Final Report

Comparative Analysis of Large Pretrained and Custom Lightweight CNN models for Breast Cancer Detection in Histopathology Images

Hsin Jee Sum

Submitted in accordance with the requirements for the degree of BSc Computer Science

2024/25

COMP3931 Individual Project

Abstract. (Summary)

BACKGROUND: Breast cancer is the leading cause of cancer deaths in women worldwide. In 2022, Breast cancer accounted for over 2.3 million new cases and contributed to over 666,000 deaths globally. Early and accurate diagnosis is crucial for increasing the survival rate of patients.

Histopathology remains the clinical gold standard: thin tissue sections are stained to highlight the various cellular features. This process reveals key diagnostic features—nuclear size and shape, texture, and cell boundaries in high resolution. However, traditional histopathological slide reviews are labor-intensive, requiring specialist training, and are subject to inter- and intra-observer variability, leading to inconsistent classification and longer diagnostic turnaround times. Automating this process with deep learning, particularly Convolutional Neural Networks (CNNs), which excel at hierarchically extracting color, texture, and shape patterns, can reduce costs, speed up reporting, and improve reproducibility.

Many recent studies have shown significant gains in medical imaging through pretrained CNN architectures such as ResNet50 and VGG16. These models leverage batch normalization and adaptive optimizers like Adam to achieve rapid convergence and high accuracy. However, these large pre-trained models come with substantial computational memory demands due to their tens of millions of parameters. To address these limitations, CNNs implemented from scratch offer a lightweight alternative. Yet, most implementations rely on single-image training with vanilla SGD as an optimizer, which is shown to lead to unstable gradients and slow convergence.

OBJECTIVE: To develop and train a lightweight CNN-from-scratch on the BreakHis 200x dataset that matches the classification performance of pretrained models, with a focus on achieving >90% malignant recall and >85% F1-score. The model should also be at least 80% smaller in size, have faster inference, and ideally train more quickly than the pretrained counterparts.

METHODS: Implement a CNN from scratch in NumPy, incorporating batched inputs and batch normalization. Adam optimization is used to improve convergence rates. Train and fine-tune the NumPy CNN on the BreakHis 200x histopathology dataset, optimizing hyperparameters to maximize classification accuracy, precision, recall, and F1-score. The CNN's performance will then be benchmarked against pretrained ResNet50 and VGG16 models under identical conditions (same train/validation/test set splits and augmentation pipeline) while using model-appropriate input preprocessing for each network.

RESULTS: An overall accuracy of 84.16% was achieved for the custom CNN, along with 90.65% malignant recall, and an F1-score of 88.73%, meeting the project goals. It performed competitively against the large pretrained models while being 40x smaller and faster in inference. Although the custom CNN was significantly slower when training.

CONCLUSION: Although the primary goals were met, the custom CNN still underperformed slightly against ResNet50 and VGG16 in classification ability. The training speed of the CNN is significantly slower than the pretrained model due to using NumPy and being restricted only to CPU training. However, the model’s lightweight nature, faster inference time, and competitive results show strong potential. With optimization through Keras and GPU acceleration, the custom CNN could achieve comparable accuracy to pretrained models, without the limitation of long training times.

Contents

Chapter 1
Introduction and Background Research

1.1 Introduction

Breast cancer remains one of the most common life-threatening diseases affecting women worldwide [1]. In the UK, breast cancer is the most common cancer, with approximately 1 in 7 women developing it in their lifetime. Approximately 55,000 women and 400 men are diagnosed annually.

Globally, in 2022, there were 2.3 million breast cancer diagnoses and over 670,000 deaths according to the World Health Organization (WHO). Although breast cancer's exact causes are complex and not fully understood, certain factors may increase the risk of developing the disease. These include genetics, age, lifestyle choices, and hormonal factors. Several non-genetic factors, such as lifestyle choices like obesity and smoking, expose the body to numerous carcinogens. These carcinogens damage DNA, leading to abnormal growth, which may lead to the development of malignant (cancerous) cells.

Between 2017 and 2019, there were an average of 56,822 new cases of breast cancer diagnosed each year in the UK, according to Cancer Research UK [27], it was also estimated that approximately 23% of all UK breast cancer cases were preventable, highlighting the critical importance of early and accurate diagnosis for effective treatment and improved survival outcomes. According to the American Cancer Society, invasive ductal carcinoma and lobular carcinoma together account for 70-80% of all breast cancer cases. While the detection of a lump of mass can be alarming, studies have shown that around 80% of breast lumps biopsied are benign and non-cancerous [2]. Common examples of benign breast tumors include fibroadenomas and tubular adenomas. Nevertheless, the American Cancer Society also states that distinguishing early-stage breast cancer diagnosis significantly improves survival rates, with five-year survival rates exceeding 95% when detected at Stage 1, compared to less than 30% at Stage 4. Despite advances in screening, traditional mammography only has a sensitivity of approximately 75-85%.

A histopathological image is a microscopic view of biological tissue that has been stained using chemical dyes to highlight different structural components, such as nuclei, cytoplasm, and extracellular matrix [3]. Common staining techniques, such as Hematoxylin and Eosin (H&E) staining, are used to enhance visual contrast between tissue structures, allowing important features to be distinguished during microscopic examination. Feature extraction in this field typically involves identifying patterns such as cell shape, tissue texture, size, and shape of the nucleus, and arrangement of cells, all of which can be indicators of malignancy (cancer).

Histopathological image analysis, a standard method of identifying breast cancer, traditionally relies on the expertise of pathologists, which may not be readily available. This is a very time-consuming, subjective process and is prone to human error.

Often in research studies, there are disagreements over the classification of the tumor, particularly in complex cases. Elmore et al. (2015) found that pathologists disagreed in roughly 25% of all cancer diagnosis cases. For atypia (a pre-cancerous condition), misclassification rates were up to 52%, for ductal carcinoma in situ (a non-invasive cancer), disagreement was around 17%. Invasive breast cancer was no exception; although the diagnostic agreement was the highest overall, it was still far from perfect [28]. Their final consensus was that there was a strong need for better systems (like AI support) to assist radiologists in mammogram interpretation to improve cancer detection while reducing false positives.

In recent years, Convolutional Neural Networks (CNNs) have demonstrated remarkable success in various computer vision tasks, including disease classification and detection. Their ability to automatically learn hierarchical features makes them well-suited for analyzing complex medical images. In histopathological image analysis, CNNs can effectively capture both low-level features, such as texture and color variations, and high-level structures, such as cellular patterns and tissue architecture [4]. Unlike traditional methods that rely on manual feature extraction, CNNs can discover relevant features during the training process, reducing human bias and improving diagnostic consistency. This capability is especially valuable in histopathology, where distinguishing between benign and malignant tissues often depends on subtle differences that may be challenging to detect with the naked-untrained eye.

1.2 Literature Review (Related Work)

1.2.1 Overview of key studies

Spanhol FA, Oliveira LS, Petitjean C, Heutte L introduced the BreakHis dataset, which consists of 7909 breast tumor images of benign and malignant types at various magnifications (40x, 100x, 200x, 400x) [5], to address the lack of publicly available histopathological images for breast cancer research. Their work laid the foundation for many subsequent CNN-based studies. Several studies have reported success using CNNs on this dataset, achieving high classification accuracy and demonstrating CNN’s potential to support pathologists in diagnosis [4][6].

Among these, Araújo et al. (2017) [7] proposed a deep learning framework using CNNs to classify hematoxylin and eosin-stained (H&E-stained) breast biopsy images into four categories-normal, benign, in situ carcinoma, and invasive carcinoma well as a binary carcinoma vs. non-carcinoma classification. Their architecture integrated multi-scale information from nuclei-level and tissue-level features and demonstrated adaptability for whole-slide images.

Lakshimi et al. (2024) [4] evaluated various deep learning approaches, particularly CNNs, for detecting and classifying breast cancer in histopathological images. Through transfer learning, they used pre-trained large networks, such as VGGNet, ResNet, and InceptionNet, to improve performance on limited datasets. Throughout the paper, the authors discussed challenges such as data scarcity, where the limited number of annotated histopathology images hinders the effective training of deep CNNs, leading to overfitting, which remains a significant risk when models are trained on relatively small datasets like BreakHis.

In their 2018 study, Rakhlin et al. directly compared VGG and ResNet models on the BACH 2018 dataset, which consists of hematoxylin and eosin-stained breast cancer histology images. They reported an accuracy of up to 87.2% in four-class classification using ResNet-34, with ResNet-34 outperforming both VGG-16 and VGG-19 in most experiments [29]. In addition, they achieved up to 93% accuracy in binary classification tasks, distinguishing carcinoma from non-carcinoma cases. Their results demonstrated that transfer learning from ImageNet-pretrained models significantly improved classification accuracy compared to training CNNs from scratch. However, the authors also acknowledged that due to the small size of the BACH dataset, the generalizability of their model was restricted.

Alessandro Saviolo [22] created a CNN architecture from scratch to classify MNIST and CIFAR-10 datasets. His implementation consisted of processing only one image at a time and using the Stochastic Gradient Descent (SGD) optimizer for gradient calculations and weight updates during backpropagation, with cross-entropy as the loss function. His architecture for MNIST consisted of two convolutional layers and a dense output layer, and achieved a test accuracy of 87.2%, which improved to 89.6% after applying L2 regularization. However, he noted that due to the nature of NumPy-based implementations, his models were restricted to only training on the CPU, which was extremely slow, especially for the more complex CIFAR-10 dataset, which took 13 hours to train and achieve reasonable results and convergence.

1.2.2 Themes, Trends, and Gaps

Recent literature has revealed several recurring themes in breast cancer histopathological image analysis. One major theme is growing reliance on deep learning methods, particularly Convolutional Neural Networks (CNNs), for tumor classification. CNNs have demonstrated strong capabilities in learning spatial hierarchies directly from image data without the need for handcrafted features [9]. They have been particularly effective in identifying subtle patterns within tissue samples that are difficult to detect through traditional methods.

A clear trend is the widespread use of transfer learning. Pretrained large CNN models such as ResNet, VGG, and Inception have been extensively fine-tuned on medical imaging datasets to enhance performance, especially when labeled data is limited [10]. This approach has become standard practice, as it enables researchers to leverage powerful feature extractors originally trained on large-scale datasets like ImageNet [8], thereby improving generalization and reducing overfitting.

Another trend is the growing focus on model explainability. Techniques such as Gradient-weighted Class activation Mapping (Grad-CAM) and saliency maps are used to visualize which regions of an image contribute most to a CNN’s decision. These methods enhance model transparency and trust, making CNNs more acceptable for deployment in clinical settings where interpretability is crucial [11].

However, while recent CNN architectures, such as ResNet and VGG, have achieved significant gains in diagnostic accuracy, most “CNN-from-scratch” implementations built in NumPy train on only one image at a time and employ basic stochastic gradient descent (SGD) optimizers. This results in noisy gradient updates, slow convergence, and inefficient use of computational resources, as modern hardware is optimized for processing mini-batches rather than individual images. In Goodfellow, Bengio, and Courville’s 2016 book “Deep Learning” [30], the authors emphasize that

mini-batch processing not only accelerates through hardware and parallelism but also stabilizes the optimization process by reducing gradient variance, as you are taking the gradients from multiple images into consideration when training instead of just one image.

