolutional layers and fully connected layers.
- Capsule Networks: Capsules or CapsNets is a new type of neural networks, designed to more closely resemble the biological neural organization in image recognition. The novelty comes from a new structures, called capsules, which can learn the pose of an object, as well as its presence. This help capsules achieve equivariance to affine transformations.
- Affine Transformation: By affine transformation we will mean a non-linear viewpoint change. Affine transformations respect parallel lines, but not distances.
- Invariance: Invariance (to a transformation) means that if you take the input and transform it, then the representation you get the same as the representation of the original. Informally, we will mean that a model is invariant, if the internal representation of an object is the same regardless of the transformation in question. Pooling layers are translation invariant.
- Equivariance: Equivariance (to a transformation) means that if you take the input and transform it, then the representation you get is a transformation of the representation of the original. Informally, we will mean that as an object is transformed, the internal representation of a model changes with it. Convolutional layers provide translation equivariance.
- Accuracy: Accuracy is the most typical metric used in classification settings and is equal to the number of samples the model predicted correctly divided by the total number of samples.
- Sensitivity[1]: Sensitivity, also known as True Positive Rate (TPR) or Recall, measures the proportion of positives that are correctly identified and is equal to: $\frac{TP}{TP+FN}$.
- Specificity[1]: Specificity or True Negative Rate (TNR) measures the proportion of negatives that are correctly identified and is equal to: $\frac{TN}{TP+FN}$.
- F1 Score[1]: The F-score, or F1-score, is a measure of a model's accuracy taking into account class imbalance and is defined as: $F1 = \frac{2TP}{2TP+FP+FN}$.

[1]For multi-class classification we define all of these metrics per class, and treat all the other classes as negative samples.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Breast cancer is one of the most common type of cancer in the UK, and about 1 out of every 8 females are diagnosed with it at some point in their lifetimes [46]. A total of $11,547$ cases of mortal breast cancer have been estimated between 2016-2018 in the UK alone, accounting for 7% of total cancer deaths on average, with 99% of mortal cases being females [66]. This makes breast cancer the 4th most common cause of cancer death in the UK [66] and the first in women aged 20 to 59 year in the US [60].

As treatment is essential in the early stages to prevent the progression of the disease [62], establishing the correct diagnosis as early as possible is critical. Although noninvasive methods are common as a first step, they are usually not sufficient to tell whether a lump or growth inside the body is cancerous (malignant) or non-cancerous (benign). In these cases the surgical removal of sample tissue (biopsy) is required, and the microscopy imaging data obtained this way is then used to establish the correct diagnosis. These images are both large in size and complex; consequently they require significant amount of time and effort from pathologists to be analysed. In order to automate the process, computer aided techniques for diagnosis have been gaining momentum over the past couple of years [20].

Deep Learning (DL) in particular has become quite popular in the field of bioinformatics due to its capability to extract knowledge from raw input [43]. For example, DL approaches can be used to classify images based on their content, which makes them very favourable to many purposes, including medical imaging. However, the most popular approaches in image classification (Convolutional Neural Networks or CNNs), discard valuable low-level information [57], which can be essential in distinguishing between types of tissue. In recent years an alternative approach, called Capsule Networks (Capsules), has been proposed to correct for this information loss [57].

Capsules have many advantages over their counterparts, which makes them a promising candidate for our purposes. Motivated by this, our goal for this study is to design and evaluate capsule networks that can be used to detect cases/types of breast cancer. We will begin, however, by stating the goals of this project explicitly.

1.2 Objectives

This project aims at implementing and testing several capsule networks based on the work of [28; 36; 55; 56; 57; 58] and training them on histopathological images of breast cancer. We will use three publicly available datasets for testing, one of which is the *histopathological image*

dataset for grading breast invasive ductal carcinomas [9], which we will refer to as Databiox as in the paper. This is a labeled dataset, indicating not the presence or absence of breast cancer, but its grade, which can be used as a prognostic factor. For a proper discussion on this dataset see Section 3.1. As this is a recent dataset, as far as we can tell, there are no DL baselines established on it, including convolutional networks. Therefore our first and second objectives are the following:

1. Establish a baseline for convolutional neural networks on Databiox.

2. Design a capsule network that outperforms the baseline on Databiox.

Although we have found no baseline for Databiox, there are other publicly available datasets for histopathological images of breast cancer, for example the *Breast Cancer Histology* (BACH) dataset [7] is a 4 class dataset, while *The Breast Cancer Histopathological Image Classification* (BreakHis) dataset [63] is binary. There are many models developed on these two, including but not limited to the two-stage convolutional network proposed in [45], which has been trained and tested on BACH. It would be tempting to claim as our next goal to beat the state-of-the-art, but in such a short amount of time the following seem more appropriate:

3. Achieve competing performance with capsule networks on BACH.

4. Achieve competing performance with capsule networks on BreakHis.

Moreover, our hope is to collaborate with the medical school and evaluate our approach on the binary dataset they have available. Hopefully this will start a discussion with the appropriate members of staff, as getting their insights on the performance of our model would be really illuminating. As the current lockdown has made this more challenging to organise, this will only be stated as an optional objective.

5. Evaluate on the binary dataset from the medical school (Optional).

In the project plan, we have also discussed developing a simple webapp to showcase the project. After careful consideration we have decided to leave this objective out and instead focus all of our efforts on evaluation. Overall, this decision has allowed for more tests, and further parameter tuning. This makes our list of objectives, now let us see how the time was spent on the project.

1.3 Project Overview

This project aimed at testing some variants of capsule networks in a medical imaging setting. The early weeks of the project were spent performing background reading, such as the core papers [28; 36; 55; 56; 57; 58], but also about deep learning in general [22], especially during the winter break. Then the focus turned towards a proof of concept, that is developing a CNN on one of the listed datasets. At the early stages, the focus was on the Databiox dataset, and the scope has been enlarged to involve the other two as well. One immediate benefit of this decision has been the availability of other baselines, as we have found none for Databiox. The binary dataset (BreakHis) has also proved to be benefitial for testing our approach, as it was simpler than the other two.

At this point the project was mostly written in Keras, which posed difficulties in integrating the first version (dynamic) capsules. We have also experimented with different image processing

steps, before settling with the current two-staged patched version. As a result, the project has been rewritten in PyTorch and structured as it is now, which sped up development significantly. There has been a final round of refactoring, which resulted in the current version of the code.

The final weeks of the project were spent running experiments on Maxwell (occasionally on Colab and Kaggle as well) and writing the report. As the final pipeline (Maxwell) has been setup rather late, the IT disruptions towards the end would have meant significant setback without an extension. During these weeks it was especially challenging to keep a proper balance between the project, and the rest of the courses, which all had scheduled deadlines in these final weeks.

Overall, this project achieved many things, for example we experiment with different datasets, models, architectures, and achieve competing performance in some cases. It also served as an entry point to deep learning and the field of biomedical AI. Nevertheless, both capsules and medical imaging remain vast topics, and the work presented here is by no means a complete overview, but one only for 30 credits. That said, this project has been designed so that it could serve as an excellent starting point for future work.

1.4 Project Structure

As a summary of this report, please see the following description of sections.

1. Introduction: States the motivation and goals of this project, including a short retrospective.
2. Background and Related Work: This section aims at describing the main theoretical background for the project, such as capsules and convolutional networks, as well as their use in medical imaging studies.
3. Methodology: This section describes the experimental setup, such as datasest models used, architectural and hardware considerations, and related design choices.
4. Implementation: Considers implementation details of the networks presented, and the sources for the variants of capsule networks. We also discuss the code structure and preprocessing steps.
5. Results and Evaluation: Discusses the performance of the models trained, for each dataset separately.
6. Conclusion and Future Work: Discusses the results and proposed how the project could be used for future work, and possible directions.
7. Appendices: Contains the user manual and maintenance manual.

Chapter 2

Background and Related Work

This section aims at describing the main theoretical background for the project. We will start with a quick introduction to deep learning and image processing, followed by the reasons convolutional neural networks were considered to be such a success. We will then discuss their drawbacks and introduce capsule networks with their many flavours. To finish, we will discuss the relevant work, that is models (CNNs and Capsules) trained on similar datasets.

2.1 Literature Review

2.1.1 Deep Learning

The scope of this paper is by no means enough to present the full hierarchy of the subfields of Artificial Intelligence (AI), but it is good practice to position ourselves in the bigger picture. The field of AI arose as a natural response to the first general programmable computers and the materialist thinking of consciousness. Although the word "intelligent" lead many into less fruitful directions, AI is a thriving field with many practical applications and active research topics.

One of the more practical directions of AI is the field of Machine Learning (ML) which, instead of the traditional knowledge-based approaches, was built on the principle that AI systems need the ability to acquire their own knowledge by extracting patterns from raw data. The goal was for computers to learn solving complicated tasks, where listing the rules explicitly by hand would not be feasible. In addition, researchers also hoped that through ML computers will be able to abstract knowledge and learn a simpler representation of the dataset, as in many cases it can simplify the problem at hand greatly (see Chapter 1 of [22] for examples).

When it is nearly as difficult to obtain a representation of the data, as to solve the original problem, one needs to resort to other approaches. To overcome this, Deep Learning (DL) uses a hierarchy of abstractions, where each representation of the data is expressed through previous, simpler representations. The quintessential example for such a chain of abstractions can be found in image classification, where lower representations learn to identify simple object such as corners and edges, which are then combined by higher level representations to identify the whole image (e.g. see Figure 1.2 in Chapter 1 of [22]).

Even though the concepts of DL have appeared as early as in 1943 [42], the current resurgence under the name deep learning begin around 2006 [27] and has been rising in popularity ever since [33]. Advances in the available processing power made DL approaches reach superhuman performance in many areas, such as speech recognition, time series analysis, and image processing [22]. In fact the field of computer vision has seen probably the most drastic shift from statistical

methods to DL (such as CNNs) that relied on big data and raw computational power, and have pushed the boundaries of what was possible [48].

2.1.2 Convolutional Neural Networks

Convolutional neural networks (Chapter 9 of [22]), are a kind of neural network for processing data that has a known grid-like structure, for example time series data, images, or video. The concept of CNNs first appeared in the 80s under the name *Neocognitron* [19]. A few years later in 1989 the first multilayered CNN named ConvNet was introduced [40], which laid the foundation for the modern 2D CNNs. In following years many improvements have made CNNs more accessible, for example the use of Graphical Processing Units (GPUs) to speed up training [47], achieving gradient descent through backpgropagation [14] and applying backpropagation to convolutional filters [15].

All of the above (and more) led to CNNs becoming the standard technique in image processing, and reaching new highs on many benchmark datasets [33]. But the accessibility of CNNs was at least partially due to the idea that the exact location of a feature is less important than its approximated location relative to other features, which in turn decreased the spatial complexity of the representation and thus the number of parameters in the model. This was achieved through *Parameter Sharing* and *Pooling*.

Pooling is the third key component of a Convolutional Layer, and works by replacing the output of the net at a certain location with a summary statistic of the nearby outputs [22]. A common example is Max-pooling, where a whole (rectangular) region is replaced by its maximal value. Although networks with pooling layers can work quite efficiently, they loose valuable local information, which in many situations could be essential. This is also made worse by using shared parameters, which makes CNNs invariant under translation. The advantage of this is that CNNs can recognise a feature (for example an eye) using the same parameters regardless whether it is in the top left corner of the image or the bottom right. As a downside, however, the internal representation of a CNN cannot take into account the important spatial hierarchies between simple and composite objects. Consider the example on Figure 2.1, a CNN (in theory) will detect a face even if the spatial relationships of the elements of a face are impossible.

Similarly, CNNs are incapable of representing the pose of a feature, i.e. orientation, size, colour. As a result a CNN trained solely on pictures like the left side of Figure 2.1 could fail to detect a face if it was upside down. This can be avoided by using data augmentation, but every variant of the original feature would have to be included, i.e. resizing, random rotations, colour perturbations etc. Some changes, however, for example affine transformations represent nonlinear changes in pixels, and data engineering only sidesteps the issue instead of solving it [55].

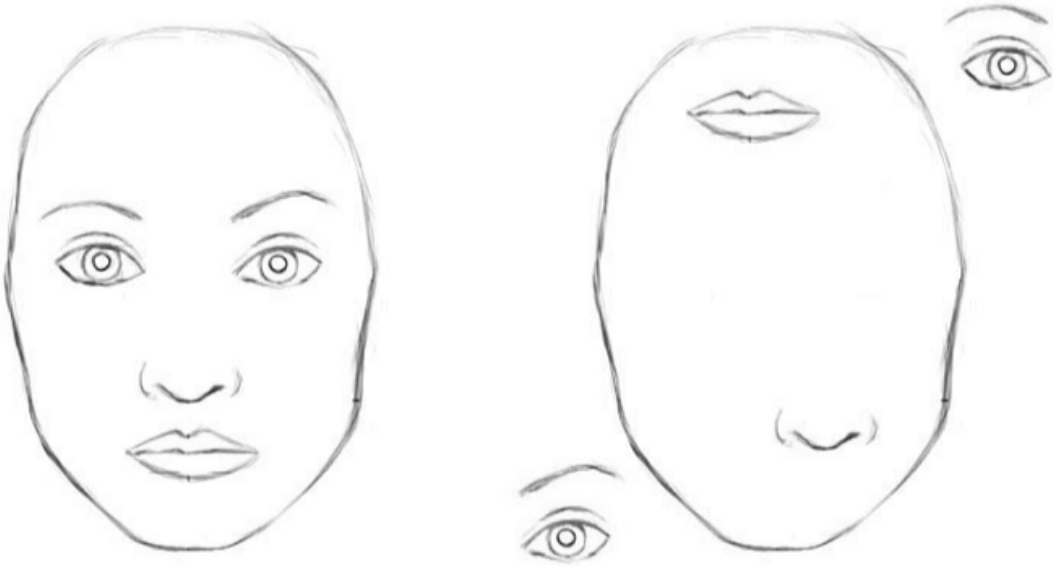


Figure 2.1: CNNs Recognising a Face [51].

Building on these drawbacks, researchers have found that it was far too easy to fool CNNs with an imperceivable, but carefully constructed nudge in the input [23]. Figure 1 (in [23]) is most astonishing in this study, as by adding imperceptible noise to an image of a panda, they managed to fool GoogLeNet [64] into believing that it is seeing a gibbon, whereas the image is really showing a panda. This has lead the field into a battle between studies making CNNs more robust to adversarial attacks, and others proposing new ways of fooling the models. We could debate more about the drawbacks of CNNs, however, these properties will be sufficient to compare them with capsule networks. Below are the drawbacks of CNNs touched upon in this section.

1. Vulnerability to adversarial attacks.
2. Inability to handle the nonlinearity of viewpoint changes without data engineering.
3. Transformation invariance.
4. Max pooling looses local information.

With that, we will now turn our attention towards capsule networks.

2.1.3 Capsule Networks

Capsule Networks (CapsNets or Capsules) are a new deep learning approach proposed to address the fundamental flaws of CNNs. Its first version appeared in [57], which was then refined several times by the same group [28; 36; 58], i.e. Geoffrey Hinton and his research team at Google. Capsules have caused a lot of excitement within the DL community, as they did not follow the traditional layered structure of neural networks, but approached the problem in an entirely different way.

The novelty comes from the introduction of new structures into the neural net, called *capsules*, which can be defined as a group of neurons that individually activate for various attributes of

an object, such as its location, orientation, size, etc. Additionally, capsules store information about the probability of an entity's presence. Thus capsules hold two pieces of information, *activation* and *pose*, where by pose we mean all features related to an entity, not just its spatial attributes. For an intuitive example see Figure 2.2, where the presence of an entity is encoded by the length of the appropriately coloured vector, and the vectors location and orientation measure the entity's pose. Finding the magnitude and orientation of these feature vectors is what Hinton calls *inverse graphics*, and even compares it to how the human vision works [26].

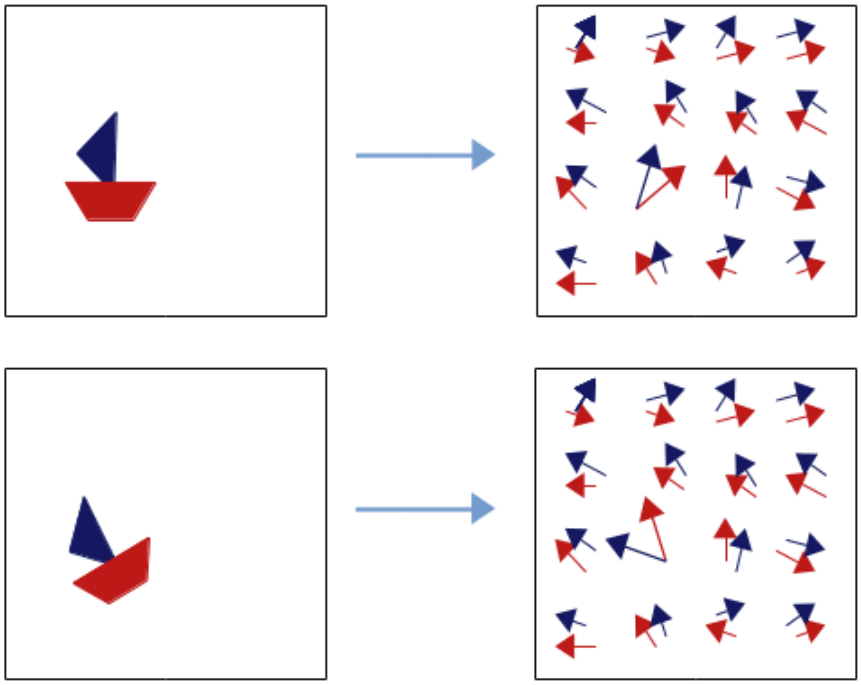


Figure 2.2: Capsules Finding Parts of Objects [11].