1.2.3 Addressing the Gaps

Recent studies often employ large pretrained CNNs such as ResNet50 and VGG16 to improve classification performance, but these models come with significant drawbacks. Their high computational and memory requirements pose challenges for training and deployment on resource-limited hardware. When applied to small medical datasets, they are prone to overfitting due to their large parameter count. In addition, because these models are pretrained on natural image datasets like ImageNet, there may be a domain mismatch that reduces their effectiveness in extracting relevant features from histopathological images.

These limitations highlight the need for a new type of CNN optimized for training efficiency, that is not only lightweight, generalizes well on medical datasets, but also achieves performance comparable to large pretrained CNN models without the need for millions of parameters.

For a reference benchmark, two of the most widely used and popular CNNs will be fine-tuned until they reach a validation accuracy of >90%, the average accuracy of previous studies (see Section 1.2.1). The number of parameters and training time will be reported to provide a definitive view of whether our model is truly more efficient.

The main objective of this study is to design and evaluate a CNN architecture that's specifically built for classifying histopathological images. Unlike large pretrained models that rely on datasets like ImageNet, which aren't directly related to medical imaging, this model focuses entirely on the domain at hand. It's designed to be lightweight, efficient, and easier to run on limited hardware, while still achieving strong performance. The target is an overall accuracy of above 80%, and more importantly, a malignant recall greater than 90%. This high recall is crucial because missing a malignant case can lead to serious, potentially life-threatening consequences. In medical settings, it's better to catch more suspicious cases, even if a few turn out to be false positives; it is better to be safe than sorry.

This project doesn't aim to beat complex models like ResNet50 or VGG16 in raw performance (although such an outcome would also be desirable), but rather to provide a practical, compact alternative that performs well enough to be used by most people. Pretrained models often contain tens of millions of parameters and are built for general-purpose tasks, while the custom model is trained from scratch and tailored to the specific dataset. One of the goals is to make predictions 50% faster than those of larger models, showcasing its efficiency.

Training time is also an important consideration. Since the custom CNN has fewer layers and parameters, it should ideally train faster as well. The aim is for training to be at least 10% quicker than the pretrained models.

1.3 Stakeholder Analysis

The primary stakeholders in this project include medical researchers, pathologists, and technical developers aiming to deploy CNN-based diagnostic tools in resource-limited settings. As the sole developer of this project, I am also a stakeholder.

To ensure this project's success, a set of functional and non-functional requirements was established, alongside a risk assessment to identify potential obstacles. These requirements were formed in line with the project's objective of building a lightweight, efficient alternative to the heavy pre-trained models, with comparable results.

Stake Holder	Requirements
Pathologist	High recall for malignant class (>90%) to avoid missed cancers
Pathologist	Short inference time (<0.5s) for quick feedback
Hospital (low resource)	The model should be small (<3MB) and not require internet or cloud
Pathologist	High overall classification accuracy (>90%)
Technical developers	Clear, modular code with reproducible results
Technical developer	Full use of version control to track model changes and experimentation
Pathologist	Visual explanation of predictions (e.g., feature maps or Grad-Cam) to increase trust and interpretability

Table. 1.1. Stakeholder Analysis Table

Risk	Stake Holder	Impact
False negatives	Patients, Pathologist	Missed cancer, leading to delayed treatment, may be life-threatening
False positives	Patients, Pathologist	Unnecessary emotional stress, biopsies, and cost
Lack of interpretability	Pathologist	Loss of trust, leading to underutilization
Slow inference	Pathologist	Longer diagnostic turnaround times or frustration

Table. 1.2. Stakeholder Risk Analysis Table

Chapter 2
Methodology

2.1 Coding Practices, System Architecture, and Goals

All source code was tracked in a Git repository hosted on GitHub [0]. The multiple versions of the implementation can also be accessed through the different branches. For this project, I adopted a feature-branch workflow: each new feature or bug-fix developed its own branch and then merged into main after review. Each commit message has a meaningful name and description, ensuring a clear history of changes.

The codebase is organized into clearly named packages and modules (as shown below), each with a single responsibility.

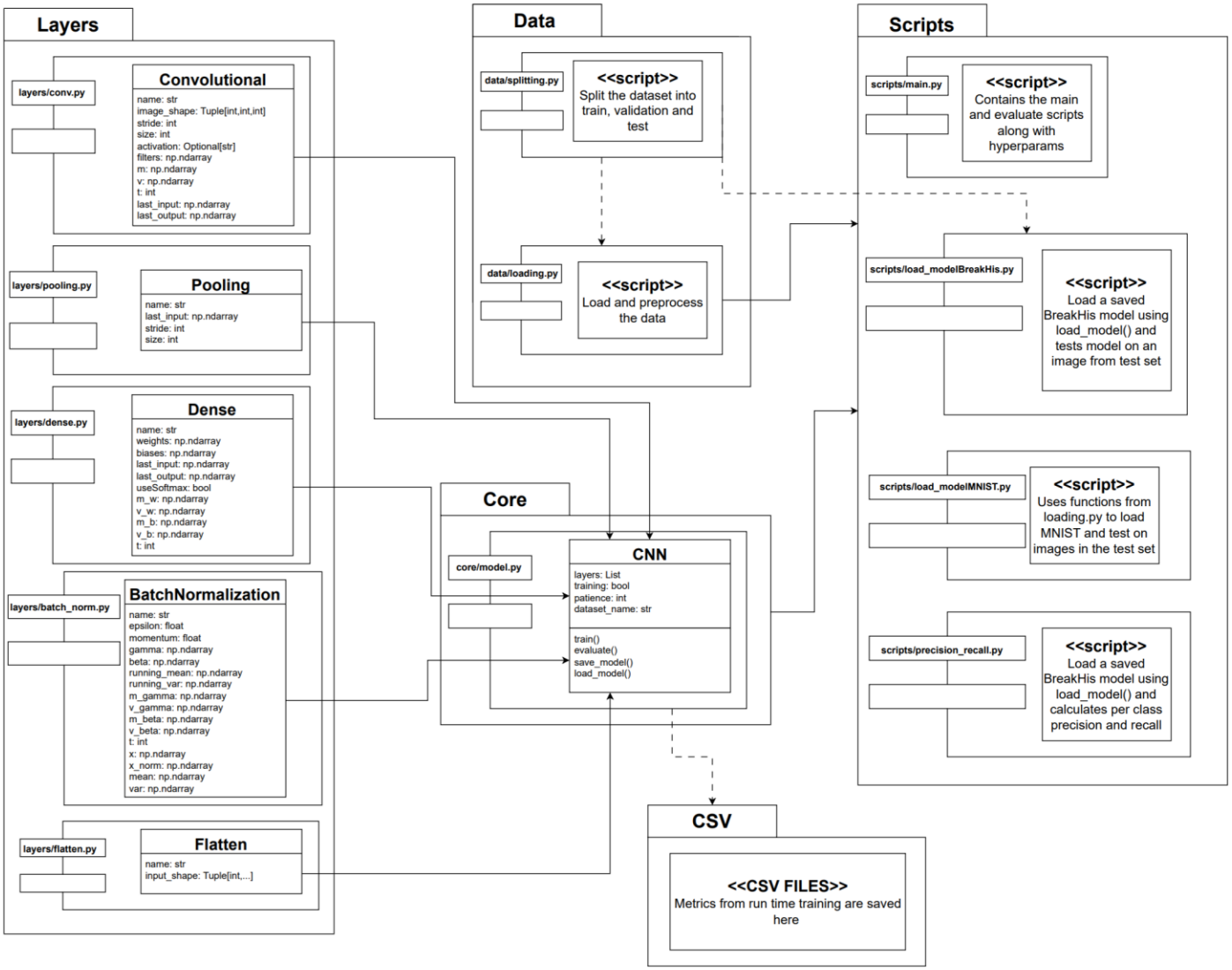
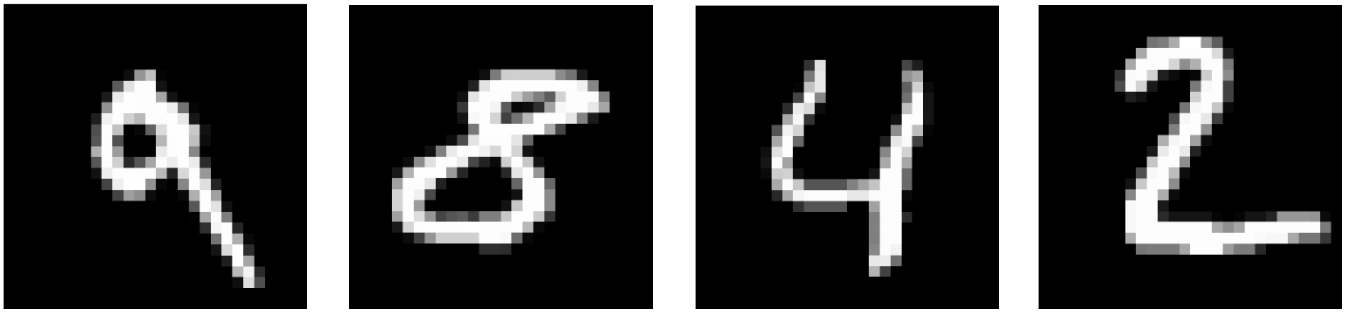


Fig. 2.1. High-level component (package) diagram of the CNN-from-scratch codebase, showing the four main packages—Layers, Core, Data and Scripts and their dependency relationships, along with CSV saving folder

2.2 MNIST

2.2.1 Data Preprocessing

When first building the CNN model from scratch, the initial tests were conducted using the MNIST dataset – a benchmark dataset consisting of 70,000 grayscale images of handwritten digits (0-9), each sized at 28x28 pixels. This dataset was first introduced in 1998 by LeCun et al. [15].



(a) (b) (c) (d)

Fig. 2.2. Samples of images acquired from the MNIST data set

Each input image in a CNN is typically represented as a vector in the form of (height, width, channels), a standard input format across all modern CNN architectures. The MNIST dataset was chosen for the initial model testing due to its simplicity and small size. MNIST images have dimensions of 28x28 pixels with a single channel (28, 28, 1) for grayscale (black and white).

In contrast, the BreakHis dataset contains full-color, high-resolution scans. To feed BreakHis into our model, we resized to 224x224 and kept all RGB channels (224, 224, 3) (see Section 2.3), which increases the input complexity, requiring a more complex layer architecture to accurately tell if our model is learning or not. Not ideal for early-stage model development.

Furthermore, MNIST has consistently demonstrated strong performance in CNN-based studies. For example, in a recent 2025 study, Cheraja hit a 99.26% accuracy on MNIST with a CNN in PyTorch [16].

MNIST images are 28x28 grayscale arrays of pixel values in the integer range 0-255. Feeding raw images to the CNN will lead to slow convergence and unstable gradients, so we first rescale each pixel to a continuous range [0, 1] using min-max normalization [26]. No data augmentation was applied.

$$[x' = \frac{x - x_{min}}{x_{max} - x_{min}}, \quad \text{where } x_{min} = 0,\ x_{max} = 255.]$$

Equation. 2.1 Min-Max equation used for preprocessing images

2.2.2 Model Architecture

The architecture of my custom CNN model draws inspiration from the designs proposed by Alessandro Saviolo [22] and Alfak, O. [17]. However, our model introduces several key improvements and optimizations.

The custom CNN model processes batches of grayscale images of the MNIST dataset (28x28x1) through a series of convolutional and dense layers. Unlike the previous two models and other CNN implementations from scratch, which processed input images individually, our model adopts batched input processing with a batch size of 32 images in random order from the train dataset. This design choice is due to several factors. First, batching significantly improves computational efficiency, processing batches allows the model to utilize more efficient memory access patterns and reduces the overhead when processing per sample. By averaging the gradient over 32 samples instead of relying on a single sample per update, the training process becomes more stable and less sensitive to noise in the data, leading to faster and more reliable convergence [23]. A batch size of 32 is chosen for this model because Keskar et al. (2017) revealed that large batch sizes tended to converge to sharp minima in the loss landscape, which can negatively affect generalization performance [22].

Convolutional layers utilize 3x3 filters, which is a standard choice to apply the convolutional operation to the input image (by sliding over the first image vertically and then horizontally) [17]. Note that while filter size is set to 3, stride is set to 2 letting us efficiently downsample the image, reducing spatial dimensions and computational costs while maintaining, essential information. In their 2012 study, Bottou and Bosquet found that subsampling can preserve learning performance while significantly improving computational efficiency [18].

Layer	Input shape	Output shape	Filter Number	Stride	Filter/Kernel Size	Activation
Convolution	(32, 28, 28, 1)	(32, 13, 13, 8)	8	2	3	ReLU
Convolution	(32, 13, 13, 8)	(32, 6, 6, 8)	8	2	3	ReLU
Flatten	(32, 6, 6, 8)	(32, 288)	-	-	-	-
Dense (with SoftMax)	(32, 288)	(32, 10)	-	-	-	-

Table. 2.1. CNN architecture for the MNIST dataset

To validate the correctness of the custom CNN's forward and backward passes, an initial simplified architecture was designed and tested on the MNIST dataset. Table 2.1 outlines the specific configuration of this network, highlighting the input and output shapes at each layer. The model layer architecture was deliberately kept shallow for two reasons, first, for faster training and easier debugging and second, due to the complexity of the dataset, even a simple architecture can obtain high accuracy on MNIST (this is the main reason for the initial four-layer architecture). A more complex architecture is used when building for BreakHis (see section 2.3.3).

Each convolution is followed by a ReLU activation function. This activation is chosen because it introduces nonlinearity into the network while being efficient to run. Unlike sigmoid or tanh activations, ReLU does not suffer from

vanishing gradient problems for positive activations [21]. Although it is noted that the ReLU activation function is susceptible to the “dead ReLU problem” [19], where neurons can permanently output zero if their input falls into the negative region, causing their gradients to vanish and preventing further learning. To mitigate this issue, proper weight initialization such as He initialization [20], was employed in the convolutional layer to maintain positive activations in training.

Following the feature extraction, the resulting feature maps are flattened into a one-dimensional vector. This vector is fed into a fully-connected (Dense) layer, where a SoftMax activation function produces a probability distribution over the target classes. For the initial MNIST experiments, this corresponds to 10 classes (digits 0-9) (see section 2.2.1). When applied to the BreakHis binary classification task, the final Dense layer is adjusted to have two output neurons (benign vs. malignant).

The following table shows the hyperparameters used to fine-tune the CNN on the MNIST dataset. The number of iterations is equal to the number of batches of size 32 in the training set, and the number of epochs is set to 1.

Hyper parameter	Value
# Iterations per epoch	1719
Learning rate	0.01
Optimizer	SGD and Adam
Lambda	0
Epochs	1

Table. 2.2. Hyperparameters tuned to train CNN on MNIST. Lambda is the regularization term for L2 regularization.

It is noted that since lambda is set to 0, no regularization is applied. By using the architecture in Table 2.3.1 and hyperparameters in Table 2.3.2, the model requires on average 23s to run 50 batches of size 32 on both optimizers, on an Intel Core i7 12th Gen CPU. With a training set of size 55000, this amounts to 13.1 minutes per epoch and 1719 batches.

2.2.3 Optimizers

When deciding which optimizer was best, two versions of the model were implemented, one using SGD for gradient descent calculations, more aligned with how Alessandro Saviolo [22] and Alfak, O. [17] implemented their CNNs, and the other version using Adam [24]. During training, metrics were logged into a Python dictionary ({'loss': [], 'accuracy': [], 'lr': [], 'val_loss': [], 'val_accuracy': []}). Each model’s weights and parameters were saved into a .pkl file using Python’s pickle library [25] at peak val_accuracy. The dictionary is then exported to a CSV after training for analysis (refer to Figure 2.1).

Optimizer	Highest Train Acc	Highest Val Acc
SGD	96.6%	78.3%
ADAM	90%	91.5%

Table. 2.3. Results of both models using hyperparameters from Table 2.2.

By using the architecture described in Table 2.1 and the hyperparameters from Table 2.2, the results are listed above. Their respective training accuracies and learning curves produced by the training process are illustrated.

Figure. 2.3 Results of both models on the MNIST dataset using hyperparameters from Table 2.2. To achieve a smoother, more interpretable curve, each accuracy point on the graph represents the average of the last 100 batches.

Other than to achieve a smoother, more interpretable curve, the averaging was done to reduce volatility due to some batches by chance containing more favorable images for classification, achieving a much higher (biased) prediction, and increasing the overall accuracy for that batch.

These results indicate that Adam converges at a significantly faster rate than SGD. Within the first ~200 batches, ADAM's training accuracy already shot up past 85%; by ~600 batches, it's hovering around 90%. In contrast, SGD is only up to ~33% after 200 batches and only reaches about 77-78% on average, even by the end of training. Moreover, Adam also generalizes better (91.5% validation accuracy vs.78.3%), suggesting that SGD is over-fitting under the hyperparameters

SGD shows more volatility of 96.6% from a single batch, as shown in Table 2.3, indicating that it can occasionally "hit" a very high accuracy or alluding to that singular batch holding more favorable images for classification.

Therefore, Adam yields both faster convergence and superior validation performance when compared to SGD, making it preferred for our lightweight NumPy-based CNN and later testing with BreakHis.

2.2.4 Sample Results

After training, the model can be used to recognize digits. Example results produced by both models on the test set are illustrated in Figure 2.3.

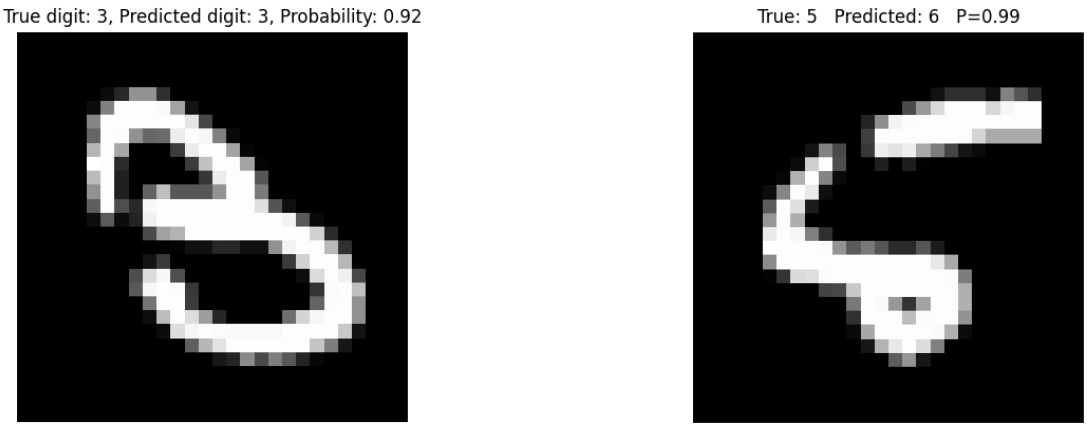


Figure. 2.4. Examples of correctly classified (left) and misclassified (right) numbers from the test set

2.3 BreakHis

2.3.0 Prelude

This project uses only 200× magnification images from the BreakHis dataset to ensure consistent resolution and reduce variability from different magnification levels. The 200× images offer a good balance of detail and efficiency, helping the model learn key features like cell density and tissue structure. To further simplify the task, only binary labels (benign vs. malignant) were used, as the goal is to test the model's efficiency rather than its ability to classify tumor subtypes.

2.3.1 BreakHis dataset

The BreakHis (Breast Cancer Histopathological Image Classification) (see Section 1.2.1) dataset was introduced by Spanhol et al. (2016) [5] to address the scarcity of publicly available, annotated histopathology images for breast cancer research. It has since become a benchmark dataset for deep learning-based classification studies in medical imaging.

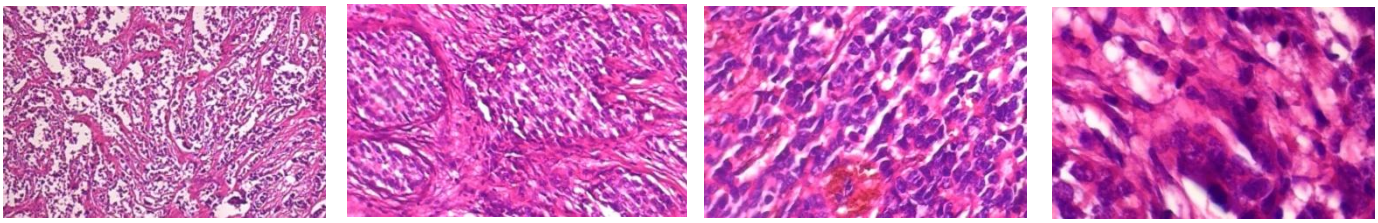
The dataset comprises a total of 7,909 microscopic images of breast tumor tissues collected from 82 patients, obtained using Hematoxylin and Eosin (H&E) staining that highlights nuclei in blue/purple and cytoplasm and extracellular matrix in pink. This color contrast allows for the extraction of features such as nuclear size and shape, cellular boundaries, and texture, which are vital for tumor classification.

Images are organized into two primary classes: 1) Benign (non-cancerous) and 2) Malignant (cancerous). Each image is captured at one of four magnification factors: 40X, 100X, 200X, and 400X

Magnification	Benign	Malignant	Testing set
X40	625	1370	1995
X100	644	1437	2081
X200	623	1390	2013
X400	588	1232	1820
Total of images	2480	5429	7909

Table 2.4 BreakHis Dataset Structure taken from Yassir Benhammou's 2018 study [31]

The dataset consists of RGB color images originally captured at a resolution of approximately 700x460 pixels, which were subsequently resized to 224x224 pixels to standardize input size for CNN architectures such as VGG16 and ResNet50, and the custom-built model. Focusing specifically on the 200X magnification subset, the dataset includes 2013 images (623 benign and 1390 malignant). We can see from the table that there is an overwhelming class imbalance favoring malignant samples. This class imbalance necessitated additional strategies during and before training to prevent bias towards the majority class (continued in section 2.3.2).



(a) (b) (c) (d)

Fig. 2.6. Samples of malignant (cancerous) breast histopathology images acquired from the BreakHis data set, illustrated in different magnification factors [5]. (a) 40X, (b) 100X, (c) 200X, (d) 400X

2.3.2 BreakHis Preprocessing

Within the provided GitHub repository [0], the script "splitting.py" recursively traverses the BreakHis directory [12], filtering for valid image files (.png, .jpg, .jpeg) under the specified magnification level (200X). It collects images by class (benign or malignant), randomly shuffles them, and splits them into three subsets:

Training set	Validation set	Testing set
80%	10%	10%

Table 2.5 Dataset splits

The splitting is performed independently for each class, ensuring balanced representation across all subsets. Images are copied into class-labeled subdirectories (e.g., train/benign/, test/malignant/) for compatibility with data loaders.