Knowing the pose of the parts of an object, the next step is to recognise the whole object. In fact, encoding the activation of a part capsule by the magnitude of the feature vectors (as in Figure 2.2) is how it was first proposed in [57]. In this version, inferring the whole objects is done by *routing-by-aggreement* (but we will refer to this approach as *Dynamic Routing*), where lower level capsules will send their input to the higher level capsule that "agrees" with their input. The agreement between capsules is measured as their dot product, which Hinton criticised in later versions, but it serves as a good introductory example.

Capsules since then have been refined many times by introducing better representations for the pose of an entity [28], improving the routing mechanism [25; 28; 55] and finally introducing unsupervised capsules [36; 58]. Nevertheless the main ideas remained the same, and even this simple version allows us to see how they present an alternative to CNNs.

1. Capsules are more robust to adversarial attacks [28].

2. Viewpoint changes have linear effects on part-whole relationships [56].

3. Capsules are equivariant to translation and affine transformations [57].

4. Unlike CNNs, capsules do not discard information about the position of an entity [57].

Although this list is not complete, the results are clear: Capsules are, in theory and practice, superior to their counterparts [57]. They also have some domain specific benefits as we will soon see, and also their own hindrances in return. Nevertheless, this introduction to capsules will be sufficient for us to be able to discuss their relevance in medical imaging.

2.2 Related Work

2.2.1 Convolutional Neural Networks in Medical Imaging

In recent years, several studies aimed at breast cancer detection and classification using CNNs have been published, all with their unique problem statements [45]. In its simplest form, the problem boils down to binary classification (malignant vs. benign). While some studies propose solutions to the 2-class problem [17; 37], others simply include it in the experimental setup along with a more complicated classification task [3; 8], or focus on classifying multiple types of cancer [6; 45].

As we can see a great deal has already been done with CNNs, and there are some key takeaways. Firstly, histopathological image classification is a challenging problem due to the complexity of the images [54], and because the differences between the classes are really subtle (see Figure 2.3; the left image has histologic grade 1, while the right one has grade 3). As a consequence, studies either used deeper CNNs [8; 54], a mix of different models [37; 45] or transfer learning [10].

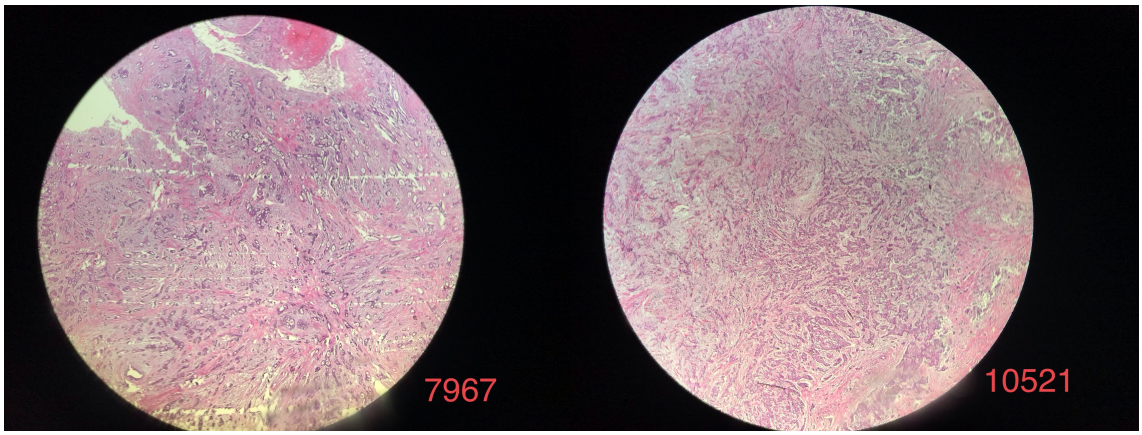


Figure 2.3: Examples of the Databiox dataset [9].

Secondly, histopathological images are costly to produce, and there are only a limited number of them available in public datasets, which is why most approaches resorted to heavy data augmentation. For example [45] produces 8 different versions of the same dataset while training using rotations and mirroring, while in [3] they produce 20 augmented samples from a single patch. Moreover, as the resolution of these type of imaged are high (2048 x 1536 pixels for BACH) feeding them to the classifier directly is unfeasible. Down-scaling does not help either, due to the subtleties between classes and image complexity. Thus the best possible option is to create patches of the images, creating an even larger dataset [6; 45].

We will take these hard-earned lessons and incorporate them into our approach. In particular, we will briefly revisit the two-staged experiment seen in [45], which in turn was inspired by [6],

as we will base our approach on the same principles. But for now let us discuss capsules in the related work.

2.2.2 Capsules in Medical Imaging

The importance of Capsule Networks for medical imaging have quickly been noticed, as their equivariance properties reduce the data requirements of CNNs, and are therefore promising for medical image analysis [30]. This study has evaluated capsules on 2 vision datasets (MNIST, Fashion-MNIST), and two medical imaging datasets (TUPAC16, DIARETDB1). In summary (dynamic) capsules outperformed other architectures, and showed increased ability to generalise with limited amounts of training data.

In another study 3D (dynamic) capsules have been used for Alzheimer's disease diagnosis based on MRI scan images [38]. The results show the capsules have outperformed 3D CNNs by 1.51%, achieving 99.76 % F1 score. This 3-dimensional settings offers a great opportunity for capsules to shine, as the feature vectors can represent object pose in space effortlessly.

There have been a few studies applying capsule networks to breast cancer classification focusing on the BACH dataset. For example in [29] 87% accuracy averaged over the 4 classes using an architecture of 5 convolutional layers and capsule layer (Primary + Dense) with dynamic routing between capsules. They have achieved this accuracy on the validation set, however, and no results on a separate test set have been reported. A similar study presented in [5] has improved on these results, achieving 92.14% accuracy with a single convolutional, capsule and dense layer, training only on 256 of the 400 images of BACH.

Other studies applying capsules to similar datasets have also contributed with a number of unique ideas [30; 72], but in general, most approaches have decided to use dynamic routing, over newer types of capsule networks. One possible reason for this is the fact that no official implementation has been released for EM Capsules and people struggled to implement it [24], and that newer capsule variations emerged only recently.

Nevertheless, dynamic routing has even been criticised by Geoffrey Hinton himself [1], which leaves us with a unique opportunity of applying newer capsules versions to breast cancer classification, building on the work presented in this section. Finishing our deep dive into background and related work, we are now ready to present our experimental setup.

[1] Source: (Youtube)

Chapter 3

Methodology

This section describes the experimental setup, such as datasest and models used, architectural and hardware considerations, and related design choices.

3.1 Datasets

There were 3 datasets considered during the project. Although these datasets are mostly similar in nature, they all have their advantages and disadvantages. These are listed and discussed below. In all 3 cases we have used a training, test and validation split of 70%, 15% and 15% respectively, but the number of samples in each of these categories depended on the patching strategy, and data augmentation, which are discussed in Sections 3.4 and 4.1.

3.1.1 Databiox

The histologic grade is a prognostic factor, an indicator of the patient's response to chemotherapy, and it has been shown that it is closely related to mortality rates. Thus the correct identification of the histologic grade is essential when considering the right treatment plan for patients [9]. The scoring is done traditionally by visual examination through the microscope, which is another time consuming process for pathologists. Considering the popularity of cancer detection and cancer subtype classification, the *histopathological image dataset for grading breast invasive ductal carcinomas* (Databiox) has been published [9].

The dataset consists of 922 samples taken from 124 patients, and labeled as either "Grade I", "Grade II" or "Grade III". Although there were more patients of Grade I and II, the number of samples taken from each patient varied as well (see Table 3.1). In particular, the images were produced at 4 different magnification levels. As we can see there is some class imbalance, 39% of the samples are of Grade II, but this difference does not require special considerations other than reporting proper evaluation metrics.

	Patients	4x	10x	20x	40x	Total Samples
Grade I	37	45	40	43	131	259
Grade II	43	59	64	63	180	366
Grade III	44	56	49	49	143	297
Total	124	160	153	155	454	**922**

Table 3.1: Databiox Sample Sizes [9].

The images are all JPEG files with RGB channels. The study states that the resolutions of

these images are either 2100 × 1574 or 1276 × 956 [9], however, we have found these numbers to differ on the downloaded dataset, with most of them being 3024 x 4032 (see Figure 3.1). This, along with the fact that the images contain a blank region around the microscopic sample (see Figure 2.3) needed extra considerations when preprocessing the dataset.

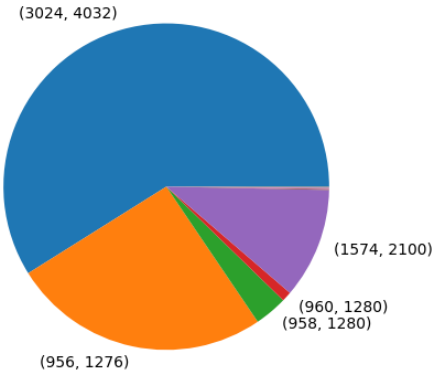


Figure 3.1: Databiox Actual Resolution Sizes.

For the experiment we resized each image to 2816 x 3072, and center-cropped with size 1536 x 2048. Doing so we managed to get rid of some of the blank area around samples (see Figure 2.3), which contained no valuable information. From these images we obtained patches of size 512 x 512, as we describe in 3.4. The mean and standard deviation of each colour channel have also been computed for normalisation purposes. We have not distinguished between magnifying factors, however, and considered it an opportunity for the networks to learn size equivariance.

Although there are a number of similar datasets, labeled with tissue and carcinoma subtypes, this one remains the only one labeled with the histologic grade [9]. It is also important to note that the graded samples are invasive ductal carcinoma, which makes it possible to use Databiox to augment another dataset with subtypes.

3.1.2 BACH

A more widespread dataset is the *Breast Cancer Histology Images* (BACH) featured in the ICIAR 2018 challenge [12]. Images of BACH are labeled as one of “normal”, “benign”, “in situ carcinoma” or “invasive carcinoma” according to the predominant cancer type in each image. The images were labeled by two pathologists without specifying the area of interest.

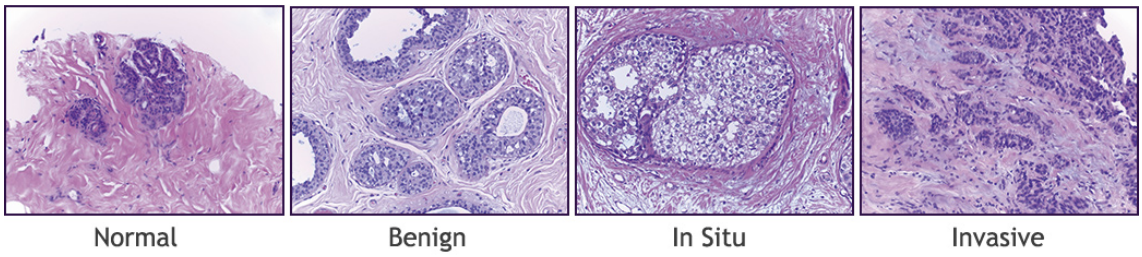


Figure 3.2: Samples of the Breast Cancer Histology (BACH) dataset [7].

The dataset consists of 400 RGB images, 100 in each subtype. The file size is tiff and the resolution is 2048 x 1536 pixels. For examples see Figure 3.2. As this is far too large for any network to handle as a whole, creating smaller patches of this dataset is required. Visual inspection shows that cell nuclei ranges from 3 to 11 pixels in radius, thus a minimal patch size of 128 x 128 is recommended [3]. In our experiments BACH images for resized to 1536 x 2048, and then patched to images of size 512 x 512 as described in 3.4.

The uniformity of the dataset, same magnification level, the fact that it is perfectly balanced, and that it was featured in a public competition made this dataset popular for different ML approaches, as we reviewed it in Section 2.2.1. As an added bonus, an unlabeled test dataset has been released with 100 samples, but we have not managed to get a hold of this.

3.1.3 BreakHis

The Breast Cancer Histopathological Image Classification (BreakHis) dataset [63] is composed of 7,909 histology images of breast tumor tissue collected from 82 patients (version 1.0). The images have magnifying factors (40x, 100x, 200x, and 400x) and are labeled as “benign” or “malignant”, with 2,480 benign and 5,429 malignant samples. Additionally it contains the subtype of the tumor i.e. adenosis (A), fibroadenoma (F), phyllodes tumor (PT), and tubular adenona (TA) for benign; carcinoma (DC), lobular carcinoma (LC), mucinous carcinoma (MC) and papillary carcinoma (PC) for malignant. For a complete breakdown of the number of samples in each subtype see Table 1 of [3].

The images are 700 x 460 pixels with RGB channels, saved in PNG format (see Figure 3.3). As these sizes are significantly smaller than for BACH and Dataiox, we have decided to resize the images to 512 x 512 and treat them as the patches. Therefore the BreakHis dataset has not been patched.

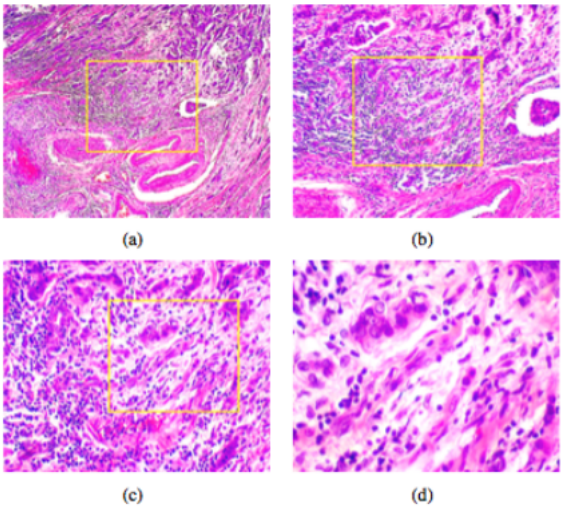


Figure 3.3: Samples of the Breast Cancer Histopathological Image Classification (BreakHis) dataset under magnification levels ((a) 40x, (b) 100x, (c) 200x, and (d) 400x) [63].

As we can see the class imbalance of BreakHis is significant, and it needs to be considered when designing the experiment, and choosing evaluation metrics. Thus we measured the networks performance not only in accuracy, but assessed its specificity and sensitivity as well as its F1 score. Concluding our discussion on the relevant datasets, we will now turn our attention towards the hardware considerations of our approach.

3.2 Computational Resources

Deep learning is infamous for its need for computational resources, thus finding the right platform for the project was essential. To address the huge computational challenge in deep learning, many tools exploit hardware features such as multi-core central processing units (CPUs) and GPUs to shorten the training time. For example GPU is the best platform for large fully-connected models, but models with large batch sizes perform best on TPU, with both of them outperfoming CPUs [68]. In general, benchmarks show significant speed up using GPUs over CPUs, some even 10-30x speedup [59], but these figures become outdated every year as new hardware is released.

Furthermore, speed is not the only resource to consider in our case. Some platforms offer GPU processing, and thus significant speedup, but are less reliable due to their nature (e.g. Kaggle). Being able to specify a list of experiments beforehand with the reassurance that it will be run is worth more in our case even if it takes 3-5x longer than with other methods. To consider a host platform as a good choice for our project it needs to satisfy the following criteria:

1. Easy to learn and set up.
2. Reliably runs experiments without supervision.
3. Decent and constant speed.
4. Continuous integration.

Most of these points are self-explanatory, except for the last. By continuous integration, we mean a working pipeline from development to running the experiment. We will see how

each considered platform solves this and the other points listed, along with their advantages and disadvantages, but in practice each platform is good for different purposes, and were used at different stages of the project (except for Google Cloud Platform).

3.2.1 Google Colaboratory

Google Colaboratory, or "Google Colab" for short, is a free online cloud-based Jupyter notebook environment that allows to train models on CPUs, GPUs, and tensor processing units (TPUs). As all Google Colab needs is a web browser, it has become a very popular choice for prototyping DL models. Its full list of advantages can be found below.

- Free to use, and very easy to set up.
- GPU acceleration (Nvidia K80 or T4 allocated at random).
- Easily integrates with Google Drive.
- Packages for DL pre-installed.
- Notebook style development, but can run commands as well.

The approachability of Google Colab guarantees criterion 1, and as we can store our dataset on Google Drive criterion 4 is also guaranteed. The notebook style development is a bonus, as it is good for prototyping but is cumbersome afterwards, luckily Google Colab does not constrain us to developing only in notebooks, and can run scripts from the command line as well. We follow with the disadvantages.

- 12 hours execution time.
- 90 minutes idle time.
- GPU not guaranteed in heavy demand and random if available.
- Google Drive files can become corrupt if it is used in another tab.
- Can stop execution due to timeout or heavy demand.

Most of the disadvantages of Google Colab stem from its free to use nature, as it was intended for interactive use. For example the idle time means that the notebook has to be constantly open and connected. Although there are workarounds for the execution and idle times, such as using checkpoints, training models this way is not ideal. From the above it is clear that Google Colab fails to fully satisfy criterion 3, but fails completely at criterion 2. Nevertheless, Google Colab has been used to setup the demo and is recommended to be used to run the project for single use.

3.2.2 Kaggle

Kaggle recently became the home of the growing data science and ML community, hosting many interesting datasets, online courses, and competitions. More importantly it also offers an environment for developing and training models. Although it is also owned by Google, its hardware resources are different (see Table 3.2).

	Google Colab	Kaggle
GPU	Nvidia K80 / T4	Nvidia P100
GPU Memory	12GB / 16GB	16GB
GPU Memory Clock	0.82GHz / 1.59GHz	1.32GHz
Performance	4.1 TFLOPS / 8.1 TFLOPS	9.3 TFLOPS
No. CPU Cores	2	2
RAM	12GB	12GB
Disk Space	358GB	5GB

Table 3.2: Google Colab vs Kaggle GPU Specs [32].

Most hardware resources offered by Kaggle are actually competing or superior to those available on Google Colab. However, experiments show that the Nvidia K80 can significantly outperform the P100 [21]. Below is the list of advantages of opting for Kaggle.

- Free to use, and very easy to set up.
- 30+ hours of GPU time per week.
- Can save datasets.
- Kaggle community.
- DL packages pre-installed.
- Notebook style development.

30 hours is guaranteed every week, which is an improvement compared to obtaining a GPU only when it is not used by paid users, but the fixed time frame would mean that every experiment had to be designed carefully, and experimentation would have to be kept to a minimum.

- 9 hours of execution time.
- 60 minutes of idle time.
- Less disk space than with Google Colab.

We conclude that although slower, Kaggle fulfills most criteria except for criterion 2, as it times out after 9 hours, or an hour after logging out. Nevertheless, Kaggle proved to be a decent platform for early experimentation.

3.2.3 Google Cloud Platform

Google Cloud Platform (GCP) is a portfolio of cloud computing services for hosting web applications from Google's data centers. GCP includes Compute Engine instances, which can be set up with almost every operating system and hardware configuration, and can essentially be used for training computationally expensive models. In theory one can obtain a hardware with GPU, user stories have reported Google to be reluctant in providing GPUs for new users, so this is not included in the pros (left column) and cons (right column).

- 300$ Computation time as new user.
- Fully customisable hardware specs.
- Reliable, as it is a paid service.
- Multi-core CPUs available.
- Complicated to set up.
- Limited Computation time.

- Essentially no GPUs.

Which means we have traded Criterion 1 for 2. GCP would be an optimal choice for larger projects, but in our case it was not a justified investment.

3.2.4 High Performance Computing at the University of Aberdeen

The High Performance Computing (HPC) service at the University of Aberdeen provides large amounts of computational processing power for academic research, and our request to use the cluster (both Macleod and Maxwell) for the project has been granted. The hardware specifications of the cluster can bee seen in Table 3.3.

Node Type	Node Name	Cores	Useable Memory	GPU
High Memory	hmem[01-11]	40 (2x Intel Xeon Gold 6138 2.00Ghz)	380G	
Consumer GPU	cgpu[01-06]	40 (2x Intel Xeon Gold 6138 2.00Ghz)	185G	2x NVIDIA 2080Ti
Compute	node[01-11]	40 (2x Intel Xeon Gold 6138 2.00Ghz)	185G	
Very High Memory	vhmem01	80 (4x Intel Xeon Gold 6138 2.00Ghz)	3000G	
Enterprise GPU	egpu01	40 (2x Intel Xeon Gold 6138 2.00Ghz)	185G	2x NVIDIA V100

Table 3.3: HPC Hardware Specifications [53].

Using the cluster also means learning how to use Slurm [61], for which there were examples provided by the university. Although setting it up from scratch required some work, support is provided by the university, and not a third party, as it would be in the case of GCP. Below are the advantages of HPC.

- Unlimited computation.
- Somewhat customisable hardware specs.
- Reliable, submitted jobs are not interrupted.
- Multi-core CPUs available.
- Provided by the university.

As a disadvantage the pipeline to use the cluster needs to be set up with a script for the task manager. Moreover, task will enter a queue first, and run at the earliest opportunity, which could mean potential waiting times. Nevertheless, HPC was the most reliable option, as it allows us to submit pre-specified experiments and the system sends a notification once the job has finished. This allowed us to design more experiments while others were running. Listing 3.1 is the bash script used to submit the experiment for training. We have experimented with multiple CPUs (Macleod) and a GPU (Maxwell), and have found the second one to be significantly faster.

```
#!/bin/bash --login

#SBATCH --partition=uoa-gpu
#SBATCH --cpus-per-task=1
#SBATCH --ntasks-per-node=1
#SBATCH --mem=64GB
#SBATCH --gres=gpu:2
#SBATCH --mail-type=ALL
#SBATCH --mail-user=m.veiner.17@abdn.ac.uk
#SBATCH --nodes=1
```

```
#SBATCH --ntasks=1


date
hostname

module load cudatoolkit-10.1.168
module list

source activate pytorch
source /uoa/home/u02mv17/Repository/.venv/bin/activate

nvidia-smi
python /uoa/home/u02mv17/Repository/main.py

date
exit 0
```

Listing 3.1: Batch Job Submission Script.

3.3 Software Considerations

The project was written in Python [67], as it is the standard programming language for machine learning and has really powerful libraries for these purposes. No other languages have really been considered for the project.

At the early stages the Keras API [13] was used for quick prototyping, and assessing the feasibility of the project. However, since multiple capsule networks had to be integrated into the experiment, we would not have been able to enjoy the perks of Keras, and instead needed to work with a lower-level framework. Thus we mainly had two options left, which were TensorFlow [2] (the back-end of Keras) and PyTorch [50].

TensorFlow was developed by Google and released in 2015. It has a large and active user base and a significant amount of official and third-party tools and platforms for many purposes, including training, deploying, and serving models. Although the API of TensorFlow 1.0 was found rather cryptic by many [31], the 2.0 version has a much more Pythonic feel to it. The main benefits of TensorFlow are the community support, wide-range of third-party tools, intangibility with other Google products, for example Google Colab, and the availability of the Keras API.

PyTorch was developed by Facebook rather than Google, and has been released to the public a year later than TensorFlow. The quaility of the official tutorials [1], however, has made up for the late start, and it soon became wide-spread among the research community [31]. As PyTorch has its back end written in C, it offers fast matrix operations, and its API is more natural to Python programmers than TensorFlow.

Choosing the right library proved easier than the hardware (see Section 3.2.4), and in the end we have opted for PyTorch. This was partly because of its intuitive style and suitability for research, but mainly the availability of implementations for the capsule network variations we have chosen for the project (see Sections 3.5 and 4.2.2). With the framework decisions fixed, we can now discuss the overall architecture for our experiments.

3.4 Overall Architecture

Although the large resolution of histology images cause problems for any neural networks, this is especially true for capsule networks, as some variations can have serious memory bottlenecks [56]. As a result, even the 512 x 512 patches can pose difficulties. To avoid resizing directly and loosing valuable information, we have decided to use convolutional layers to reduce the image size, and learn an intermediate representation of the patches, which are then fed to the capsule layers.

The overall setup followed the architecture proposed in [45] (also based on [6] and more), which is a two-staged convolutional neural network trained on BACH. The first stage consists of training the so called the patch-wise network, and it operates on overlapping patches of size 512 x 512 which are created with a sliding window of size 512 x 512 pixels and stride 256. The patch-wise network is a series of convolutional layers and a fully connected layer for classification. This network is trained separately from the second stage, and its purpose is to learn a down scaled representation of the samples through the classification task. The architecture is designed so that the last convolutional layer outputs images of size 64 x 64 (we fixed the tunable parameter C in the original architecture of [45]).

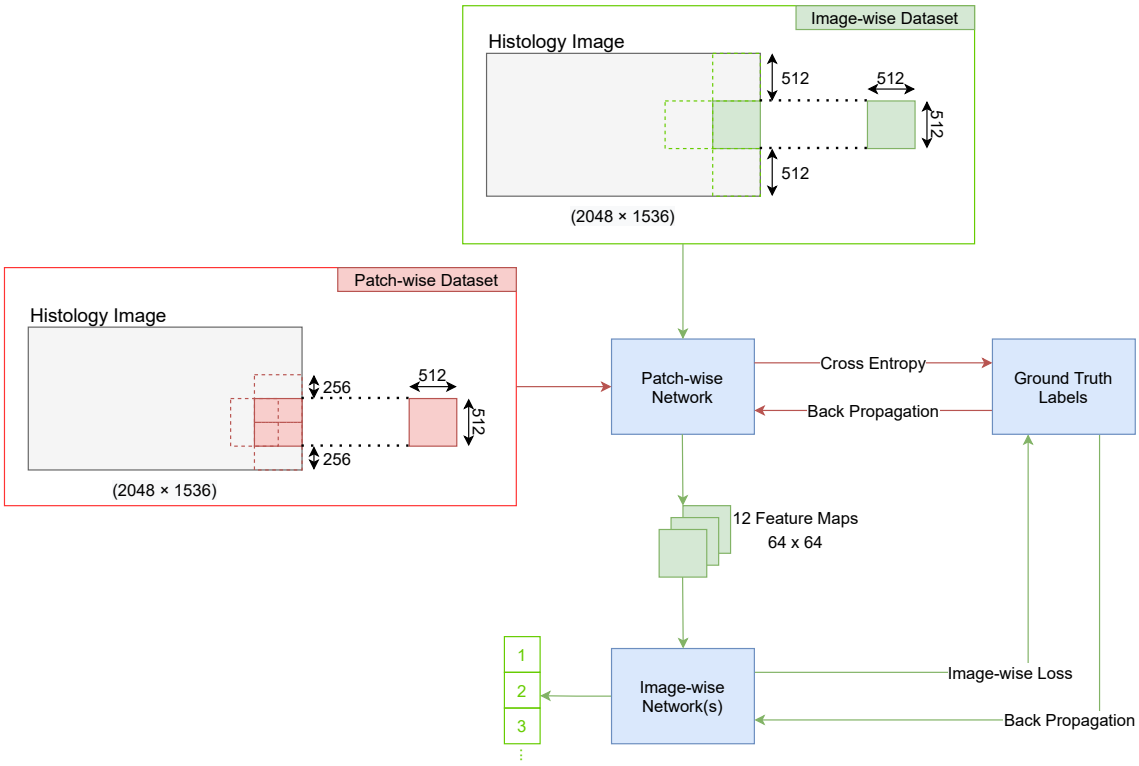


Figure 3.4: Overall Architecture Adapted from [45].

The patch-wise network proposed in [45] consists of 5 convolutional blocks, with each block containing 3 convolutional layers, plus ReLU activation and Batchnorm. Each block doubles the number of channels of its convolutional layers, starting from 16, i.e. the layers in the second block have 32 feature channels, the third have 64, etc. Lastly, a final convolutional layer and a dense layer for classification follow. We also experimented with by leaving out the last two blocks from

this architecture, which reduced the number of parameters in the model significantly.

After training the patch-wise network, the fully connected layer is removed, and the rest is used by the image-wise network (second stage) to obtain down sized image patches by feeding them through the trained convolutional layers of the patch-wise model. This time non-overlapping patches are used, and a batchful (12) of patches create the original image. The images are still 512 x 512, but created with a stride of 512. Figure 3.4, shows the overall process.

The 12 non-overlapping patches are fed through the trained patch-wise network, which down-scales them to 64 by 64 pixels. These smaller patches are fed to the image-wise network, which outputs 12 predictions (or 12 x X, probabilities, where X denotes the number of classes). These are then aggregated with majority voting, max or sum of probabilities, discussed below:

- Majority voting: The label of each patch is predicted, and the most popular label is the overall prediction.
- Sum voting: The probabilities for each label are summed, and the label with the largest probability is predicted.
- Max voting: The most confident prediction is the overall predicted label.

The winning label is the predicted class for the whole image. Figure 3.5 highlights this phase of the training.

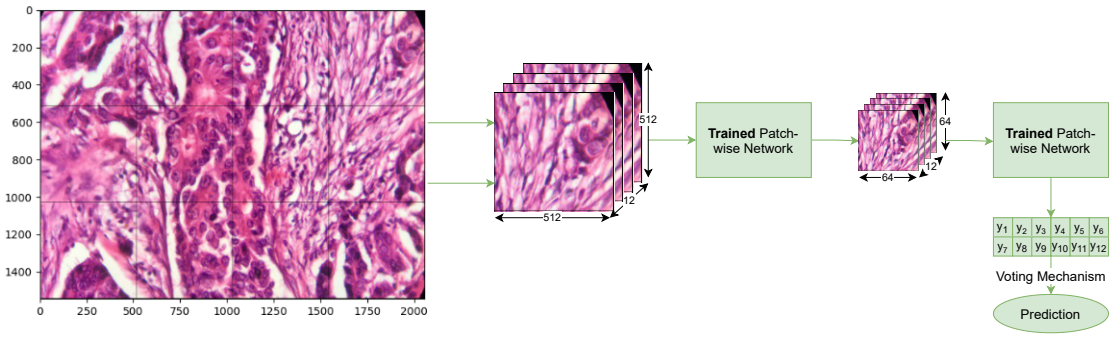


Figure 3.5: Image-wise Phase of Training.

Although the patch-wise network is fixed, because its purpose is not to classify but to down-size, but the image-wise network is not, and we may choose other models (like capsule networks) for classification. We will now present our choices for the image-wise stage, and more.

3.5 Chosen Networks

We have tested 5 image-wise models, 2 convolutional and 3 capsule networks. Additionally, we have experimented with 2 networks trained on their own, without the patch-wise network. Due to the lack of a better name, to these we will refer to as Mixed Networks. Here we briefly present our choices, architecture, and their implementation details are discussed in Section 4.2.

3.5.1 Image-wise Networks

We have designed two convolutional neural networks as baseline. One is the architecture proposed by [45], and it follows the same block-like structure described above. It contains two blocks with channels 64 and 128, another convolutional layer, and 4 fully connected layers with ReLU and Dropout, with nodes $128, 128, 64, X$, with X equaling the number of classes. In the experiment we refer to this model as "NazeriCNN". Additionally, we have designed a CNN with 3 convolutional and 3 fully connected layers, named "BaseCNN". For the architecture see Figure 3.6.

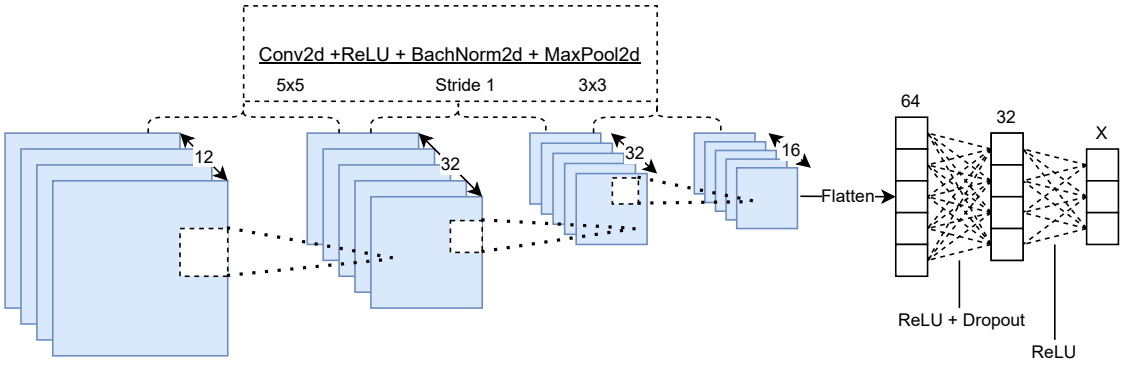


Figure 3.6: BaseCNN Architecture.

For the capsule networks, we have chosen Dynamic Capsules [57] as a baseline, with similar architecture as in the original study, only accommodating for images of size 64 x 64. We have also picked Self-Routing (SR) Capsules [25], which fixes many of the shortcomings of dynamic routing. Lastly, we have tested Variational Capsules (or VarCaps) [55], which significantly reduced the number of parameters in the model. We hypothesise that VarCaps will outperform other methods, except perhaps SR Capsules.

3.5.2 Mixed Networks

We have also tested many architectures in the exploratory phases of the project, but two of them were kept to obtain a baseline not only for capsules but for the two-staged approach. One is a mix of VarCaps with the 3 blocks of the patch-wise network built in i.e., trained at once. The idea was to test the separate phases of training with this approach.

Additionally, we have decided to try a pre-trained network. We picked EfficientNet B0 [65], which is a is a convolutional neural network architecture that scales better than other pre-trained networks, outperforming many, including ResNet and Inception, on ImageNet. The pre-defined and pre-trained EfficientNet B0 is loaded, and its final layer is changed, to be used for classification. The pre-trained weights were the frozen, training only the fully connected layer, a method known as transfer learning [49]. This way, the network reuses information about image recognition it has already learned. With these two models we were able to better establish the performance of the two-staged approaches.

Chapter 4

Implementation

This section describes how the networks were implemented, including sources, and libraries used, as well as the code structure. We will start by discussing the preprocessing steps for the described datasets.

4.1 Preprocessing

Regardless of the image-wise model we use, the overall architecture expects images (or patches) of 512 x 512 pixels in size. In [45] patches are generated from images on the fly, by looking up the correct rectangular region from the image through indexing. This method proved to be really slow as it introduces unnecessary computational steps. Reading in images one-by-one and instead unrolling them into a batch of patches turned out to be much faster. However, these patches are the same between runs, and thus we decided to save time overall by creating the patches and saving them beforehand. Figure 4.1 shows an example of a patched image from Databiox.

As an added benefit, dataloaders do not spend time resizing images. As this happens on the CPU rather than the GPU (by default) this can introduce a bottleneck in the training process, and slowe it down significantly. There is still some data preprocessing at runtime, including normalisation and colour jitter with hue and saturation both at 0.05, as well as concatenating the dataset with rotated and mirrored versions of it as in [45].