Different preprocessing strategies were applied based on the architecture requirements of the models used: VGG16, ResNet50, and my custom CNN. The preprocessing included resizing all images to 224x224 to match the input sizes required by VGG16 and ResNet50. Using TensorFlow's preprocess_input function, each respective model was applied to normalize based on ImageNet statistics. Data augmentation was applied to the training sets using random rotations, width and height shifts, shear and zoom transformations, and horizontal and vertical flipping. For validation and training data, only preprocessing was applied without augmentation. The custom CNN used the same preprocessing function as VGG16 and followed a similar augmentation pipeline. However, in contrast to the other two models, my custom CNN accepts the train, test, and validation datasets in the form of fully preloaded NumPy arrays. Therefore, the custom CNN data loading also included a step where all images and labels from training, validation, and test sets were fully loaded into memory using a custom data collection method. This approach allowed for greater flexibility in handling datasets during model training and evaluation.

Augmentation Type	Details
Rotation	Random rotation up to 40°
Width Shift	Random shifts up to 20% width
Height Shift	Random shifts up to 20% height
Shear	Random shear up to 20%
Zoom	Random zoom up to 20%
Horizontal Flip	Enabled
Vertical Flip	Enabled
Preprocessing	Based on the model

Table 2.3. Data augmentation applied to the dataset for all three models

One limitation encountered in the BreakHis dataset was a disparity in the number of images between the malignant and benign classes. Specifically, there were significantly more malignant images than benign ones (see section 2.3.1). This imbalance presents a challenge, as it may lead to a biased model that performs well on the majority class

(malignant) but poorly on the minority class (benign), ultimately reducing the model's generalization ability and reliability in real-world applications. To address this issue, several mitigation strategies were employed:

1. Class-balanced data splitting: The dataset was split into training, validation, and test sets in a way that preserved the class ratio, minimizing skew during evaluation.
2. Data augmentation: The minority class (benign) was over-sampled by more extensive augmentation, increasing its representation in the training data and reducing overfitting.
3. Class weighting in loss function: Weighted cross-entropy loss was used when training the model to assign higher penalties to misclassifying the minority class.
4. Evaluation using per-class metrics: Precision, recall, and F1-score were computed for each class to ensure that the model was not simply favoring the majority class.

2.3.3 BreakHis Architecture

Similar to the MNIST implementation (see Section 2.2.3), the custom CNN model for BreakHis processes batches of data with a batch size of 32. However, unlike MNIST's grayscale images, each BreakHis image contains three color channels, Red, Green, and Blue (RGB), resulting in an input shape of [224, 244, 3] per image. These images are first preprocessed and normalized, grouped into batches of size 32, and reshaped to include the batch dimension, forming tensors of shape [batch size, height, width, channels], taking the form of [32, 224, 224, 3].

The architecture consists of three convolutional layers, also using ReLU activations, with progressively increasing filter counts (16 > 32 > 64). Each Convolution layer is then immediately followed by a batch normalization layer to stabilize training and accelerate convergence. After the final convolution, just like in section 2.2.3, the resulting feature maps are flattened into a vector. This is then passed through a fully connected dense layer, and finally to the final dense output layer with 2 neurons (for each of the classes, malignant or benign) and a SoftMax activation to perform binary classification.

Several design choices were made to optimize both the learning performance and efficiency of the model, as the goal was to create a lightweight CNN that performed well on the dataset (>80% accuracy). In addition to using standard 3x3 filters, an initial stride of 4 was selected in the first convolutional layer to aggressively down-sample the spatial dimensions of the input image (224x224), reducing the processing and number of parameters early in the network. This approach eliminates the need for a separate pooling layer in early stages, though it can come at the loss of some information. To mitigate this, the stride was reduced to 2 in the second and third convolutional layers.

Following the second convolution, a max pooling layer with a stride of 2 was applied to further reduce the spatial dimensions while retaining dominant features. This strategy is preferred over using a single convolutional layer with a stride of 4, as pooling preserves important features by selecting the maximum value in each "pooling window", instead of going over them entirely. The third and final convolutional layer used the same parameters as the second but did not include additional pooling, ensuring that the output retained sufficient spatial detail before being flattened.

To increase stability and convergence speed, a batch normalization layer was implemented and applied after each convolutional layer, helping reduce internal covariate shift. After the third convolution, the output feature maps were flattened into a vector of 2304 features, which was passed through a fully connected dense layer with 128 neurons before being fed into the final dense output layer with 2 neurons and a SoftMax activation, performing binary classification of the images into either benign or malignant categories.

Layer	Input shape	Output shape	Filter Number	Stride	Filter/Kernel Size	Activation
Convolution	(32, 224, 224, 3)	(32, 56, 56, 16)	16	4	3	ReLU
Convolution	(32, 56, 56, 16)	(32, 28, 28, 32)	32	2	3	ReLU
Pooling	(32, 28, 28, 32)	(32, 14, 14, 32)	-	2	2	-
Convolution	(32, 14, 14, 32)	(32, 7, 7, 32)	32	2	3	ReLU
Flatten	(32, 7, 7, 32)	(32, 1568)	-	-	-	-
Dense	(32, 1568)	(32, 128)	-	-	-	-
Dense (with SoftMax)	(32, 128)	(32, 2)	-	-	-	-

Table 2.4. CNN architecture for BreakHis, including hyper-params, stride, and kernel size

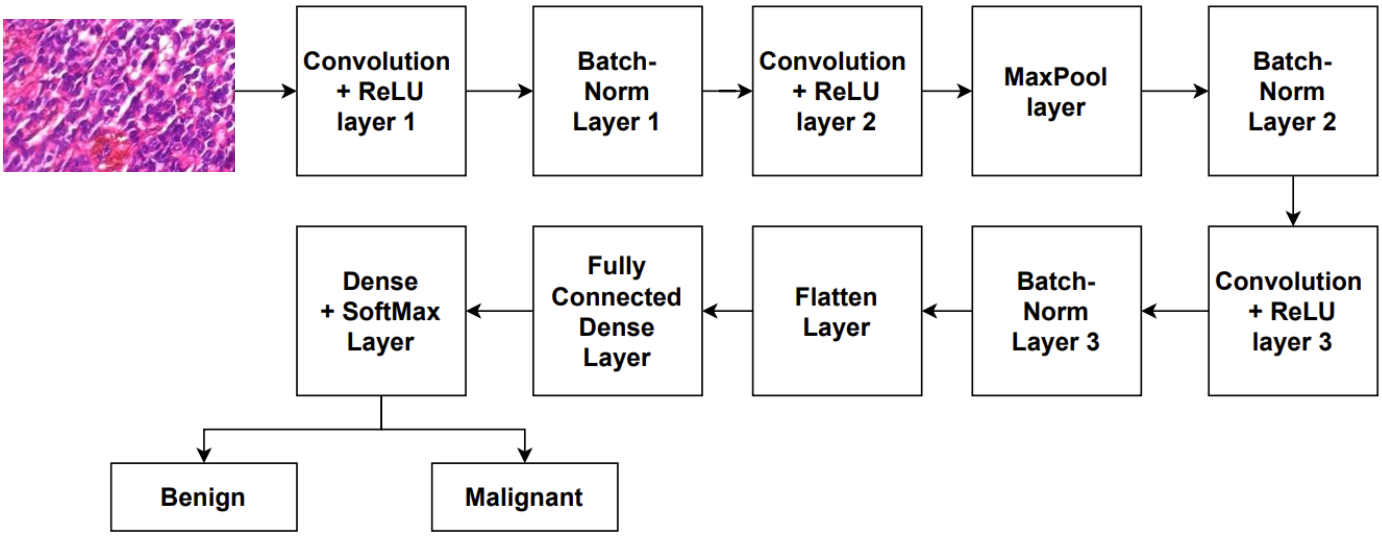


Fig. 2.7. Visual representation of full CNN layer architecture

Each training epoch (a full pass through the training set) consists of 20 evaluation steps. At each step, the model is evaluated, and metrics such as average training accuracy over the last 20 steps, validation accuracy, loss, and learning rate are logged into a CSV file for monitoring. A learning rate scheduler reduces the learning rate every 5 steps to help

convergence. Early stopping is applied with a patience of 10 steps, terminating training when no improvement in validation accuracy is observed, to reduce runtime and prevent overfitting. Similar to Alessandro Saviolo and Alfak, O's implementation, L2 regularization with a lambda value of 0.1 is applied to the weighted cross-entropy loss to improve generalization.

Hyper parameter	Value
# Batches per epoch	51
# Steps per epoch	20
Initial Learning Rate	0.01
Optimizer	Adam
Lambda	0.1
Epochs	5
Patience	10 Steps

Table. 2.5. Hyperparameters used when fine-tuning the custom CNN

2.3.4 VGG16 and ResNet50

To establish strong baselines, both ResNet50 and VGG16 were trained and evaluated on the BreakHis 200× dataset. The models used ImageNet-pretrained weights to take advantage of transfer learning, which helped speed up training and improve performance on the small dataset. Aside from standard preprocessing and augmentation (see Section 2.3.2), the first layers were frozen, only the last 20 layers were fine-tuned to adapt to the classification task. Early stopping (patience of 10) and a batch size of 64 were used during training. Both models were trained until they reached at least 90% validation accuracy. Their performance was then evaluated using precision, recall, and F1-score for each class to account for the dataset's class imbalance.

Hyper parameter	Value
# Epochs	200
Loss Function	Categorical Cross-Entropy
Initial Learning Rate	0.0001
Optimizer	Adam
Class Weights	Balanced
Batch Size	64
Patience	10 Steps

Table. 2.6. Hyperparameters used when fine-tuning pre-trained CNN

Chapter 3

Validation, Results, Implementation, Evaluation, and Discussion

3.0 Prelude

All saved CSV files containing metrics during training are provided in the GitHub repository [0] for reference.