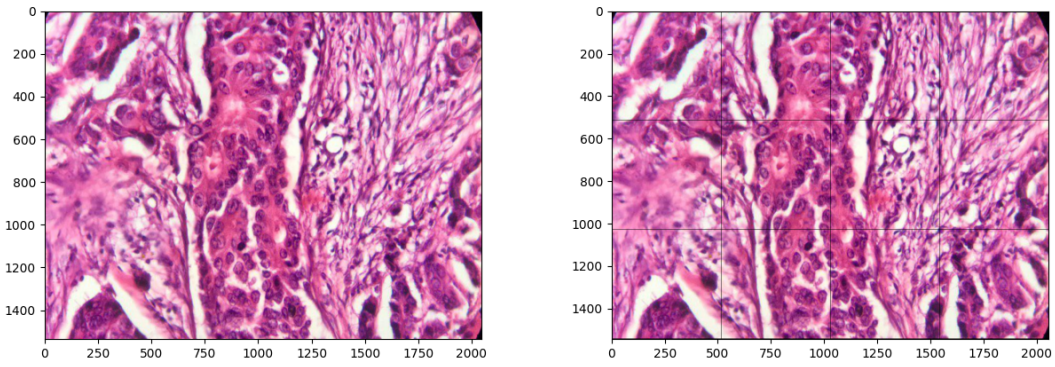


Figure 4.1: Creating Non-overlapping Patches for the Image-wise Networks.

Moreover, patches were sorted into training, test and validation sets while being created, which had proportions 70%, 15% and 15% respectively. The splits were done using the same random seeds, which is set by the code (see Listing 4.1). We also computed the means and standard

deviations per channels at this stage, however, initial results proved this not to be beneficial in all cases. The datasets.py script contains the functionality we have discussed so far, as well as plotting and checking the resolution distribution of a dataset (Databiox).

```
SEED = 123

def set_seed(seed=SEED):
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    np.random.seed(seed)
    random.seed(seed)
```

Listing 4.1: datasets.py - set_seed Function.

4.2 Code Structure

Although (Jupyter [35]) notebooks are good for writing quick experiments, and have been used in the early stages of the project, they are not efficient for continuous evaluation and development. Moreover, since we opted for HPC (see Section 3.2.4), we had to structure the code differently. The experiment is launched using *main.py*, where one can specify the network and training parameters. The source code is provided in the *src* folder, which has gone through many phases of refactoring.

Following the object-oriented style of PyTorch, networks were defined as classes, inheriting from the *nn.Module* base class, which defines methods such as setting the network into training / evaluation mode, using CPU / GPU, etc. The foundation for all our models is the *Model* class defined in *model.py*. This class defines most methods used by the networks in our experiment, including the core training and testing loops, saving and loading, plotting loss and accuracy metrics, and calculating different evaluation metrics. The training and testing loops also call the method *propagate*, which does forward propagation, and computes loss if relevant. This method was abstracted as it is dependent on the network, and can be overridden.

Patch-wise and image-wise networks (and mixed networks) inherit from the Model class. They also inherit methods from *nn.Module* such as setting the network into training or evaluation mode, as well as, the basic functionalities defined in the *Model* class, which are common for all networks in the experiment. Both types of networks are required to provide a *forward* method following the PyTorch style. Moreover, if a network is using different loss function, or calls the forward pass differently, they are required to override the *propagate* method. A summary of the class hierarchies can be seen on Figure 4.2. Note that class and method names are not listed explicitly, but by their functionality for conciseness.

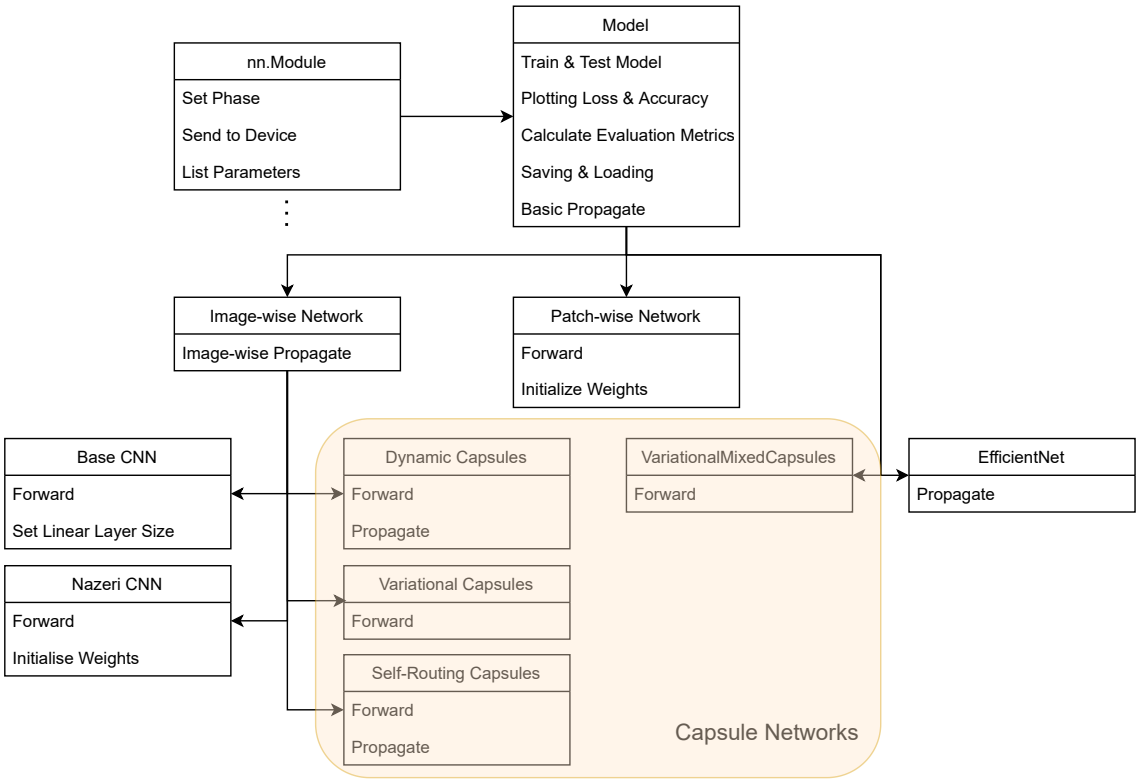


Figure 4.2: Class Hierarchies for Different Models.

4.2.1 Patch-wise Network

The patch-wise network is defined by the *PatchwiseModel* class found in *patchwisemodel.py*. Most of the functionality for this class comes from inheritance from the *Model* and *nn.Module* classes. The patch-wise network can be created using the 5 block architecture proposed in [45] or with simply 3 blocks, which output the same image size. The weights of convolutional layers are initialised as in [45], and the forward pass is defined. This class does not override the *propagate* method.

4.2.2 Image-wise Networks

Image-wise networks are defined in *imagewisemodels.py* and its base class is the *ImagewiseModels*, which inherits again from *Model*. This model has a bare-bones structure, and it defines a patch-wise model to be used for downscaling, and overrides the *propagate* method, to pass input images thourgh the patch-wise network first. As in Figure 4.2 patch-wise network variations inherit from this class.

Two convolutional networks have been implemented as image-wise models. The model defined by the *NazeriCNN* class has the same architecture as in [45] (discussed in Section 3.5.1), and the same weight initialisation method, as well as forward pass.

The other network is defined by the *BaseCNN* class. Its weight initialisation is handled by PyTorch. The only relevant change is the method *set_linear_layers*, which sets the number of nodes of the first linear layer to be the size of the flattened image after the convolutional layers. This became handy as PyTorch offers no convenient way of setting this parameter, apart from calculating or trying it out beforehand. The method automates this procedure.

Writing our own version of capsule networks was well outside of the scope of this project, therefore we had to resort to publicly available implementations, of which, fortunately, there are many [39]. Cut and slightly modified source codes for capsules are in the corresponding folders within the *src* folder, as well as their *Readme*-s and linceses. To reiterate, these scripts have not been created in this project, only modified.

For dynamic capsules, we have used the PyTorch implementation by the user XifengGuo [69], who has implementation of dynamic capsules for TensorFlow and Keras as well. The source code can be found in the *DynamicCaps* folder, but only containing the essentials such as definitions for the capsule layers (dense and primary [57]), and the loss function. The actual network is defined in *imagewisemodels.py* by the *DynamicCapsules* class, following the source code of XifengGuo, and adapted to work with our overall architecture. The class includes an override for propagate.

The variational capsules were based on the code published by the author of the study, which introduced the idea [55]. The source code can be found in the *VarCaps* folder, and it contains definitions of the convolutional capsule layers, primary capsule layers, and the routing mechanism based on variational bayes. The image-wise network is defined following the model definition of the original code [18], but with its final kernel sized changed to work with our image sizes. The class *VariationalCapsules* can also be found in the *imagewisemodels.py* file.

Lastly, the self-routing capsules were based on the official implementation by the authors of [25]. The *SRCaps* folder contains the relevant source code, which only consists of a definition for the self-routing capsule layer. The model is defined by the *SRCapsules* class following the original architecture and code [16] with, again, the final kernel size change to fit the image sizes. This repository actually contains code for other capsule variations as well, including dynamic capsules, however, it has only been found after dynamic capsules have been integrated.

4.2.3 Mixed Networks

The mixed networks can be found in the *mixedmodels.py* file. One mixed network is a variational capsules (class *VariationalMixedCapsules*), again based on [18], but with the patch-wise layers (3 blocks) included within the model. This way the patch-wise layers are adjusted upon training as well.

The other mixed network is the EfficientNet B0 based on [41] and is defined in the class *EffNet*. The initialisation creates the pre-trained EfficientNet, and redefines its classification layer with the number of classes depending on the dataset. The parameters of pre-trained layers are then frozen, in order to make use of transfer learning. We have not experimented with fine-tuning the pre-trained layers through lower learning rates. *EffNet* is the only class that does not define a forward pass, because the pre-trained network has it defined.

Chapter 5

Results and Evaluation

This section presents the main findings of this project. That is the performance of the models discussed so far, and their evaluation. This will be presented for each dataset separately. In all cases below, the Adam optimiser [34] has been used to find the minimum for the cross entropy loss, or the specific capsule loss.

5.1 BreakHis

The patch-wise model on BreakHis was trained with 3 blocks using a batch size of 32, and learning rate 0.001. The model contained 119, 141 parameters, all of which were trainable. It used 44,288 training samples 38,752 of which were augmented images, using rotations and mirroring. The training was planned to run for 30 epochs, but it timed out after completing only 18. This posed no problem however, as the network started overfitting from epoch 4 (see Figure 5.1). As constant checkpoints ensured that the best model we have encountered so far is always saved, we have used the model as in epoch 4 for training with image-wise networks.

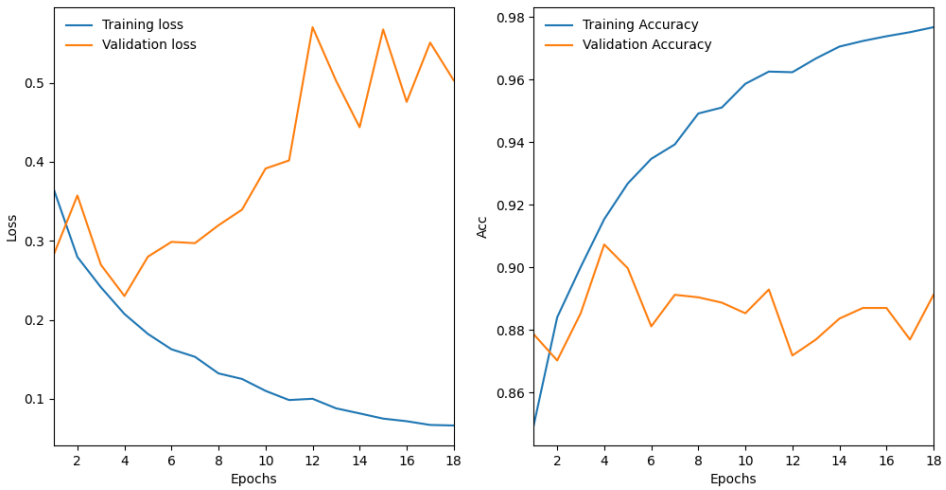


Figure 5.1: Patch-wise Network on BreakHis.

The training accuracy of the model was 0.9154. For validation we have used 1,186 samples and the model achieved 0.9073 accuracy. The test set consisted of 1,187 samples, and although its primary use was not to assess the performance of the patch-wise network (as it will not be used

for classification) but to serve as a subset of the data, which the model has not seen. Regardless, we assessed the model's accuracy on the test set, and it achieved 0.91 accuracy, 0.95 sensitivity, 0.83 specificity, and 0.94 F1-score. As we defined these metrics for multi-class classification (see Terminology) it is important to note that we have calculated these metrics by considering benign as our positive class and malignant as negative. The confusion matrix is visible below.

		True diagnosis		
		Malignant	Benign	Total
Predicted	Malignant	302	64	366
	Benign	39	782	821
	Total	341	846	**1,187**

Overall this gave the image-wise networks a good start, which were trained for 10 epochs using the loaded and pre-trained patch-wise net. We have assessed the performance of the BaseCNN, the CNN presented in [45] (NazeriCNN), dynamic capsules (DynamicCaps), self-routing capsules (SRCaps) and variational capsules (VarCaps). All models have been trained on the same training set and evaluated on the same validation and test set as the patch-wise network (see discussion in Section 3.1). The learning rate was set to 0.001 and the batch size was 32 with the exception of VarCaps, where the memory bottleneck required us to set the batch size to 16 instead. This decision could explain why VarCaps were lagging behind the other capsules in performance during training (see Figure 5.2).

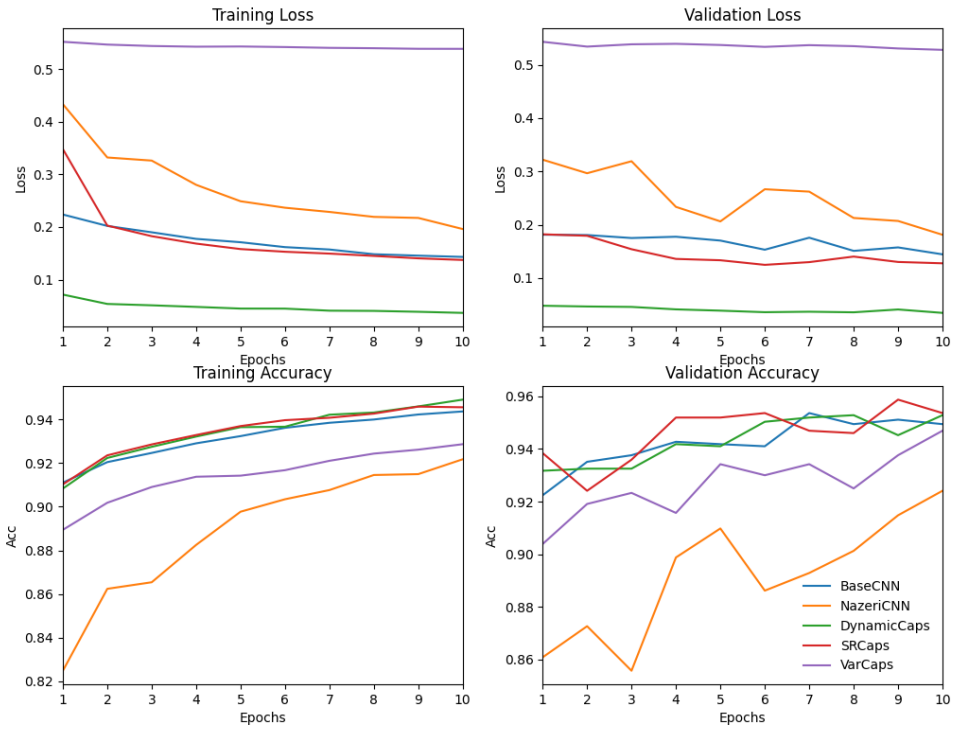


Figure 5.2: Image-wise Networks on BreakHis.

All capsule networks used three rounds of routing. Dynamic capsules used a reconstruction loss coefficient of 0.392, which was the default in [69]. For VarCaps we used 4 pose dimensions for the capsules and architecture parameters of 64, 16, 16 and 16 (see [18]). None of the models presented here have used pre-calculated metrics for image normalisation. The number of trainable parameters in each model is presented in Table 5.1.

Overall the image-wise networks achieved competing performance, with SRCaps achieving the highest validation accuracy (0.9587) at epoch 9. Although VarCaps was lagging behind at the beginning it caught up towards the end (see Figure 5.2), and given more time to train it might have turned out to be the best performing image-wise network. We summarise the evaluation metrics of these models together with the mixed networks in Table 5.1.

We have also trained variational capsules with built in CNN layers (VarMixedCaps) and EfficientNet (EffNet). As discussed only the last fully connected layers of the EfficientNet were trainable, that is 164,226 parameters out of the total of 4,171,774. EffNet has been trained using a batch size of 64 and VarMixedCaps used a batch size of 32 for the same reasons as above. Both models used the same number of samples as the image-wise nets for training, testing and validation, with a learning rate of 0.001. As we can see EffNet outperformed all of our models and achieved near perfect performance (see Figure 5.3).