3.1 Pretrained Model Results

3.1.1 Accuracy

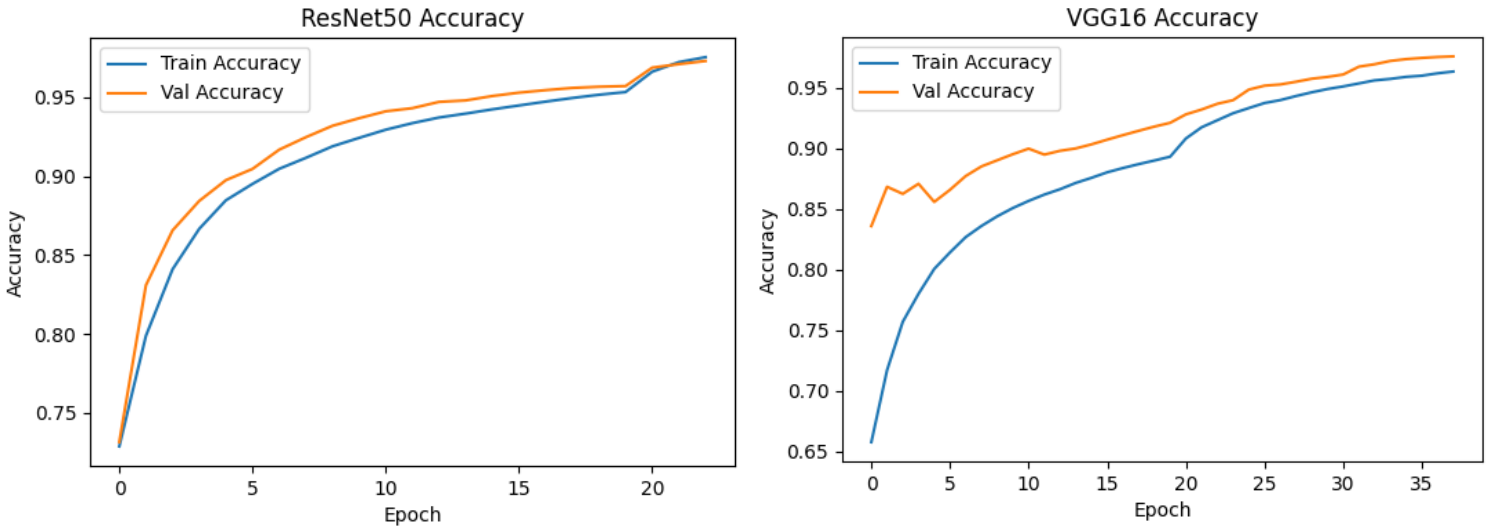


Fig. 3.1. Accuracy graph of ResNet50 and VGG16, averaged over a running mean for clarity

Both ResNet50 and VGG16 achieved high training and validation accuracy, indicating successful learning and generalization. ResNet50 reached a slightly higher training accuracy at 98.82% compared to VGG16 at 97.14%, suggesting it fits the training data better. However, VGG16 achieved a superior validation accuracy of 98.51%, outperforming ResNet50's 97.51%. This implies that VGG16 may generalize slightly better to unseen data despite being a deep and heavier model. Both validation accuracy on average are slightly higher than training accuracy, this is to be expected.

It is also to be noted that early stopping was triggered at epoch 24 for ResNet50, while for VGG16 it occurred at epoch 39. This implies that ResNet50 converged faster during training, reaching its optimal validation performance earlier, whereas VGG16 required more epochs to achieve peak performance. This could suggest that ResNet50's architecture is more efficient when learning, whereas the longer training of VGG16 may have resulted in better generalization on the validation set, as it had a higher final validation accuracy.

Overall, both ResNet50 and VGG16 demonstrated strong and comparable performance, achieving similarly high levels of accuracy. While ResNet50 reached a training accuracy of 98.82% and a validation accuracy of 97.14%, VGG16

achieved 97.14% training accuracy and 98.51% validation accuracy. This makes a strong baseline for the large pre-trained models.

3.1.2 Loss

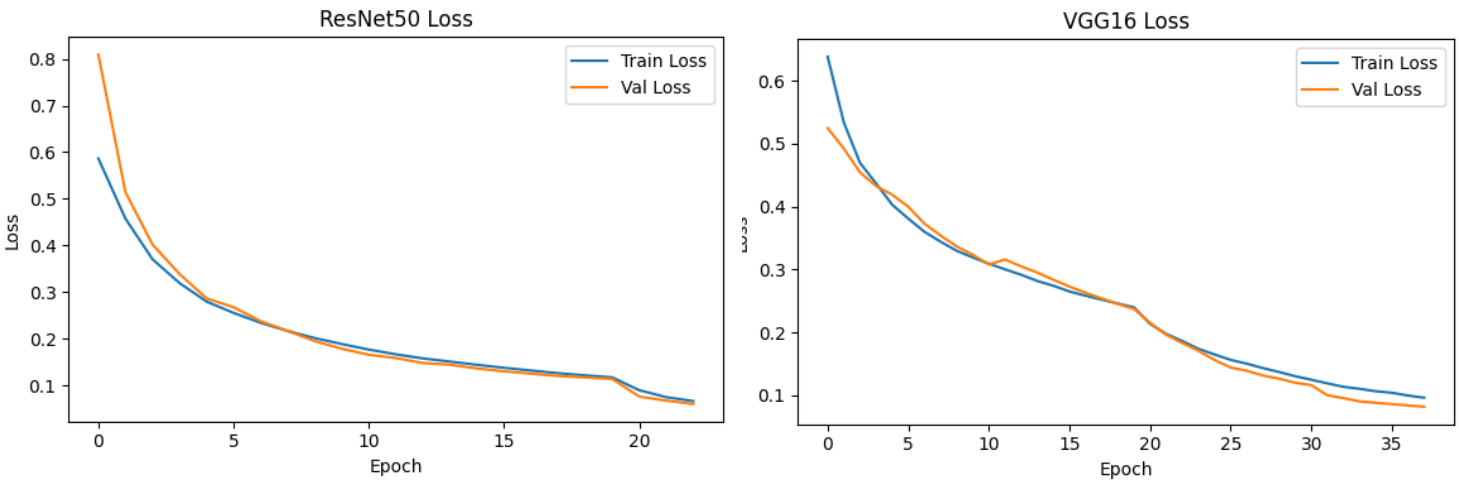


Fig. 3.2. Loss curves of ResNet50 and VGG16, averaged over a running mean for clarity

When comparing the loss curves, both ResNet50 and VGG16 demonstrated a steady decline in training and validation loss, indicating the models were learning effectively throughout the training process. ResNet50 showed a faster convergence, with training loss rapidly decreasing and stabilizing by the time early stopping was triggered. Its final training loss was 0.0295, and validation loss was 0.0230, suggesting a close match between training and validation performance with minimal overfitting.

VGG16 required more epochs to reach its optimal performance, with early stopping triggered at epoch 39. It concluded with a slightly higher training loss of 0.0761, but a lower validation loss of 0.0557. Overall, on average, the loss curves for both models were smooth and stable, with no signs of sharp spikes or divergence. This indicates the training process was well-regularized and that both models maintained good generalization ability without significant overfitting.

3.1.3 F1-Score, Precision, and Recall

The confusion matrix was derived from the test set. It is observed that ResNet50 outperformed VGG16 on most metrics in the test set, achieving an overall accuracy of 96.04%, compared to 93.56% for VGG16. In terms of precision, ResNet reached 98.52%, meaning it had few false positives and was highly reliable when predicting malignant cases. VGG16, while slightly behind in precision (93.75%), demonstrated strong recall at 97.12%, which suggests it was more sensitive to malignant cases and better at minimizing false negatives. ResNet's recall, at 95.68%, was still high, indicating balanced performance. The F1-score (a metric that considers both precision and recall) achieved by ResNet50 is 97.08%, and VGG16 scores 95.41%.

While VGG16 achieved a higher validation accuracy during training, ResNet50 outperformed it in F1-score on the test set. This may be due to differences in how each model handles the malignant class. VGG16's higher recall results in more correct malignant predictions, but at the cost of increased false positives, which lowers its precision and F1-score. ResNet50, in contrast, maintained a better balance between precision and recall, leading to a higher F1-score. This may suggest the test set slightly favors models with better malignant class precision, or that ResNet50 simply generalizes better with the test set.

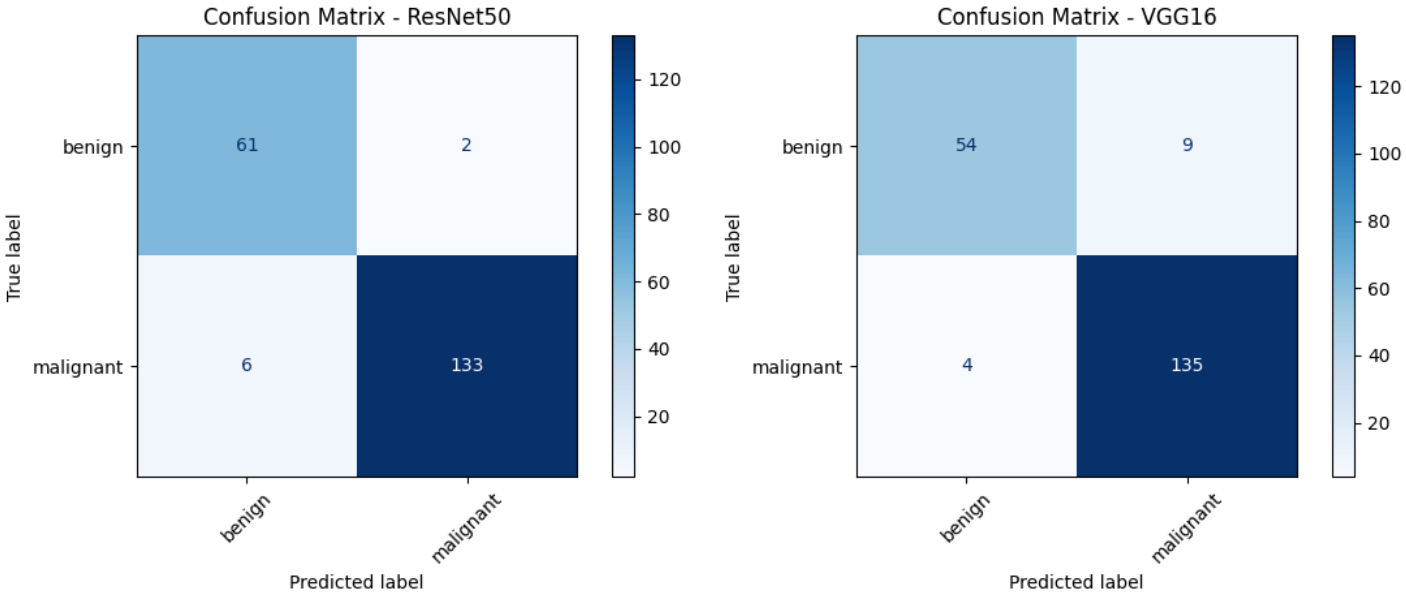


Fig. 3.3. Confusion Matrix of ResNet50 and VGG16

3.2 Custom Model Results

3.2.1 Preliminary information

The following data was recorded at roughly every 2.5 batch intervals, with 1/20 epochs representing one interval (one epoch has 51 batches). As each step represents approximately 1/20 of an epoch, these values reflect partial-progress snapshots rather than full epoch summaries. With 42 steps logged, the training spanned roughly two epochs before early stopping was triggered due to a lack of improvement in validation accuracy. The model was always evaluated using the full evaluation set during each step.

This logging choice was chosen to account for the extended training time and limited computational resources available. Unlike the pre-trained CNNs, which can make use of TensorFlow's CuDNN and CuDA support for GPU-accelerated training, the custom CNN was implemented entirely using NumPy and restricted to a CPU-only setup; training beyond a few epochs became too computationally expensive. Logging at regular intervals provided better insight into the model's learning progress while in a resource-limited environment.

3.2.2 Accuracy

The training accuracy increases rapidly from 59.38% to a peak of 90.63%, before settling at 84.38% over the first several logging steps. By the final logged step (roughly at the end of the second epoch), training accuracy had settled at 84.38%, indicating solid overall training. Validation was more volatile, fluctuating between 16.42% and 89.05%, but ultimately stabilized, ending at 88.06%.

Averaging across the second epoch (steps 20 onward), training and validation accuracy were 77.38% and 79.67%, respectively. These results, despite being derived from a short training run, show the custom CNN's potential, though the high variance in validation accuracy suggests further regularization or more training could improve generalization.

Overall, the training accuracy of the custom CNN showed a consistent upward trend, indicating effective learning throughout the training period.

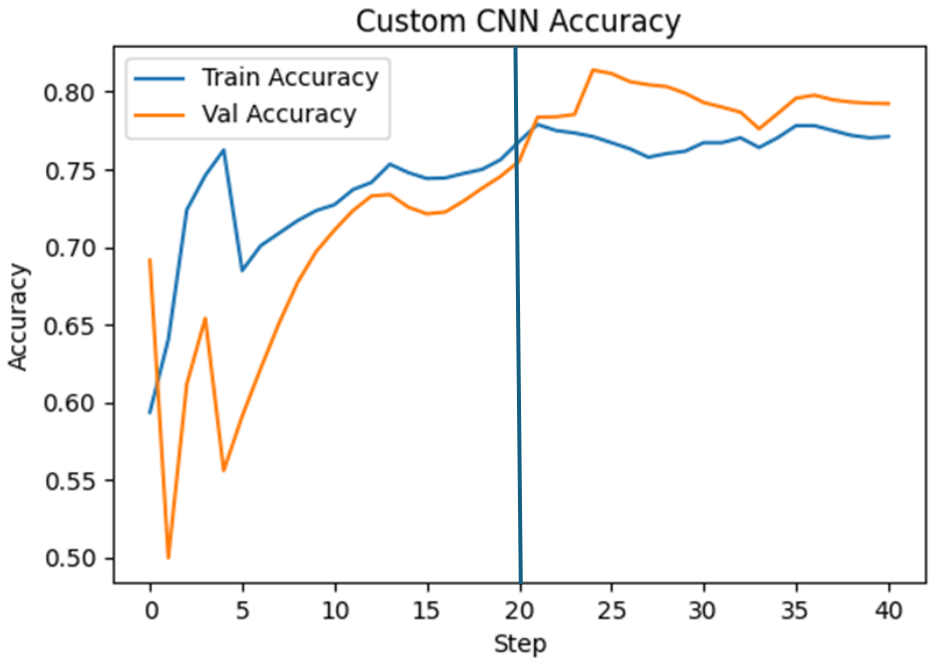


Fig. 3.4. Accuracy graph of the Custom CNN, averaged over a running mean for clarity, with the straight line meaning the end of one epoch

3.2.3 Loss

The training loss for the custom CNN started at 4.9 and decreased steadily to 2.36 by the final logged step, demonstrating consistent learning progress over the training period. The validation loss, which began at a much higher 8.25, fluctuated significantly throughout the training period, reaching a peak of 21.09 but eventually stabilizing and dropping to 0.58 at the final step.

Averaging across the second epoch (steps 20 onward), training and validation losses were 2.452 and 0.9198, respectively. This gap (validation being lower) suggests the model was beginning to generalize well, with the validation

loss decreasing more rapidly than the training loss, a sign of strong performance emerging just before early stopping was triggered.

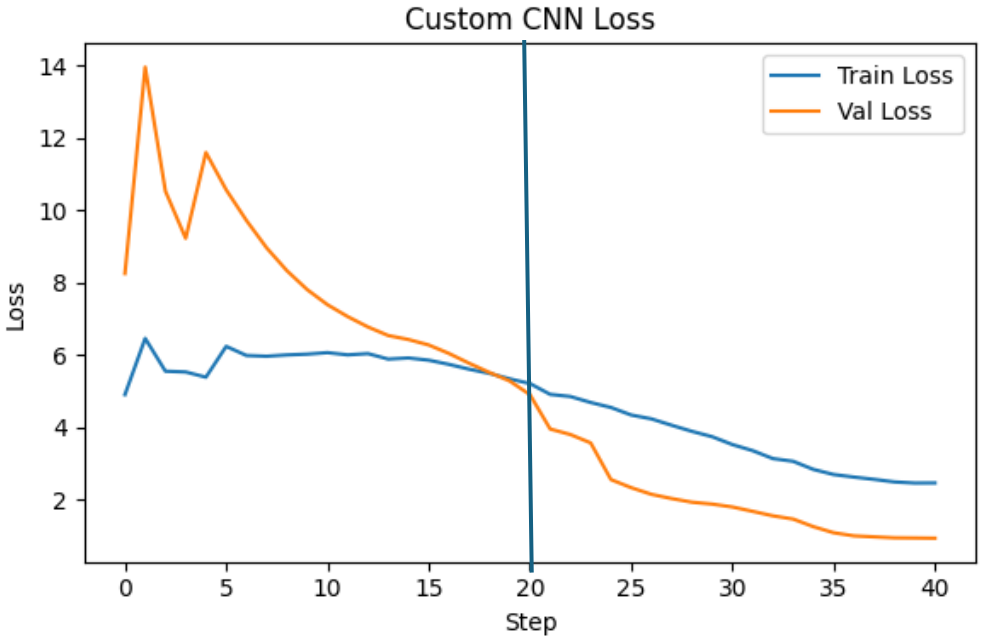


Fig. 3.5. Loss curves of the Custom CNN, averaged over a running mean for clarity, with the straight line meaning the end of one epoch

3.2.4 F1-Score, Precision, and Recall

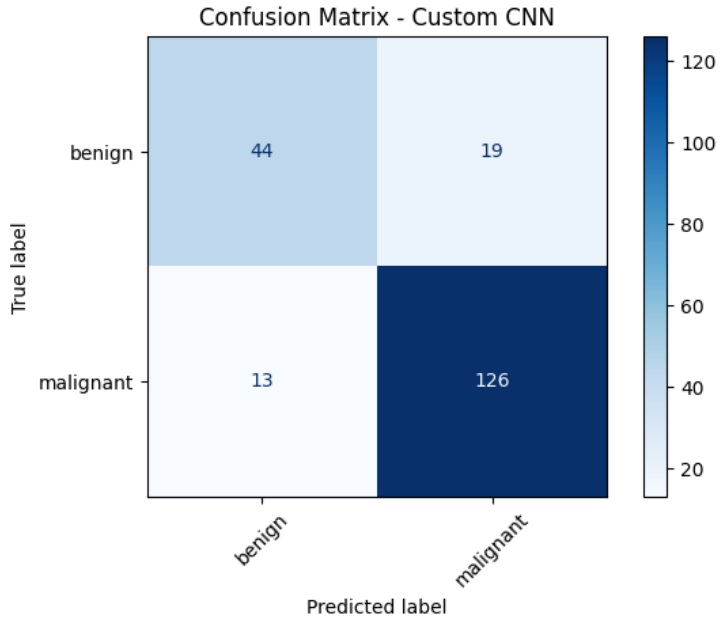


Fig. 3.6. Confusion Matrix of the custom CNN

The custom CNN's performance on the test set is illustrated in Figure 3.6. The model correctly identified 126 malignant and 44 benign cases, while misclassifying 13 malignant samples as benign (false negatives) and 19 benign

samples as malignant (false positives). This results in an overall accuracy of 84.16%, with a precision of 86.06%, a recall of 90.65%, and an F1-score of 88.73%.

The per-class accuracy of benign samples is 70% and 91% on malignant samples. This indicates that despite our efforts at oversampling and weighting, the model is still slightly biased towards malignant cases. While this bias helps minimize missed malignant cases, a goal in cancer detection, it comes at the cost of a higher false positive rate.

Overall, these results show that the model is reasonably effective at identifying benign and malignant cases, though its relatively high number of false positives suggests room for improvement, such as improved regularization, balancing, or architectural improvements.

3.2.5 ROC Curve

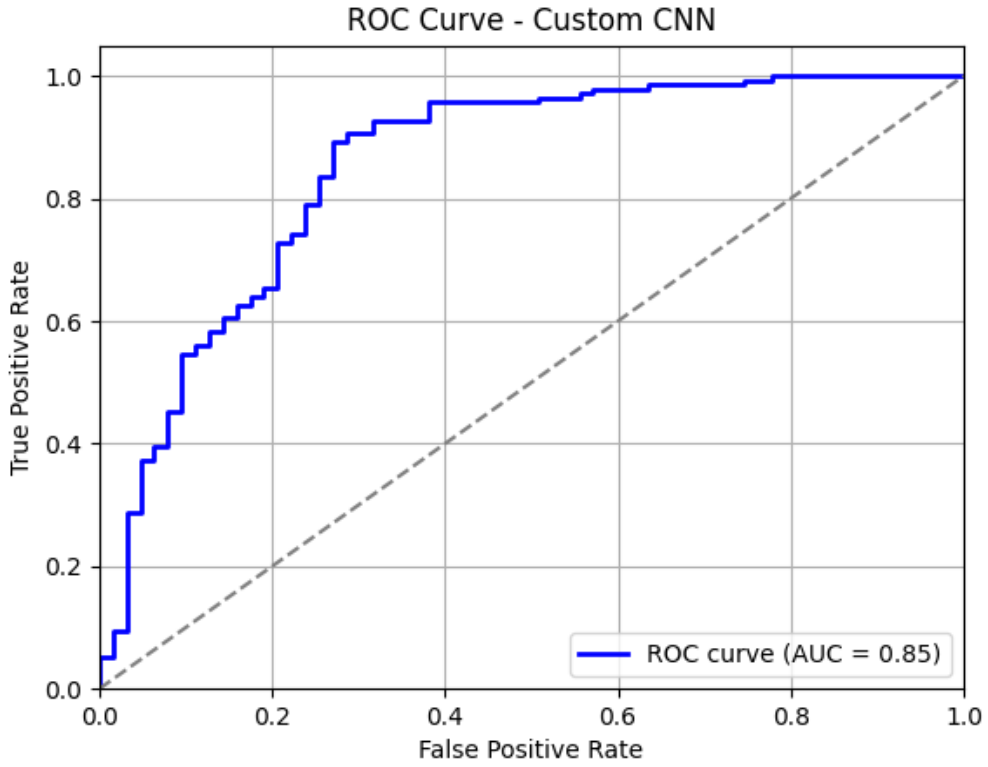


Fig. 3.7. ROC Curve of the custom CNN

The ROC curve for the custom CNN (Figure 3.7) shows strong diagnostic performance with an AUC of 0.85. The model achieves high recall and relatively low false positives, making it effective for screening malignant breast tissue.

It is observed that the curve bends towards the top-left, indicating high sensitivity and low false positive rate, desirable traits for cancer screening models. Compared to random chance (AUC = 0.5), the custom CNN works as expected and is reasonably reliable.