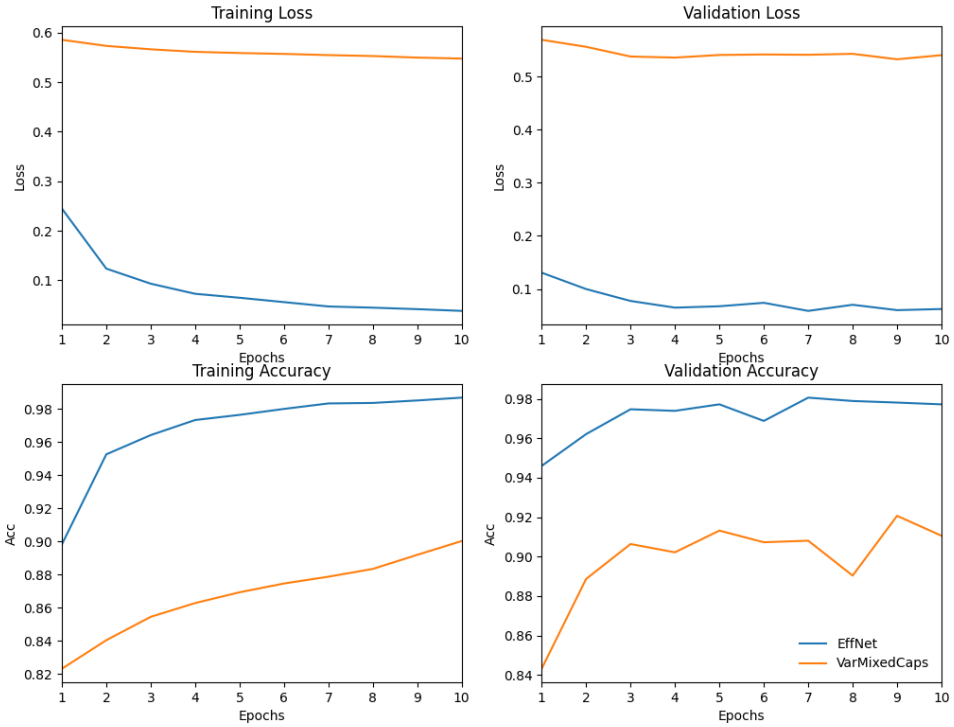


Figure 5.3: EfficientNet and VarMixedCaps on BreakHis.

As we have not created a patched dataset (see Section 3.1) we did not employ any voting mechanism, and the predicted labels were final. As presented in Figure 5.2, EffNet achieved the

best performance, and SRCapsule scored the highest among image-wise networks. All image-wise models achieved a test accuracy of 0.95, but SRCaps performed slightly better than others with F1 score. Note also that variational capsules (VarCaps and VarMixedCaps) had significantly less parameters than other networks (EffNet had many non-trainable). The fact that it was able to generalise so well with less than 10% of the number of parameters of other models, is an achievement on its own.

Model	Batch	Trainable Params.	Train Acc.	Val. Acc.	Test Acc.	Sens.	Spec.	F1
BaseCNN	32	2,329,095	0.96	0.95	0.95	0.95	0.94	0.96
NazeriCNN	32	520,488	0.94	0.92	0.94	0.95	0.94	0.90
DynamicCaps	32	14,783,653	0.96	0.95	0.95	**0.98**	0.90	0.95
SRCaps	32	2,634,757	0.96	0.96	0.95	0.96	0.93	**0.97**
VarCaps	16	**197,145**	0.93	0.95	0.95	0.95	0.96	0.96
VarMixedCaps	32	311,831	0.89	0.92	0.93	0.93	0.94	0.95
EffNet	64	164,226	**1.00**	**0.98**	**0.98**	0.97	**0.99**	**0.97**

Table 5.1: BreakHis Performance Metrics Summary.

Overall all the models achieved promising performance on BreakHis, but transfer learning proved to be inferior. The predictions for SRCaps and EffNet can be seen on the confusion matrices on Figure 5.4. We will now move on and increase the number of classes in our dataset by one.

	True diagnosis		
	Malignant	Benign	Total
Malignant	342	24	366
Benign	33	788	821
Total	375	812	**1,187**

	True diagnosis		
	Malignant	Benign	Total
Malignant	355	11	366
Benign	8	813	821
Total	363	824	**1,187**

Figure 5.4: Confusion Matrix for SRCaps (left) and EffNet (right).

5.2 Databiox

Databiox proved to be a more difficult dataset than BreakHis, which is why we experimented with many versions of the patch-wise networks (see Figure 5.5). The 3 block version of the patch-wise model (Patchwise) quickly rose to a validation accuracy of around 0.50 but further training showed that the model was unable to continue learning. Using the original 5 block architecture (Patchwise_Original) we managed to improve on this and achieve a validation accuracy of 0.563 in 20 epochs. Although this is not an impressive achievement, all models converged around this point, even after removing data augmentation and training for longer (Patchwise_Original_NoAug).

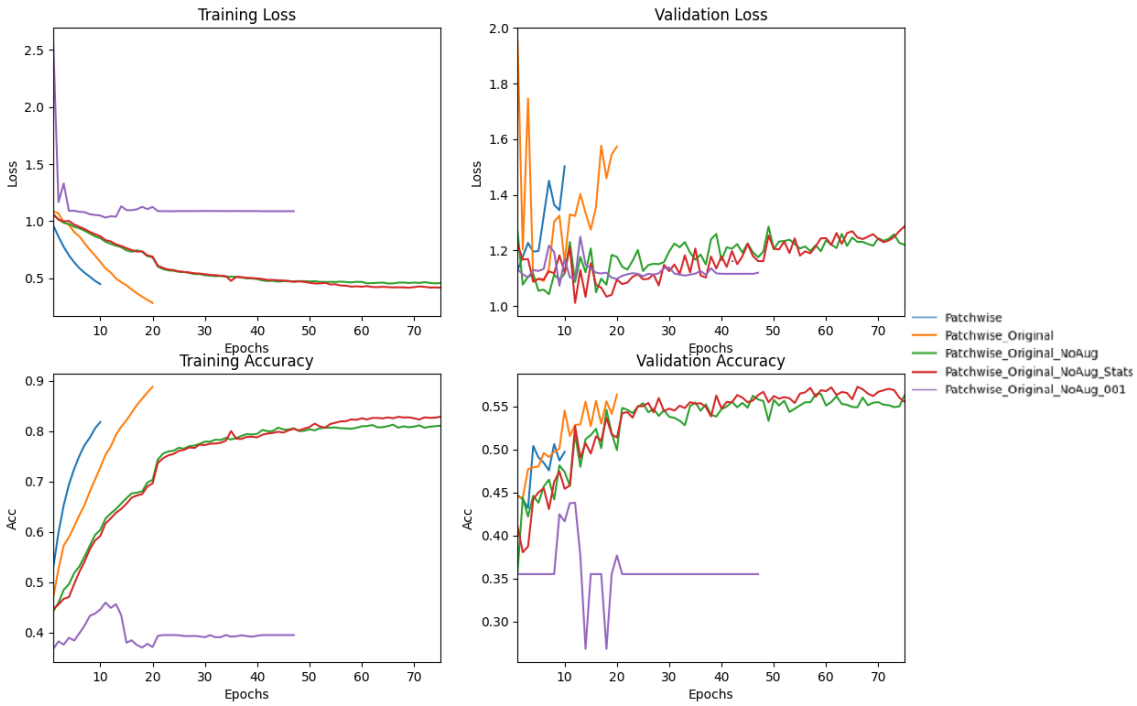


Figure 5.5: Patch-wise Networks on Databiox.

We can also observe that using the pre-computed statistics to normalise the images further improved the accuracy, altough not significantly (Patchwise_Original_NoAug_Stats). Lastly, increasing the learning rate from 0.001 to 0.01 (Patchwise_Original_NoAug_001) halted the learning process altogether. All models used a batch size of 32.

The best performance was achieved by Patchwise_Original_NoAug_Stats at epoch 66. The training, validation and test accuracy were 0.5727, 0.87 and 0.64, respectively. The model predicted all 3 labels with equal frequency and the averaged F1 score[1] was 0.69.

The image-wise networks were trained on 30,960 training samples, validated on 1,656 and tested on 1,668 images with the learning rate set to 0.001. Batch sizes and the results can be seen on Figure 5.6 and Table 5.2. Overall, all models struggled to learn the dataset, and although the training accuracy was slowly increasing in all cases, the performance on the validation set was poor and inconsistent. Although EffNet achieved slightly better performance than the rest, it too converged to the same level of validation accuracy.

[1]The F1 score was computed for each label separately treating them as the positive samples and all other labels as negative. The averaged F1 is the mean of these three F1 scores.

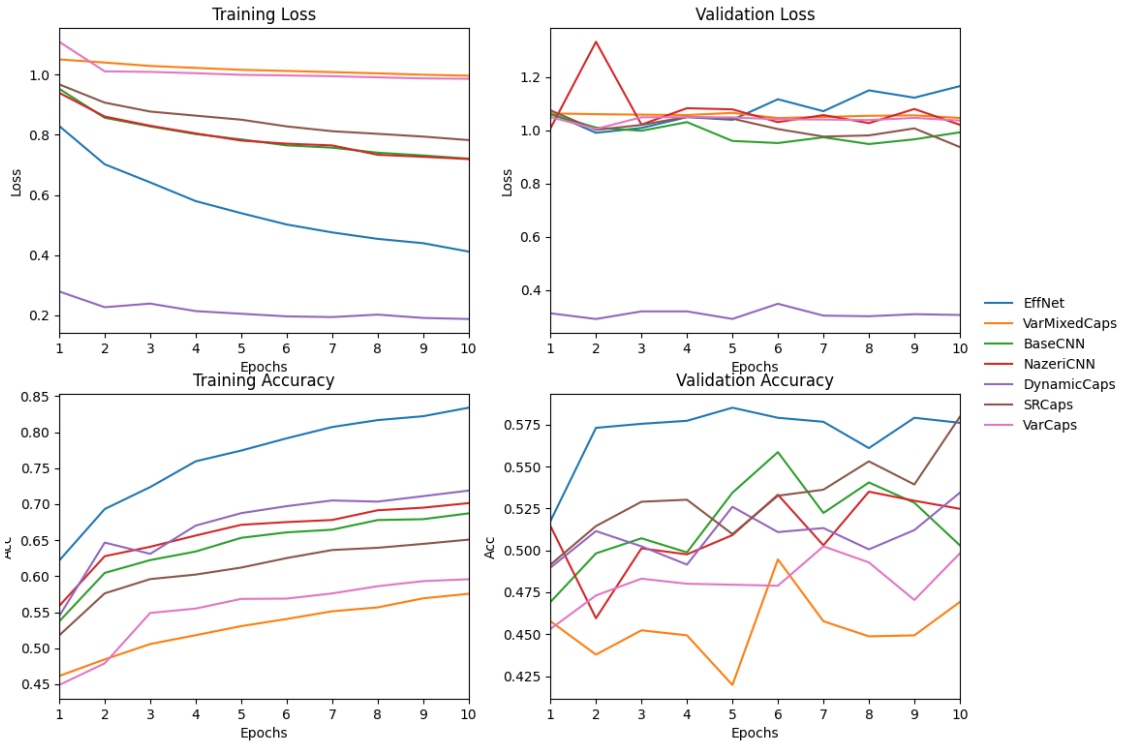


Figure 5.6: Image-wise Networks on Databiox.

Table 5.2 also shows that the voting mechanism helped on the performance overall, with 0.71 being the highest test accuracy achieved by EffNet through majority voting. We can also see that variational capsules were significantly behind in training accuracy, which is likely due to the small(er) batch size that was needed to lower the working memory needs of variational capsules.

Model	Batch Size	Training Accuracy	Validation Accuracy	Test Accuracy	Maj. Vote	Sum Vote	Max Vote
BaseCNN	32	0.69	0.56	0.57	0.63	0.66	0.66
NazeriCNN	32	0.72	0.54	0.58	0.63	0.63	0.65
DynamicCaps	32	0.73	0.53	**0.61**	0.64	0.63	0.63
SRCaps	32	0.71	0.58	**0.61**	0.64	0.65	**0.67**
VarCaps	8	0.58	0.50	0.55	0.58	0.60	0.58
VarMixedCaps	8	0.54	0.49	0.52	0.57	0.55	0.54
EffNet	32	**0.84**	**0.59**	**0.61**	**0.71**	**0.68**	0.64

Table 5.2: Databiox Performance Metrics Summary.

It is apparent from Table 5.2 that none of the models surpassed the best achieving patch-wise model in test accuracy. As patch-wise networks all converged to the same level of validation accuracy, it is possible that the image-wise networks suffered from the lack of fit of the patch-wise net. However, even VarMixedCaps and EffNet could not generalise fully, which hints at the issues lying somewhere else. We will leave further discussion on these results for the next chapter and move on now to our last dataset.

5.3 BACH

The first patch-wise models on BACH have been trained with a batch size of 64 and with half of the augmented images i.e., with rotations enabled, but no mirroring. This resulted in 39,208 training, 2,100 validation and 2,110 test samples. However, the models achieved significantly worse performance than on BreakHis, which is why we have experimented with some of the hyperparameters. 0.01 learning rate proved to be more erratic than 0.001, but using the original 5 block architecture (named "Original" on Figure 5.7), canceled the smoothing effect.

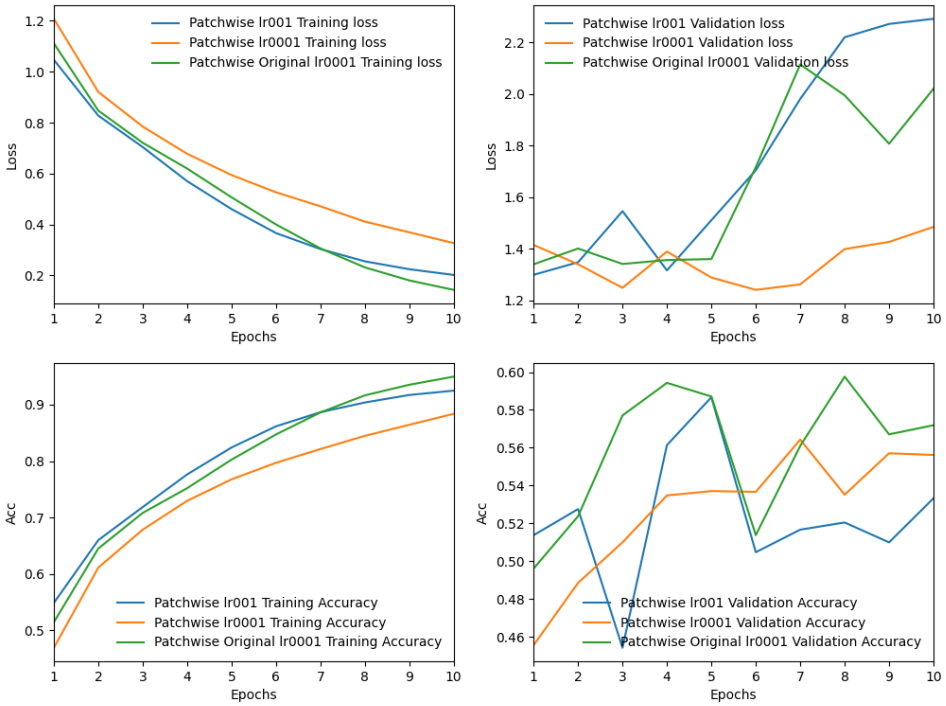


Figure 5.7: Patch-wise Network on BACH Showing the Effect of Learning Rate and Architecture.

Overall using the original architecture resulted in a slight increase in performance, achieving 0.5976 validation accuracy and 0.60 test accuracy at epoch 8. This model has been trained for a further 10 epochs (totaling 20 epochs) with the hopes of achieving higher accuracy, but it never surpassed 0.59, even though the training accuracy continued to rise. In fact, others have also been reporting being stuck at a local maximum, see Issue 9 on the repository of the original implementation [44] (although they used different parameters).

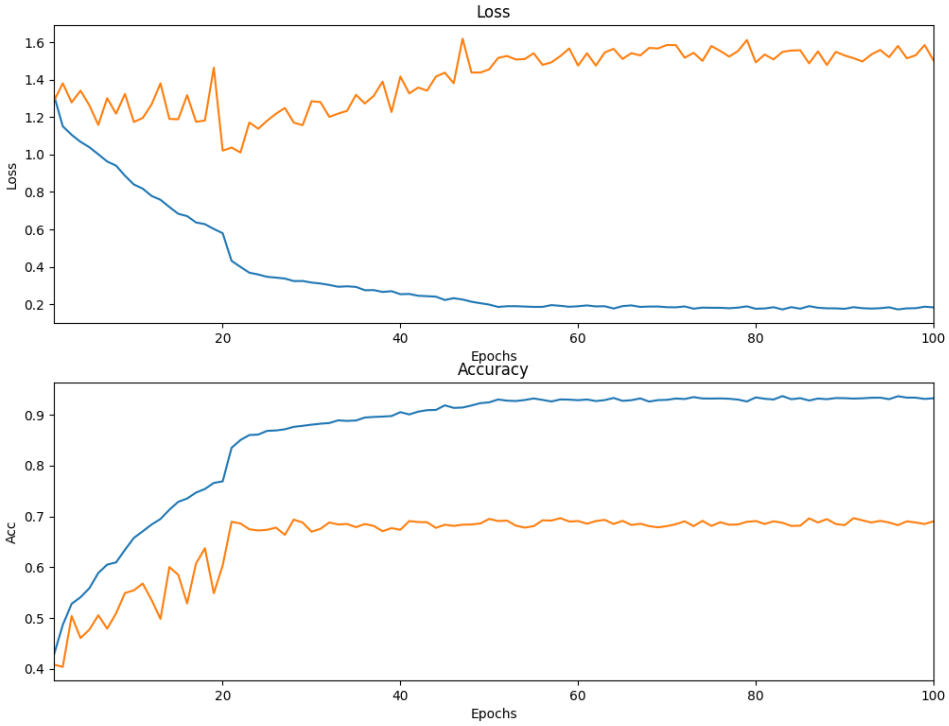


Figure 5.8: Patch-wise Network on BACH without Data Augmentation.

We have also noticed that data augmentation did not have a significant effect on training. Thus the final patch-wise network for BACH was trained with the original 5 block architecture, 0.001 learning rate, batch size of 64 and without additional augmented samples, that is with only on 9,800 training samples, but for 100 epochs. As we can see on Figure the network converges after 20 epochs on a validation accuracy of 0.69. The maximum was attained at epoch 58 with a validation accuracy 0.696 and training accuracy 0.96 and test accuracy of 0.71.