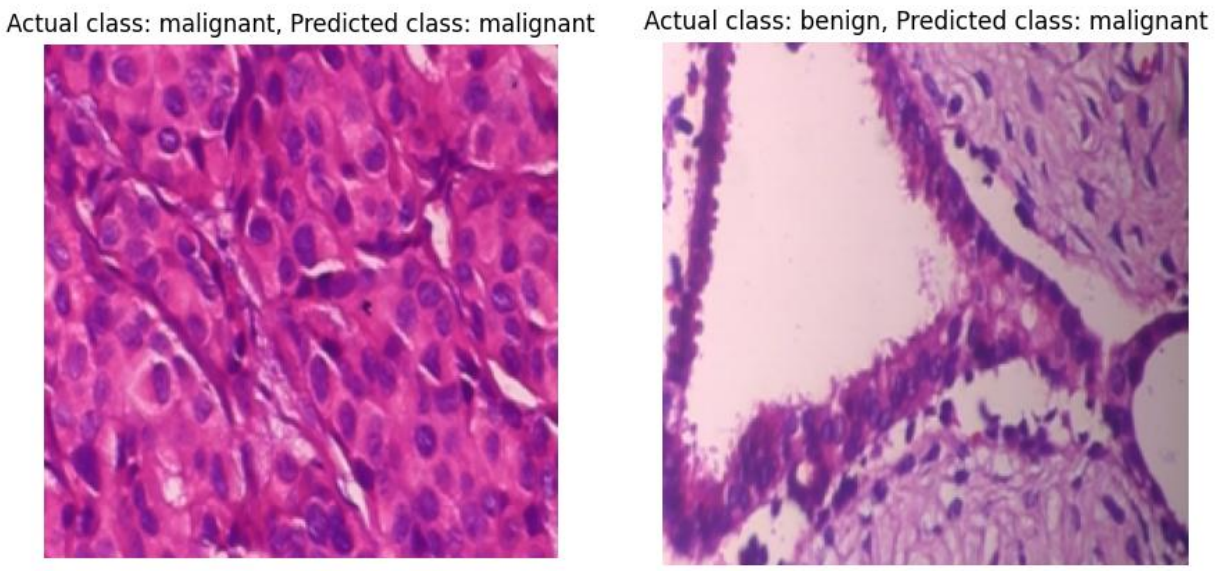


Fig. 3.7. Example output from the Custom Model of correctly classified (left) and misclassified (right) samples. Note that the samples were resized to 224x224 (see section 2.3.2), hence the square shape

3.3 Efficiency and Performance of the Models

Model	F1-Score	Precision	Recall (Malignant)	Recall (Benign)	Test Accuracy	Validation Accuracy	Iterated Epochs
Custom CNN	88.73%	86.90%	90.65%	70%	84.16%	89.05%	2
VGG16	95.41%	93.75%	97.12%	85.71%	93.56%	98.51%	39
ResNet50	97.08%	98.52%	95.68%	96.83%	96.04%	99.5%	24

Table 3.1. Overview of model performances

Table 3.1 summarizes the model performances discussed in Sections 3.1 and 3.2. While the pretrained models achieved the highest absolute performance in F1-score and overall accuracy after full training (24-29 epochs on GPU), the custom CNN demonstrated remarkable efficiency. It achieved an F1-score of 88.73%, a malignant recall of 90.65%, and an overall accuracy of 84.16%.

After training and saving each model's best model file, the Custom CNN took up just 2,475 KB, compared to 112,620 KB for VGG16 and 162,536 KB for ResNet50. This model size reduction is expected as the Custom CNN is designed to have fewer parameters and a more minimal architecture overall.

In addition to the smaller file size, the custom model achieved the fastest inference time (time required to predict a single sample), averaging 0.31 seconds per sample, compared to 1,75 seconds for VGG16 and 1.85 seconds for ResNet50, making the model significantly more responsive.

The VGG16 model contains approximately 14.7 million parameters, split into 7.08 million trainable and 7.64 million non-trainable (frozen) parameters due to transfer learning. Similarly, ResNet50 comprises 23.6 million parameters, of which 8.94 million are trainable. In contrast, the Custom CNN comprises only 319,106 trainable parameters.

A more meaningful comparison can be made by examining performance at Epoch 2, the point at which the custom CNN completed its entire training. At epoch 2, ResNet50 achieved 86.89% training and 93.03% validation accuracy, and VGG16 achieved 77.64% training and 90.05% validation accuracy. These results are still comparable to the custom CNN, showing that the custom model, despite being smaller and having fewer parameters, still converged rapidly and performed competitively against these industry-standard models.

Model	Total Params	Trainable	Non-trainable	Training Time	Inference Time	Hardware
Custom CNN	319,106	319,106	0	49 min	0.31s	CPU
VGG16	14.7M	7.08M	7.64M	22.75 min	1.75s	GPU
ResNet50	23.6M	8.94M	14.66M	16 min	1.85s	GPU

Table 3.2. Number of parameters and training times of each model. Inference time was averaged over 20 predictions.

It is noted that the custom CNN built using NumPy is not able to leverage GPU acceleration via TensorFlow's CuDNN and CUDA. Consequently, its training time was substantially longer at 49 minutes for just 2 epochs, compared to the GPU-accelerated pre-trained models, which completed 39 (VGG16) and 24 (ResNet50) epochs in just 22.75 minutes and 16 minutes, respectively.

It is also observed that ResNet performed slightly better overall than VGG16. These results are in line with the findings from Rakhlin et al.'s 2018 study [29], where a ResNet model outperformed VGG16 in most metrics (see Section 1.2.1).

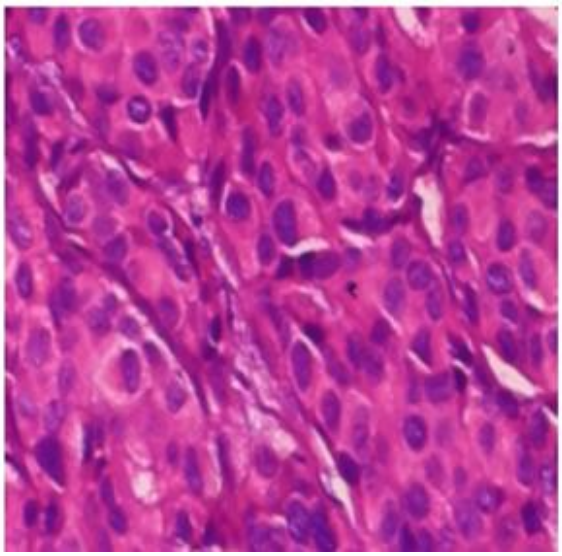
To provide a fairer comparison, both pre-trained models were also trained using TensorFlow without GPU acceleration. Each model was run for 10 epochs on the CPU, and the average training time per epoch was recorded. For ResNet50, the average time per epoch increases significantly to approximately 40 seconds (on GPU) to 6-8 minutes (on CPU). At this rate, completing the full 24 epochs would require approximately 2.7 hours. Similarly, VGG16 showed a comparable increase in training time, with each CPU-only epoch taking 6-8 minutes, making the full 39 epochs require approximately 4.5 hours.

3.4 Feature Maps

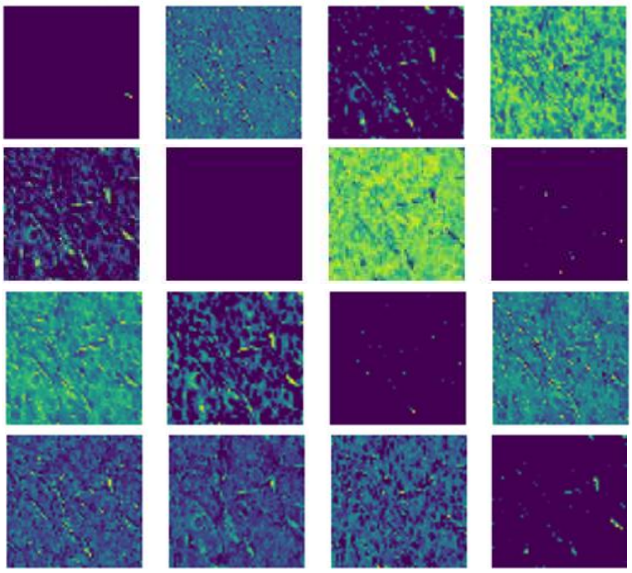
The different feature maps produced by the model across the layers are illustrated in Figure 3.8. Each layer produces x feature maps, where x is the number of filters processed by each layer (see Table 2.4). By visualizing the feature maps, we can gain some understanding of what features the model detects. This is important to fine-tune the model architecture, since we can observe if some parts of our desired features (i.e., texture, colors) are detected among all the

layers of activation or if the activations fade out at a certain layer. In general, early layers of the network detect low-level features (e.g., color, edges), and the deeper layers of the network detect high-level features (e.g., shapes, objects).

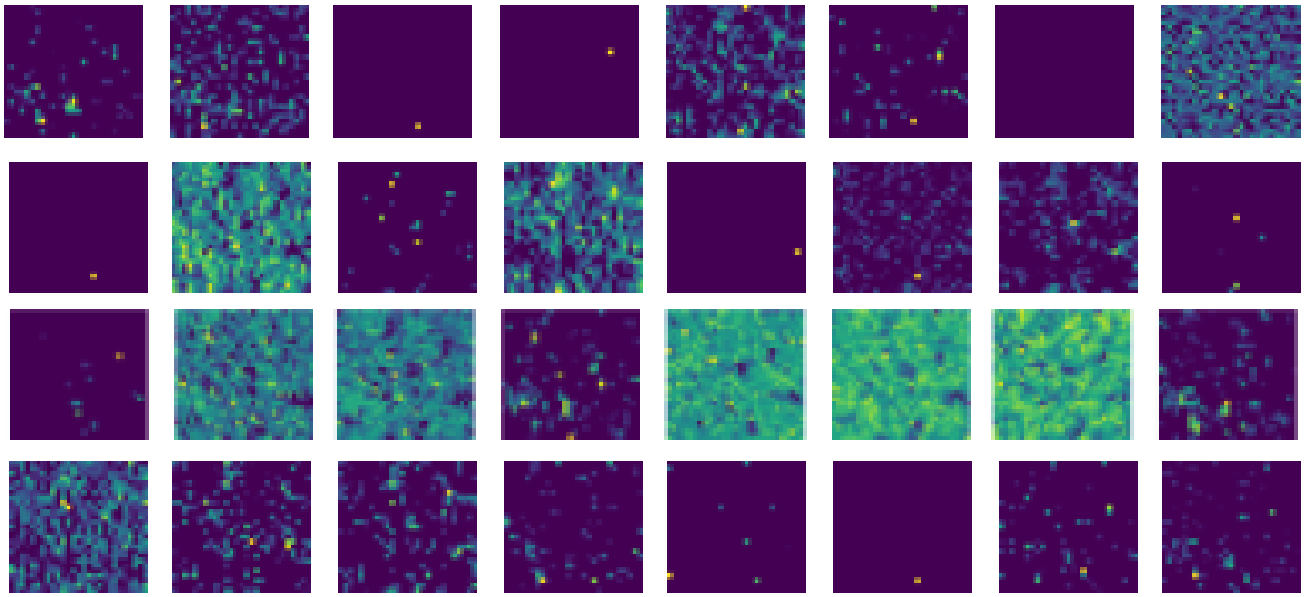
It is observed from the illustration below that the raw input image is a different color from the resulting feature maps, which is partly because of the preprocessing operations applied to the image (see Section 2.3.2), which adjust the pixel intensity ranges before it is passed into the network.