The image-wise and mixed networks were trained with half of the augmented images, i.e. using 13,440 training and 720 validation samples. The learning rate was set to 0.001 and the batch sizes can be seen in Table 5.3. All models used the same parameters as before except for VarCaps, which smaller layer sizes (64, 16, 8, 8). It seemed, however, that all image-wise networks stopped learning early, as none surpassed the patch-wise network in validation accuracy, and only the BaseCNN achieved better test accuracy than the patch-wise net with 0.72. EffNet, however, again surpassed the rest, achieving 0.75 validation accuracy at epoch 10. The summary metrics for each model can be seen in Table 5.3 as well as the results of the patch votes.

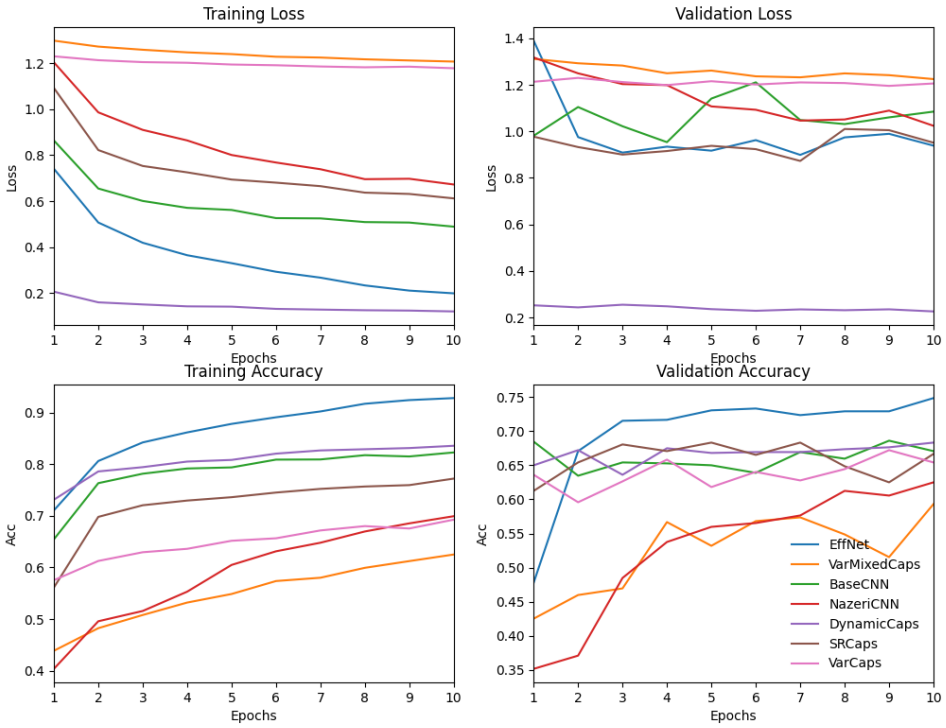


Figure 5.9: Image-wise Networks on BACH.

As we can see the voting scheme offered further improvements in performance for all models, but even with that most of them performed suboptimally. Note that for VarMixedCaps the voting results have not been assessed, due to technical difficulties during training, but we hypothesise that their performance would have been inferior to EffNet (and the others). In general majority voting seemed the most accurate across all models.

Model	Batch Size	Training Accuracy	Validation Accuracy	Test Accuracy	Maj. Vote	Sum Vote	Max Vote
BaseCNN	32	0.92	0.69	0.72	0.83	0.77	0.75
NazeriCNN	32	0.80	0.63	0.61	0.73	0.68	0.58
DynamicCaps	32	0.92	0.68	0.67	0.75	0.78	0.73
SRCaps	32	0.86	0.68	0.68	0.77	0.75	0.70
VarCaps	16	0.63	0.67	0.64	0.79	0.74	0.70
VarMixedCaps	16	0.63	0.59	0.55	-	-	-
EffNet	64	**0.98**	**0.75**	**0.81**	**0.90**	**0.88**	**0.87**

Table 5.3: BACH Performance Metrics Summary.

Chapter 6

Conclusion and Future Work

This project aimed at using capsule networks for breast cancer classification and comparing them with their counterparts, convolutional neural networks. For this, we have implemented a two-stage approach that works on image patches and tested it on three different datasets. Here we briefly discuss the results, draw conclusions and propose directions for future work.

6.1 Discussion

As we have seen (see also Table 6.1) the results were mixed. We can say that, overall, EffNet clearly outperformed other approaches across all datasets, with SRCapsules competing in some metrics. It is possible that this was due to the fact that other models have not seen as many images as EffNet and did not have enough time to learn everything. As EffNet was the only model significantly differing from the rest, in that it did not have the layers of the first stage, these results can also be inherent to our experiment.

Dataset	Model	Training Accuracy	Validation Accuracy	Test Accuracy	F1	Maj. Vote	Sum Vote	Max Vote
BreakHis	BaseCNN	0.96	0.95	0.95	0.96		N/A	
	NazeriCNN	0.94	0.92	0.94	0.90			
	DynamicCaps	0.96	0.95	0.95	0.95			
	SRCaps	0.96	0.96	0.95	**0.97**			
	VarCaps	0.93	0.95	0.95	0.96			
	VarMixedCaps	0.89	0.92	0.93	0.95			
	EffNet	**1.00**	**0.98**	**0.98**	**0.97**			
Databiox	BaseCNN	0.69	0.56	0.57	0.57	0.63	0.66	0.66
	NazeriCNN	0.72	0.54	0.58	0.58	0.63	0.63	0.65
	DynamicCaps	0.73	0,53	**0.61**	0.59	0.64	0.63	**0.67**
	SRCaps	0.71	0.58	**0.61**	0.61	0.64	0,65	**0.67**
	VarCaps	0.58	0.50	0.55	0.55	0.58	0.60	0.58
	VarMixedCaps	0.54	0.49	0.52	0.52	0.57	0.55	0.54
	EffNet	**0.84**	**0.59**	**0.61**	**0.64**	**0.71**	**0.68**	0.64
BACH	BaseCNN	0.92	0.69	0.72	0.71	0.83	0.77	0.75
	NazeriCNN	0.80	0.63	0.61	0.61	0.73	0.68	0.58
	DynamicCaps	0.92	0.68	0.67	0.67	0,75	0.78	0.73
	SRCaps	0.86	0.68	0.68	0.68	0.77	0.75	0.70
	VarCaps	0.63	0.67	0.64	0.64	0.79	0.74	0.70
	VarMixedCaps	0.63	0.59	0.55	0.55	-	-	-
	EffNet	**0.98**	**0.75**	**0.81**	**0.80**	**0.90**	**0.88**	**0.87**

Table 6.1: Overall Performance Metrics Summary.

Within the two-stage approach, that is between the image-wise networks capsules scored

higher than other convolutional baselines, but not by far. It is likely that the patch-wise phase of the training dominated the overall performance of all models, which then only had little room to diverge.

The best results have been obtained on BreakHis, on which although we have used the two-staged approach, we did not create separate patched datasets. Thus, it is possible that the lower performance on Databiox and BACH can be explained by the data preprocessing. We argue that patching was indeed the correct step to take, as it was the only feasible option for the other two datasets, but further experimentation with the separate datasets and finding the correct amount of overlap for patch-wise could help on improving these results. Let us now recall our objectives in order to discuss what has been achieved in this project. In the introduction we listed the following goals:

1. Establish a baseline for convolutional neural networks on Databiox.
2. Design a capsule network that outperforms the baseline on Databiox.
3. Achieve competing performance with capsule networks on BACH.
4. Achieve competing performance with capsule networks on BreakHis.
5. Evaluate on the binary dataset from the medical school (Optional).

As we have promised, we have established a baseline on Databiox, held by Efficient Net with 0.71 voting accuracy on the test set. We remain sceptical about the fact that we could find no studies reporting performance on Databiox and the varying image sizes, as opposed to they reported in [9]. It is possible, however, that the grade of cancer is more subtle and difficult to identify than its type, and the problem requires further considerations.

As for our second objective, we have seen that capsules only achieved competing performance on Databiox at most. They all achieved a test accuracy of 0.61, and dynamic and self-routing capsules got higher max voting accuracy, but EffNet scored higher overall. If we allow ourselves to only focus on the image-wise approaches, however, then most capsules performed better than the CNN baselines, with the exception of variational capsules, which was again likely due to the low batch size.

To compare our results with others on BACH, see Table 6 (*Comparison of our approach against the results of those from the state-of-the-art with the breast cancer classification scenario.*) in [4]. The range of the accuracy of listed models is from 0.875 to 0.9751, with the best performance achieved via transfer learning. The best accuracy capsules have achieved on BACH was by dynamic capsules using sum voting, but even this is lower than the ones listed in the study. EffNet, however, did achieve competing performance beating 4 models in Table 6, if majority voting is used.

On BreakHis both EffNet and capsule networks (for example SRCapsules) achieved competing performance, as they almost achieved perfect scores. As an example, in [10] the average accuracy (over magnification levels) was 0.9842, again achieved by transfer learning. In Table 6 (*Performance comparison with MI state-of-the-art counterparts.*) and (Table 7 - 10) in the study, the performance of other approaches are listed, comparing our results to these we have in fact achieved satisfying results.

Unfortunately, the ongoing situations made collaborating with the medical school difficult to organise, and with the many approaches and 3 different datasets, our efforts have already been quite distributed. Thus, this optional goal has not been achieved, even though it was one of the biggest selling points of the project.

6.2 Conclusions

This project proved to be quite challenging for many reasons. The main motives for choosing this project were to pivot into deep learning and the field of biomedical AI. This posed the difficult task of learning the principles of both of these fields, and creating a good project at the same time. Even though the results obtained in this paper were mixed, this project has already been a success, in that it served as an introduction to said fields.

Among the many challenges was finding a good platform for the project to run on. We have discussed several option in Section 3.2, but as the research on the computational resources has been carried out while working on the project, and as access to Maxwell has been granted only towards the end, all platforms (except for Google Cloud Services) have been used at some point to run experiments. Nevertheless, none of them were optimal. Kaggle only allowed limited GPU quota per week and it deletes working memory as soon as it finished running, and Google Colab is only available for a limited amount of time, which is not shown to the user. Using the Maxwell cluster, however, meant waiting in the queue. There was also one week towards the end when Maxwell was unavailable, which also meant setback.

On the other hand, capsules remained a novel approach that held up to convolutional neural networks. Variational capsules, for example, managed to achieve competing performance with only a small portion of the parameters that other models had. Their current implementation, however, relies on large matrix multiplication, which introduced a memory bottleneck during training. In general, capsules should be used for smaller image sizes, which can restrict their relevance somewhat. We have also seen that performance is largely influenced by other decisions in the experiment, most of which do not need to be made in other areas of image classification. For example, patching medical images is a common approach, as we have seen, as well as using votes to retrieve the overall prediction of the whole image. Data augmentation and preprocessing was also more influential than in other classification tasks.

The two-staged approach, however, turned out to be difficult to manage. There was a constant pressure to spend more time on improving the performance patch-wise model, as they seemed to be driving the overall results. Yet doing so wasted precious time from the image-wise networks, and trying a new patch-wise approach meant rerunning the other experiments as well.

All in all, capsules are a fair competition to CNNs even in medical imaging, but transfer learning remained the strongest. We will now follow with proposing directions for future work.

6.3 Future Directions

Given the promising results on BreakHis, with some fine-tuning and further training it would be possible to beat the baselines listed in [10]. For the other datasets, another approach would be most recommended. Creating smaller patches of the images, and feeding them directly to the networks could possibly achieve better performances, but in this case finding the right patch size would be key.

There are also other types of capsule networks that we have not tried, for example EM capsules and self-supervised capsules. Also new (and surprising) uses for capsule networks still surface, for example natural language processing [70; 71], which are worth investigating. It would also be insightful to combine capsule networks with transfer learning, either by putting capsules on top, or using pre-trained capsule networks from other datasets.

Lastly, in real settings these models would be used by radiologist, who would be also interested in the reasons for a prediction. Thus highlighting areas of interest via class activation maps [73] as in [52] would prove to be as useful, if not more, as predicting with high accuracy.

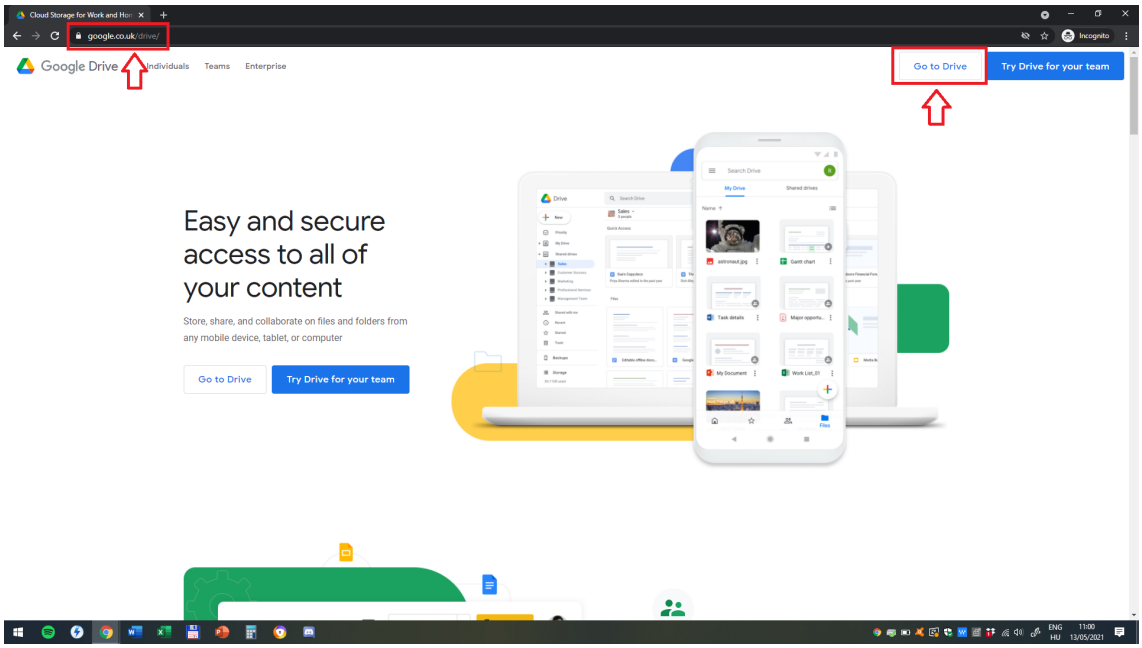
Chapter 7

Appendices

7.1 User Manual

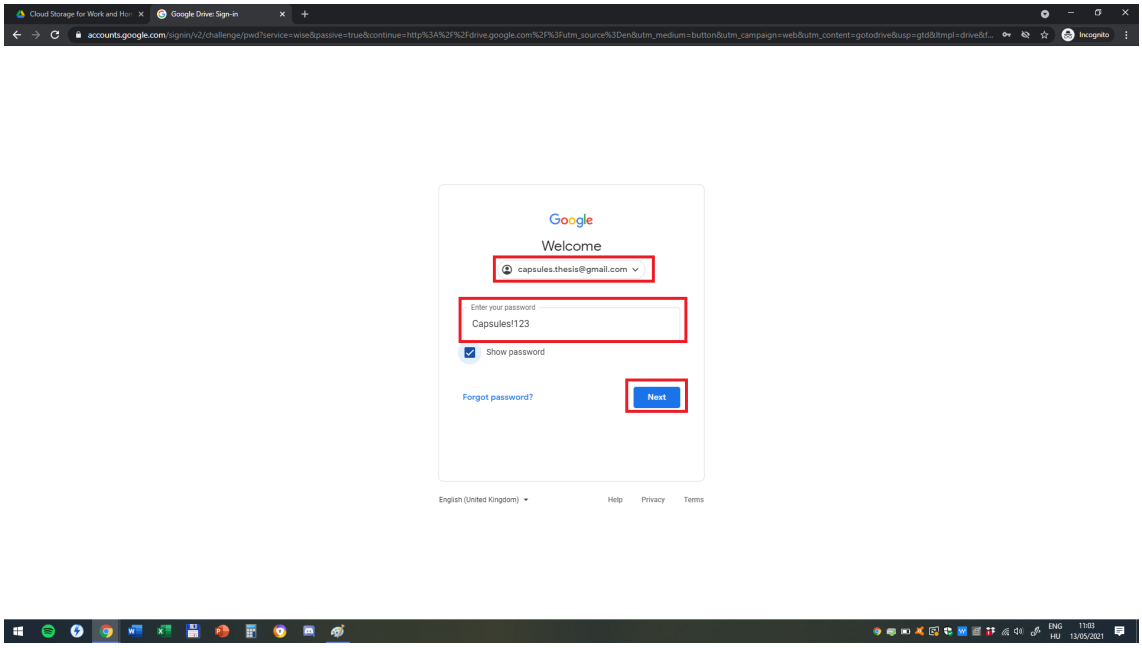
In this manual we briefly present how to replicate the results in this project. The easiest way is through Google Colab, for which I have created an account. Please follow the steps below:

1. Navigate to `https://www.google.co.uk/drive/` and presss “Go to Drive”.

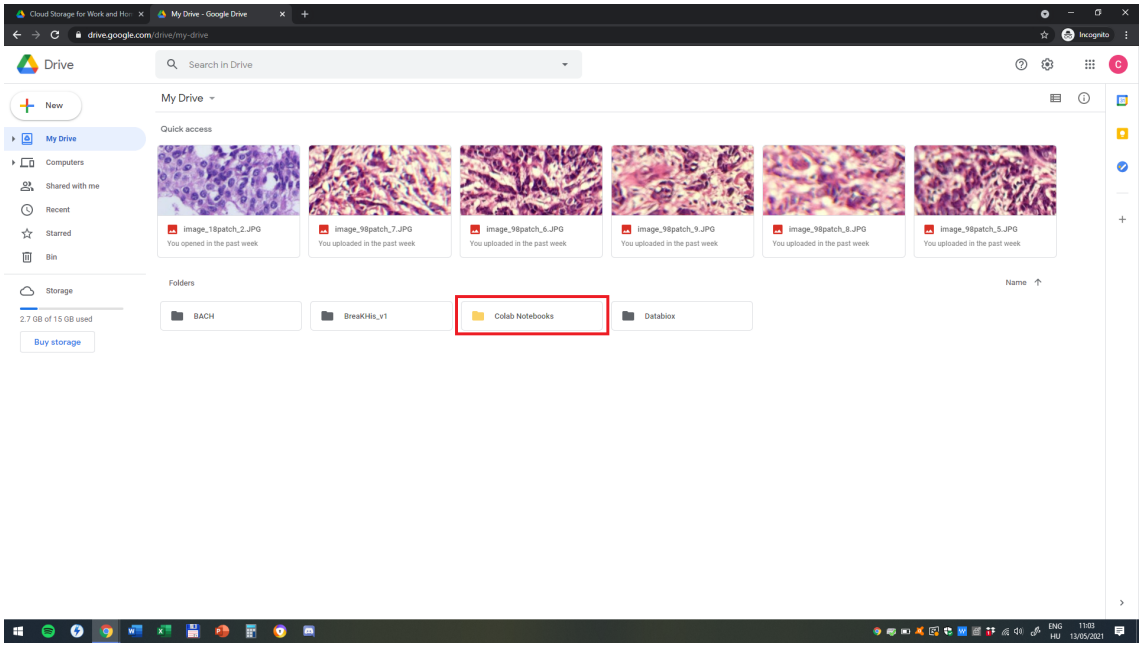


2. When prompted log in with the following credentials:

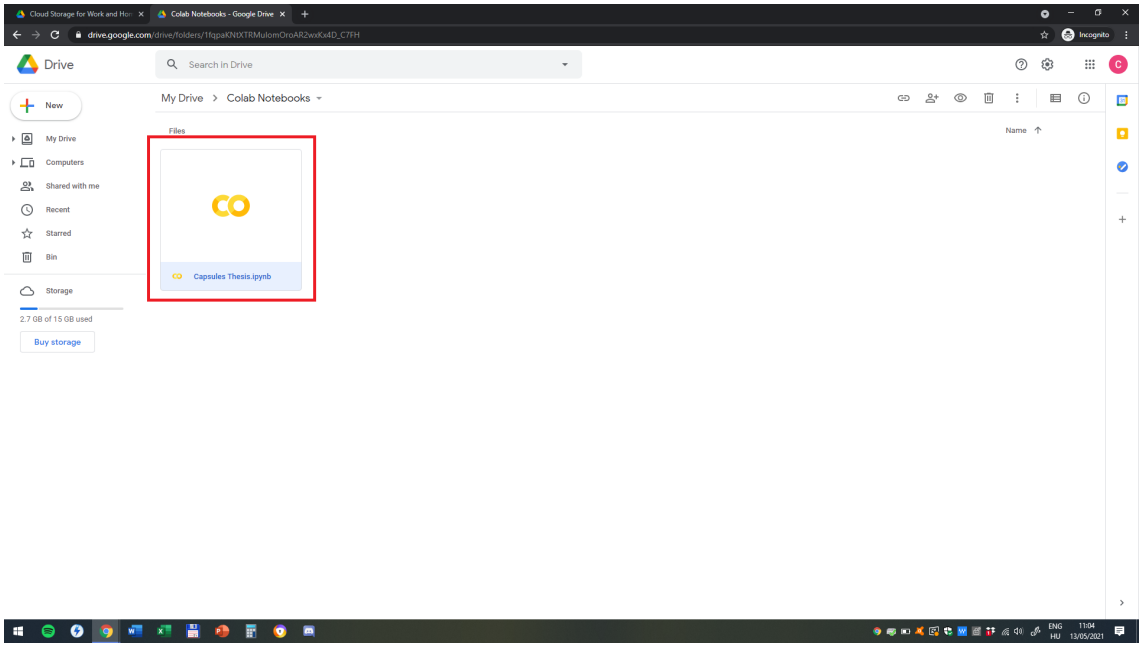
   - Username: capsules.thesis@gmail.com
   - Password: Capsules!123



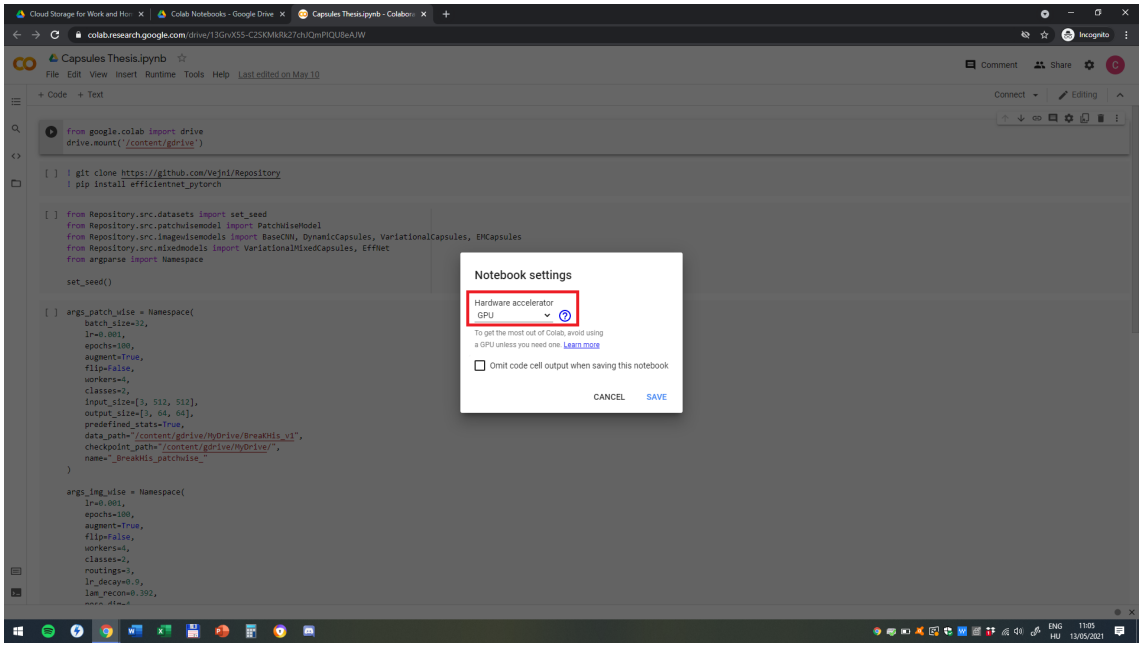
3. Double click on the Colab Notebooks folder.



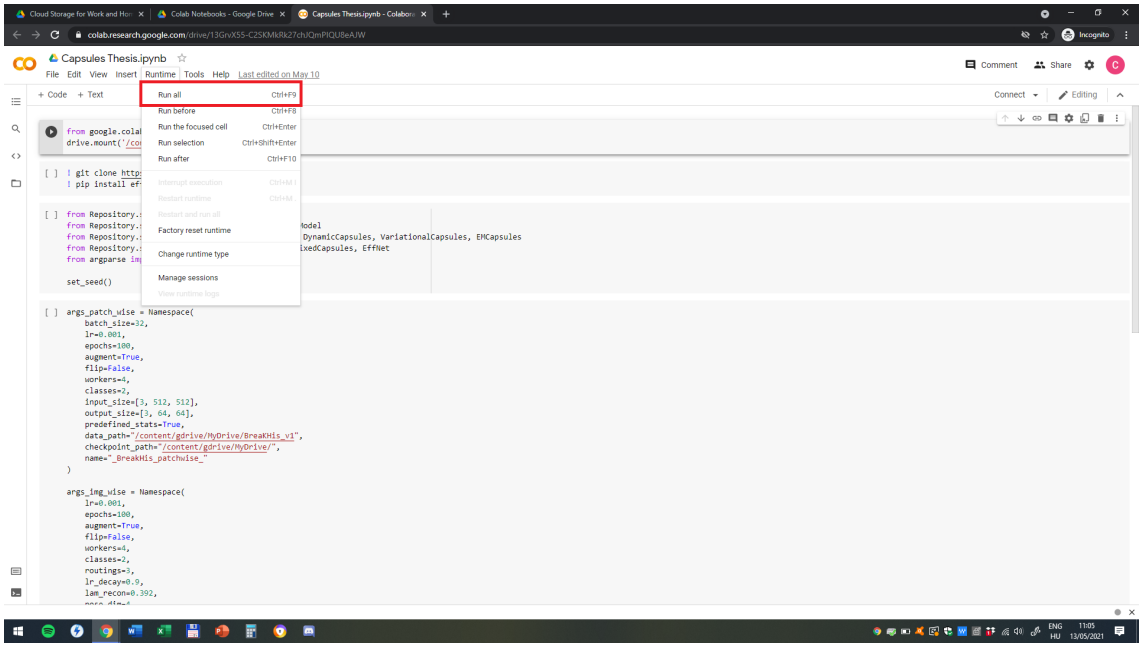
4. Double click on Capsules Thesis.ipynb.



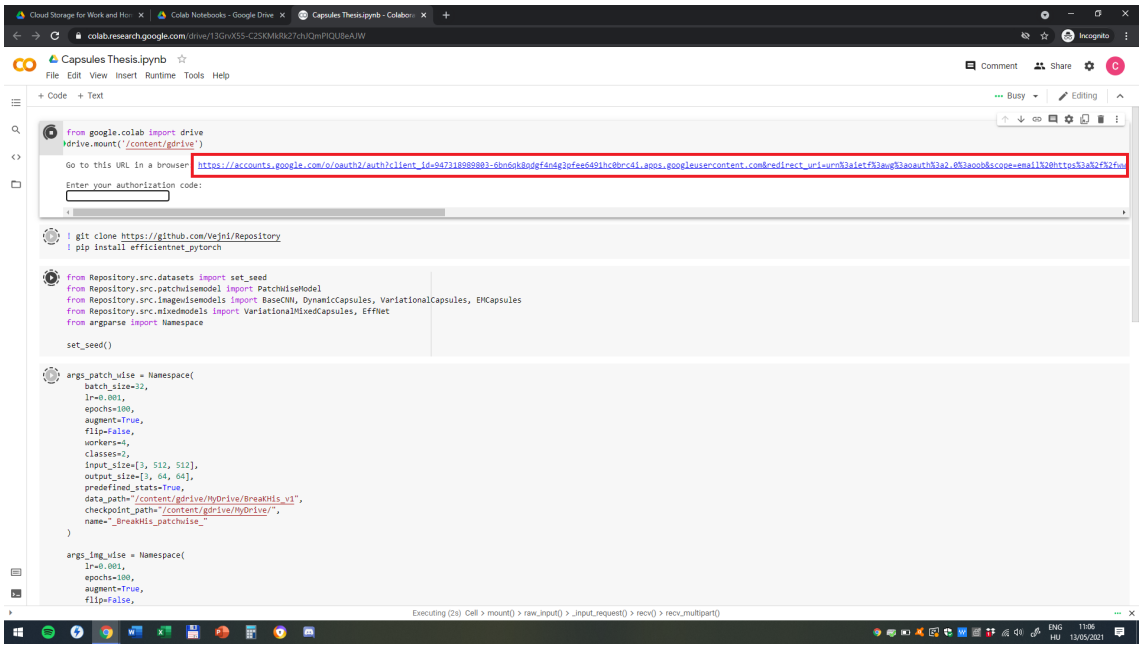
5. (Optional) On Google Colab, check Hardware acceleration is set to GPU. Navigate to "Runtime" > "Change runtime type" and make sure the "Hardware accelerator" is set to GPU.



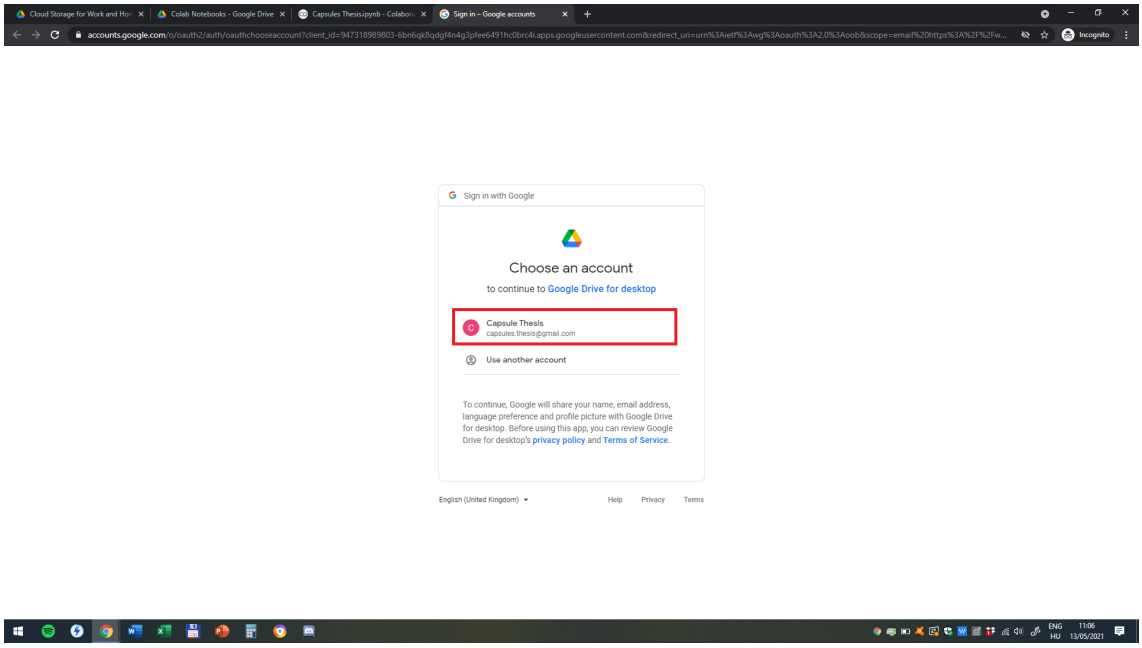
6. Open the "Runtime" menu and select "Run All".



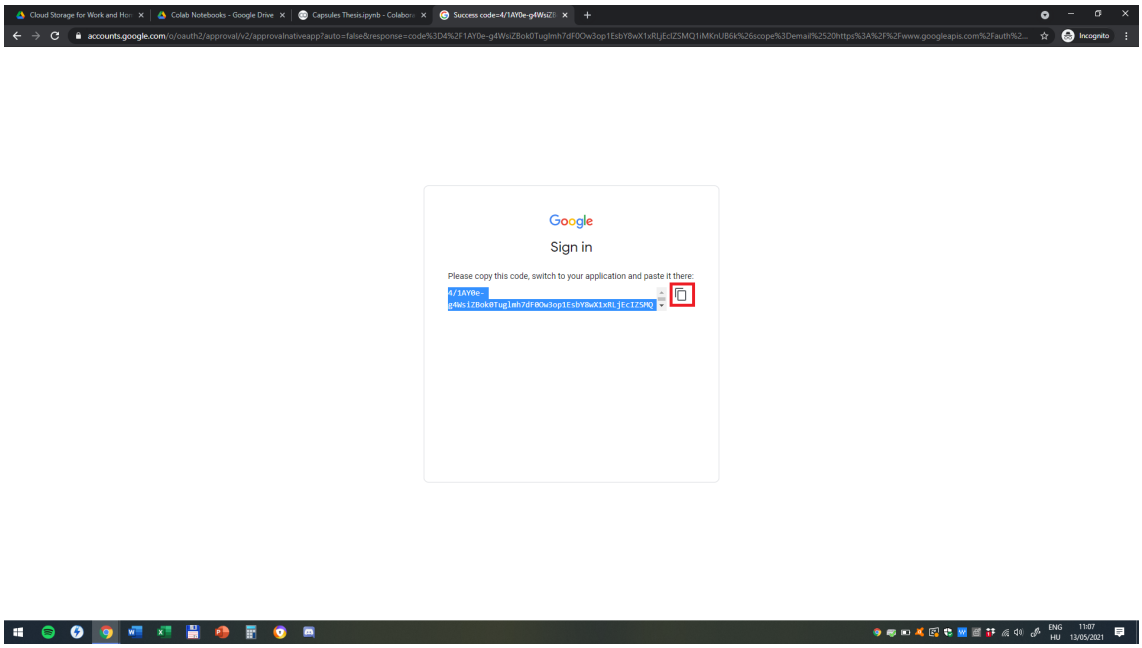
7. When prompted in the output of the first cell, click on the link.



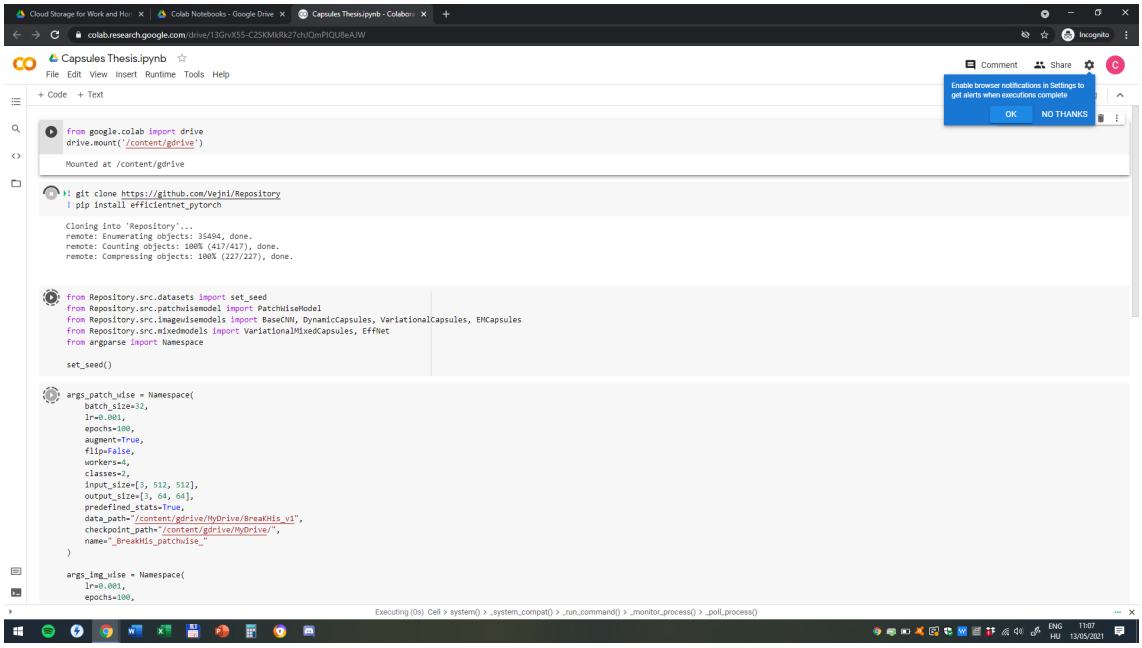
8. Log in with the account provided in step 2.



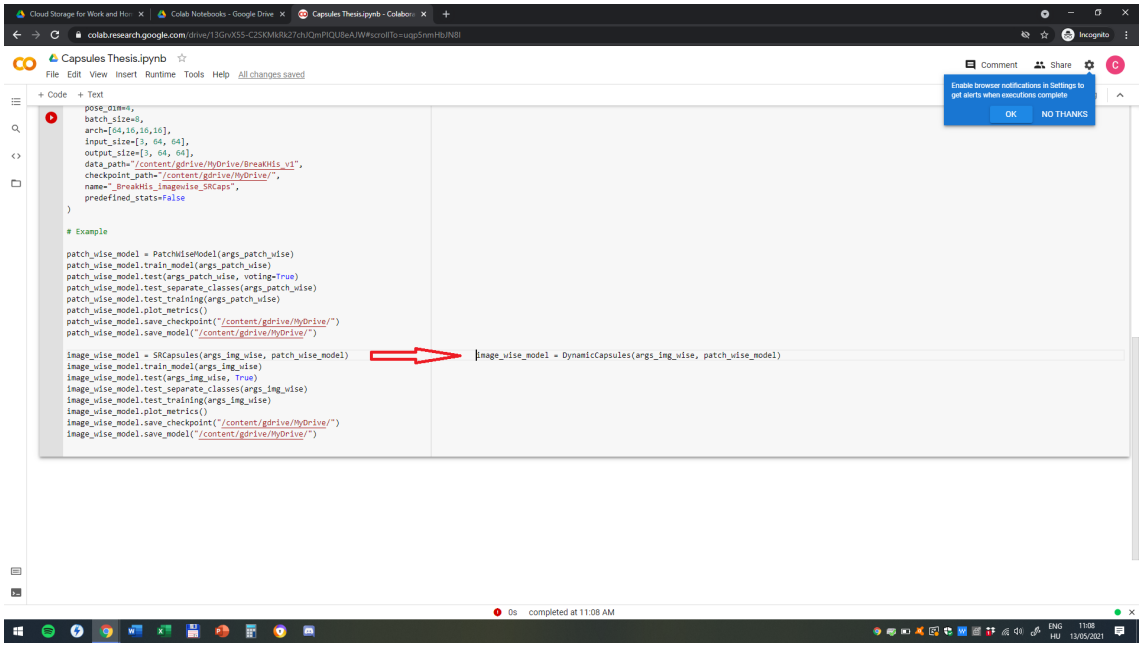
9. Select allow, copy the authentication code and paste it in the input field on Google Colab.



10. The notebook will train a patch-wise network, and a self-routing capsules on BreakHis as a default. This should take a while.



11. (Optional) To train a different image-wise network, simply create a different object, all available image-wise networks are imported in cell 3.



Important note: Variational capsules have a serious memory bottleneck, and had some issues running on Google Colab. In this case lowering the batch size, and getting rid of the data augmentation (set augment to false) can help.

7.2 Maintenance Manual

To set up the project on a home computer, it is easiest to clone the repository git clone `https://github.com/Vejni/Repository`, and download the datasets from Google Drive, logged in

as the user described in the User Manual, Step 2. Alternatively one can download the original datasets and recreate the split and patches using the *datasets.py* script. All project dependencies are in the *requirements.txt* and can be installed via pip.

The main folder contains project files, such as the submit file *task.sh* for Maxwell, *.gitignore, README.md* for git and the launch file *main.py*. The *data* folder contains the datasets (not uploaded to git). The *docs* contain files for the project report, while the *models* folder contains saved models and their checkpoints during training. The *src* folder contains the source code for the project. The overall structure is the following:

- Root: The root folder contains project speficic files, but also the main script which can be run locally.
- src: Contains the source code, such as definitions for the patch- and image-wise networks, training loops and methods for dataset manipulation. Additionally, it contains truncated code for the 3 types of capsules used in the project: DynamicCaps, VarCaps and SRCaps, which can be found in the appropriate folders.
- models: This folder contains saved models, as well as saved checkpoints. Additionally the outs folder contains outputs from experiments, which were then used to create plots. These files are NOT unaltered, as some have been trimmed or put together for ease of plotting.
- docs: Contains the presentation, report and poster files.
- data: Folder for datasets. It is recommended to create the datasets here using the appropriate script.

Description of the files within, can also be found in Chapter 4, but briefly, the *datasets.py* script contains methods to recreate and test the datasets. The *model.py* script defines the basics for all networks used in the project, and patch-wise, image-wise networks and mixed models inherit from the base model. All of these classes can be found in the appropriately named scripts. The folders *DynamicCaps, SRCaps, VarCaps* contain source code for the capsule networks used in this project. They original sources are listed in previous sections as well as within the script. The files within the folders have been cut to the necessities for our project.

To test the networks on a new dataset, it is recommended to download the dataset into the *data* folder, then using the *datasets.py* script to recreate the patches and the test-train-validation split. It is recommended to select the correct resizing, and patching strategy, which should depend on the dataset itself. Running the script will create two folders, one for patch-wise and one for image-wise training. The only things left are to point the models to the new folder, and specify the correct number of classes, in the arguments.

Bibliography

[1] Welcome to pytorch tutorials. *PyTorch*. https://pytorch.org/tutorials/.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Md Zahangir Alom, Chris Yakopcic, Mst Shamima Nasrin, Tarek M Taha, and Vijayan K Asari. Breast cancer classification from histopathological images with inception recurrent residual convolutional neural network. *Journal of digital imaging*, 32(4):605–617, 2019.

[4] Laith Alzubaidi, Muthana Al-Amidie, Ahmed Al-Asadi, Amjad J Humaidi, Omran Al-Shamma, Mohammed A Fadhel, Jinglan Zhang, J Santamaría, and Ye Duan. Novel transfer learning approach for medical imaging with limited labeled data. *Cancers*, 13(7):1590, 2021.

[5] MA Anupama, V Sowmya, and KP Soman. Breast cancer classification using capsule network with preprocessed histology images. In *2019 International Conference on Communication and Signal Processing (ICCSP)*, pages 0143–0147. IEEE, 2019.

[6] Teresa Araújo, Guilherme Aresta, Eduardo Castro, José Rouco, Paulo Aguiar, Catarina Eloy, António Polónia, and Aurélio Campilho. Classification of breast cancer histology images using convolutional neural networks. *PloS one*, 12(6):e0177544, 2017.

[7] Guilherme Aresta, Teresa Araújo, Scotty Kwok, Sai Saketh Chennamsetty, Mohammed Safwan, Varghese Alex, Bahram Marami, Marcel Prastawa, Monica Chan, Michael Donovan, et al. Bach: Grand challenge on breast cancer histology images. *Medical image analysis*, 56:122–139, 2019.

[8] Dalal Bardou, Kun Zhang, and Sayed Mohammad Ahmad. Classification of breast cancer based on histology images using convolutional neural networks. *Ieee Access*, 6:24680–24693, 2018.

[9] Hamidreza Bolhasani, Elham Amjadi, Maryam Tabatabaeian, and Somayyeh Jafarali Jassbi. A histopathological image dataset for grading breast invasive ductal carcinomas. *Informatics in Medicine Unlocked*, 19:100341, 2020.

[10] Said Boumaraf, Xiabi Liu, Zhongshu Zheng, Xiaohong Ma, and Chokri Ferkous. A new transfer learning based approach to magnification dependent and independent classification

of breast cancer in histopathological images. *Biomedical Signal Processing and Control*, 63:102192, 2021.

[11] Charlotte Burmeister. Capsule networks - better cnns?, Jan 2020.

[12] Grand Challange. Iciar 2018 - breast histology cancer images. `https://iciar2018-challenge.grand-challenge.org/Dataset/`. Accessed: 07-04-2021.

[13] Francois Chollet et al. Keras, 2015. https://github.com/fchollet/keras.

[14] Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural computation*, 22(12):3207–3220, 2010.

[15] Dan Claudiu Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Twenty-second international joint conference on artificial intelligence*, 2011.

[16] coder3000. Sr-capsnet. *GitHub*. https://github.com/coder3000/SR-CapsNet.

[17] Sumaiya Dabeer, Maha Mohammed Khan, and Saiful Islam. Cancer diagnosis in histopathological image: Cnn based approach. *Informatics in Medicine Unlocked*, 16:100231, 2019.

[18] fabio deep. Variational-capsule-routing. *GitHub*. https://github.com/fabio-deep/Variational-Capsule-Routing.

[19] Kunihiko Fukushima, Sei Miyake, and Takayuki Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (5):826–834, 1983.

[20] Farzad Ghaznavi, Andrew Evans, Anant Madabhushi, and Michael Feldman. Digital imaging in pathology: whole-slide imaging and beyond. *Annual Review of Pathology: Mechanisms of Disease*, 8:331–359, 2013.

[21] GitHub. Apex pytorch cifar-10 experiments. `https://github.com/ceshine/apex_pytorch_cifar_experiment`. Accessed: 08-04-2021.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[23] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[24] Ashley Daniel Gritzman. Avoiding implementation pitfalls of "matrix capsules with em routing" by hinton et al. In *International Workshop on Human Brain and Artificial Intelligence*, pages 224–234. Springer, 2019.

[25] Taeyoung Hahn, Myeongjang Pyeon, and Gunhee Kim. Self-routing capsule networks. 2019.

[26] Geoffrey Hinton, Alex Krizhevsky, Navdeep Jaitly, Tijmen Tieleman, and Yichuan Tang. Does the brain do inverse graphics. In *Brain and Cognitive Sciences Fall Colloquium*, volume 2, 2012.

[27] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[28] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with em routing. In *International conference on learning representations*, 2018.

[29] Tomas Iesmantas and Robertas Alzbutas. Convolutional capsule network for classification

of breast cancer histology images. In *International Conference Image Analysis and Recognition*, pages 853–860. Springer, 2018.

[30] Amelia Jiménez-Sánchez, Shadi Albarqouni, and Diana Mateus. Capsule networks against medical imaging data challenges. In *Intravascular Imaging and Computer Assisted Stenting and Large-Scale Annotation of Biomedical Data and Expert Label Synthesis*, pages 150–160. Springer, 2018.

[31] Ray Johns. Pytorch vs tensorflow for your python deep learning project. *Real Python*. https://realpython.com/pytorch-vs-tensorflow/.

[32] Amirhossein Kazemnejad. How to do deep learning research with absolutely no gpus - part 2, Aug 2019.

[33] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8):5455–5516, 2020.

[34] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[35] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.

[36] Adam R Kosiorek, Sara Sabour, Yee Whye Teh, and Geoffrey E Hinton. Stacked capsule autoencoders. *arXiv preprint arXiv:1906.06818*, 2019.

[37] Marek Kowal, Paweł Filipczuk, Andrzej Obuchowicz, Józef Korbicz, and Roman Monczak. Computer-aided diagnosis of breast cancer based on fine needle biopsy microscopic images. *Computers in biology and medicine*, 43(10):1563–1572, 2013.

[38] KR Kruthika, HD Maheshappa, Alzheimer's Disease Neuroimaging Initiative, et al. Cbir system using capsule networks and 3d cnn for alzheimer's disease diagnosis. *Informatics in Medicine Unlocked*, 14:59–68, 2019.

[39] Sebastian E. Kwiatkowski. Awesome capsule networks.

[40] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[41] lukemelas. Efficientnet-pytorch. *GitHub*. https://github.com/lukemelas/EfficientNet-PyTorch.

[42] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[43] Seonwoo Min, Byunghan Lee, and Sungroh Yoon. Deep learning in bioinformatics. *Briefings in bioinformatics*, 18(5):851–869, 2017.

[44] minqz2009. #9: patch-wise model could only reach 68%. *GitHub*. https://github.com/ImagingLab/ICIAR2018/issues/9.

[45] Kamyar Nazeri, Azad Aminpour, and Mehran Ebrahimi. Two-stage convolutional neural

network for breast cancer histology image classification. In *International Conference Image Analysis and Recognition*, pages 717–726. Springer, 2018.

[46] NHS. Breast cancer in women. `https://www.nhs.uk/conditions/breast-cancer/`. Accessed: 18-03-2021.

[47] Kyoung-Su Oh and Keechul Jung. Gpu implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, 2004.

[48] Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep learning vs. traditional computer vision. In *Science and Information Conference*, pages 128–144. Springer, 2019.

[49] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[50] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[51] Max Pechyonkin. Understanding hinton's capsule networks. part i: Intuition.

[52] Kenneth A Philbrick, Kotaro Yoshida, Dai Inoue, Zeynettin Akkus, Timothy L Kline, Alexander D Weston, Panagiotis Korfiatis, Naoki Takahashi, and Bradley J Erickson. What does deep learning see? insights from a classifier trained to predict contrast enhancement phase from ct images. *American Journal of Roentgenology*, 211(6):1184–1193, 2018.

[53] OCF plc. Ocf user manual for university of aberdeen maxwell hpc cluster. `https://www.abdn.ac.uk/it/documents-uni-only/OCF-User0-Manual-Abderdeen-Maxwell.pdf`. Accessed: 08-04-2021.

[54] Venubabu Rachapudi and G Lavanya Devi. Improved convolutional neural network based histopathological image classification. *Evolutionary Intelligence*, pages 1–7, 2020.

[55] Fabio De Sousa Ribeiro, Georgios Leontidis, and Stefanos Kollias. Capsule routing via variational bayes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3749–3756, 2020.

[56] Fabio De Sousa Ribeiro, Georgios Leontidis, and Stefanos Kollias. Introducing routing uncertainty in capsule networks. *Advances in Neural Information Processing Systems*, 33, 2020.

[57] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. *arXiv preprint arXiv:1710.09829*, 2017.

[58] Sara Sabour, Andrea Tagliasacchi, Soroosh Yazdani, Geoffrey E Hinton, and David J Fleet. Unsupervised part representation by flow capsules. *arXiv preprint arXiv:2011.13920*, 2020.

[59] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. Benchmarking state-of-the-art deep learning software tools. In *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, pages 99–104. IEEE, 2016.

[60] Rebecca L Siegel, Kimberly D Miller, and Ahmedin Jemal. Cancer statistics, 2016. *CA: a cancer journal for clinicians*, 66(1):7–30, 2016.

[61] Slurm. Workload manager. `https://slurm.schedmd.com/documentation.html`. Accessed: 08-04-2021.

[62] Robert A Smith, Vilma Cokkinides, and Harmon J Eyre. American cancer society guidelines for the early detection of cancer, 2004. *CA: a cancer journal for clinicians*, 54(1):41–52, 2004.

[63] Fabio A Spanhol, Luiz S Oliveira, Caroline Petitjean, and Laurent Heutte. A dataset for breast cancer histopathological image classification. *Ieee transactions on biomedical engineering*, 63(7):1455–1462, 2015.

[64] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[65] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[66] Cancer Research UK. Breast cancer mortality statistics. Accessed: 18-03-2021.

[67] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[68] Yu Emma Wang, Gu-Yeon Wei, and David Brooks. Benchmarking tpu, gpu, and cpu platforms for deep learning. *arXiv preprint arXiv:1907.10701*, 2019.

[69] XifengGuo. Capsnet-pytorch. *GitHub*. https://github.com/XifengGuo/CapsNet-Pytorch.

[70] Min Yang, Wei Zhao, Lei Chen, Qiang Qu, Zhou Zhao, and Ying Shen. Investigating the transferring capability of capsule networks for text classification. *Neural Networks*, 118:247–261, 2019.

[71] Wei Zhao, Haiyun Peng, Steffen Eger, Erik Cambria, and Min Yang. Towards scalable and reliable capsule networks for challenging nlp applications. *arXiv preprint arXiv:1906.02829*, 2019.

[72] Yushan Zheng, Zhiguo Jiang, Haopeng Zhang, Fengying Xie, Dingyi Hu, Shujiao Sun, Jun Shi, and Chenghai Xue. Stain standardization capsule for application-driven histopathological image normalization. *IEEE journal of biomedical and health informatics*, 2020.

[73] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.