(a) Correctly classified input image from Figure 3.7

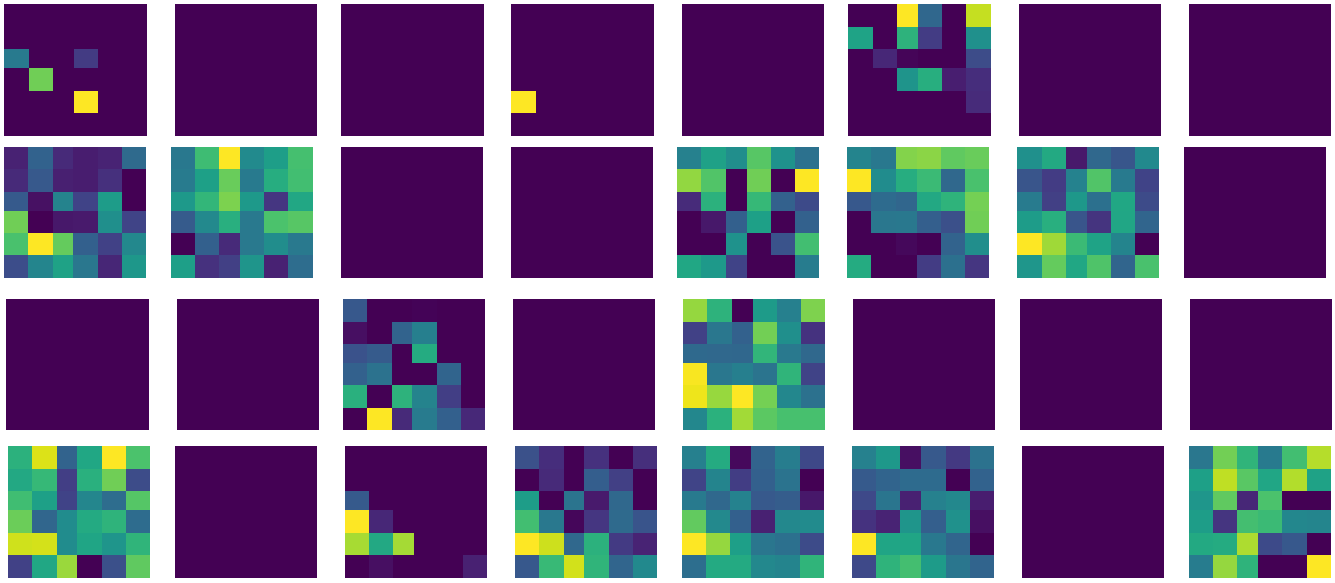


(b) First Convolution Layer (16 filters)



(c) Second Convolution Layer (32 filters)

Figure 3.8. Part 1: Feature maps produced by the first and second convolution layers



(d) Third Convolution Layer (32 filters)

Figure 3.8. Part 2: Feature map produced by the third convolution layer

The visualization above provides insight into the hierarchical feature extraction process of the custom CNN. These maps illustrate how the network transforms the input image into increasingly abstract representations to classify between benign and malignant images.

The feature maps in the first convolutional layer show sparse activations, with only a few filters responding strongly to regions of the input image. These activations likely correspond to simple features such as edges, color gradients, or texture patterns.

In the second layer, there are increased activations across more of the image in several filters. This means that the model is now detecting more complex patterns. Similar to the first layer, not all filters are activated, which is good; it shows that the filters are becoming specialized for different types of features.

In the third layer, the feature maps become much more abstract, showing dense and varied activations with larger patterns. The maps appear to be more blocky and low resolution, which is to be expected due to the down-sampling from previous layers (e.g., strided convolution, max-pool layer). Compared to the first layer, many more filters are significantly more activated (have color and are not black).

As depicted across the different feature maps, the CNN successfully learns a hierarchical representation of histopathological data.

Chapter 4
Discussions, Conclusion, and Future Work

4.1 Interpretation of the Results in Context

The primary objective of this project was to develop a lightweight convolutional neural network that could classify breast cancer histopathological images with competitive accuracy compared to established pretrained models such as VGG16 and ResNet50.

Specific performance goals included achieving an overall accuracy of greater than 80% and a recall score exceeding 90% for malignant cases. These metrics were chosen due to the importance of detecting malignancies in medical applications, where false negatives could lead to life-threatening outcomes.

The results were met successfully as the custom CNN achieved an accuracy of 84.16%, with a malignant recall of 90.65% and an F1-score of 88.73%. These results are particularly significant considering the model was trained entirely on a CPU-only system with a limited number of parameters (319,106) and no access to GPU acceleration. The custom model demonstrated rapid convergence, completing its full training cycle within two epochs and achieving performance comparable to VGG16 and ResNet50 at the same training point (epoch 2), despite being significantly lightweight.

Several factors contributed to the custom CNN's strong performance despite hardware limitations. These included using the Adam optimizer for faster convergence, ReLU activations with proper initializations, training in batches to make training more stable, a weighted loss function to help address the class imbalance in BreakHis, and the use of a batch normalization layer after each convolutional layer to stabilize training. In addition, techniques such as data augmentation, early stopping with a patience of 10, and a learning rate scheduler to prevent overfitting and wasted cycles, making the CNN more efficient when training.

The pretrained models naturally outperformed the custom CNN in raw accuracy and F1-score, with ResNet50 reaching 96.04% test accuracy and VGG16 reaching 93.56%. However, this improved performance came at the cost of significantly larger model sizes (14.7-23.6 million parameters vs 319,106 parameters of the custom CNN), longer inference times, and higher hardware requirements. In contrast, the custom CNN offered superior inference speed (0.31s/sample), significantly smaller best model file sizes (2,475 KB), and a parameter count more than 40x smaller, making it a promising candidate for deployment in low-resource environments such as rural clinics or mobile devices.

A trade-off observed during this study was between peak model performance and efficiency. While pretrained models achieved the highest accuracy, they required significantly more memory, longer inference times, and relied heavily on GPU acceleration, resources that may not be available in all deployment environments. In contrast, the custom CNN offered a far more compact and efficient alternative, achieving competitive accuracy while being optimized for performance even on CPU-only systems.

4.2 Limitations

While this project achieved its core objectives and demonstrated the reliability of the lightweight custom CNN for histopathological image classification, several limitations should be acknowledged.

Due to the use of NumPy for implementing the custom CNN, the model is restricted to training solely on the CPU. This significantly limited the training speed and computational efficiency. As a result, the custom model is trained for only 2 epochs, which may not be sufficient for full convergence or optimal performance. In comparison, even the pretrained models, when trained on the same CPU, completed two epochs in approximately 14 to 16 minutes, whereas the custom CNN required around 49 minutes.

This discrepancy is likely due to TensorFlow using highly optimized C++ and CUDA backends, including libraries like Eigen for linear algebra, cuDNN for GPU acceleration, and XLA (Accelerated Linear Algebra) for compiler-level optimization [32][33]. According to Harris et al. NumPy operations execute on a single thread and lack native GPU support [34], making them less suited for the high-throughput needs of deep learning applications.

The model was trained and evaluated solely on images captured at 200X magnification. Although this choice was because consistency in evaluation was needed to compare performances, it limits the model's generalizability to other magnification levels (e.g., 40X, 100X, or 400X), which are commonly used in diagnostic practice.

The BreakHis dataset exhibits a large degree of class imbalance at 200X magnification (623 benign images and 1390 malignant). Although performance metrics such as F1-score and recall, and techniques such as oversampling and class weights were implemented, the imbalance and class bias are still very present within the results (see Section 3.1.2 and 3.2.4). More sophisticated data balancing techniques or augmentation strategies could help address this in future iterations (see Section 4.4 Future Work).

The current NumPy custom CNN implementation does not include any model interpretability methods such as Grad-CAM, LIME, or SHAP. These tools are increasingly important in medical AI applications to justify and visualize model decisions. This is important because medical practitioners often require not only accurate predictions, but also insight into why a diagnosis was made, especially in cancer diagnosis. Incorporating such techniques would enhance the transparency of the model and its trustworthiness in clinical settings.

Looking at the feature maps (see Section 3.4), it is obvious that the custom CNN model employed a very aggressive down-sampling method (via large strides and max pooling) to drastically reduce feature map dimensions to improve computational efficiency and reduce training time. While the design choice amounted to faster inference and lower memory usage, it may have led to an overly abstracted feature map, causing the model to lose fine-grained spatial information crucial for distinguishing subtle features. As a result, some important features may have been discarded too early in the network, possibly limiting classification performance.

4.3 Future work

As mentioned in Section 2.2.3, using the ReLU activation function without proper weight initialization can lead to the “dead ReLU problem”, where neurons output zero and stop learning. While this issue was mitigated using He initialization in the current custom CNN, further precautions could be implemented in future versions. One promising solution is to adopt the Leaky ReLU activation function, as demonstrated in Alessandro Saviolo’s implementation [22]. Unlike standard ReLU, which outputs zero for all negative inputs, Leaky ReLU allows a small, non-zero gradient (e.g., 0.01 * x) when the input is negative. This ensures that neurons continue to receive gradient updates even when their activations are negative, reducing the risk of neurons becoming inactive.

Future work should consider migrating the custom CNN implementation from NumPy to Keras, a high-level API built on TensorFlow. Keras supports GPU acceleration via TensorFlow’s integration with CUDA/cuDNN, allowing for efficient training on NVIDIA GPUs. This would reduce training times significantly and enable the use of more complex architecture and longer training cycles, reducing the need for early aggressive down-sampling, allowing more features to be preserved for accurate classification. Keras provides built-in functions for model building, checkpointing, data augmentation, and visualization tools like TensorBoard and Grad-CAM, which would improve model performance and interpretability. Since Keras is part of TensorFlow, the highly optimized C++ and CUDA backends can be leveraged, providing significantly improved efficiency in terms of inference and training time.

The results show that despite implementing class weights and oversampling to address the class imbalance, the model is still relatively biased towards the majority class, which may lead to false positives (see Section 3.2.4). To address this, future work could implement more advanced class balancing techniques, such as Synthetic data generation. Techniques like the Synthetic Minority Oversampling Technique (SMOTE) have shown improvements in classifier performance across various domains with imbalanced data [35]. Alternatively, General Adversarial Networks (GANs) can create realistic synthetic histopathology images that augment the benign class, as demonstrated in studies like Frid-Adar et al. (2018), where GAN-augmented data improved liver lesion classification [37].

In this study, the classification task was limited to only 2 classes (malignant and benign), using images at a single magnification level to reduce the problem complexity. However, the BreakHis dataset contains not only images at multiple magnification levels, but also the different subtypes of malignant and benign tumors (e.g., Adenosis, Fibroadenoma). In real-world applications, pathologists evaluate tissue slides at multiple magnification levels and need to distinguish between the various subtypes. Future work should incorporate images from all magnification levels, helping the model learn features only visible at certain levels, allowing it to generalize better. Furthermore, expanding training from binary (2-class) to multi-class classification would allow the model to distinguish between the dataset’s 8 included subtypes (Adenosis, Fibroadenoma, Phyllodes Tumor, Ductal Carcinoma, Lobular Carcinoma, etc.). This can be achieved through multi-branch CNNs (multiple instance learning) [36], where each branch specializes in a specific magnification or subtype, and their final outputs are fused for final classification.

List of References

[0] Github Link: https://github.com/hsinJee/BreastCancerCNN

[1] Bray F, Ferlay J, Soerjomataram I, Siegel RL, Torre LA, Jemal A. Global cancer statistics 2018: GLOBOCAN estimates of incidence and mortality worldwide for 36 cancers in 185 countries. CA Cancer J Clin. 2018 Nov;68(6):394-424. doi: 10.3322/caac.21492. Epub 2018 Sep 12. Erratum in: CA Cancer J Clin. 2020 Jul;70(4):313. doi: 10.3322/caac.21609. PMID: 30207593.

[2] Azam S, Eriksson M, Sjölander A, Gabrielson M, Hellgren R, Czene K, Hall P. Mammographic microcalcifications and risk of breast cancer. Br J Cancer. 2021 Aug;125(5):759-765. doi: 10.1038/s41416-021-01459-x. Epub 2021 Jun 14. PMID: 34127810; PMCID: PMC8405644.

[3] Gurcan MN, Boucheron LE, Can A, Madabhushi A, Rajpoot NM, Yener B. Histopathological image analysis: a review. IEEE Rev Biomed Eng. 2009;2:147-71. doi: 10.1109/RBME.2009.2034865. Epub 2009 Oct 30. PMID: 20671804; PMCID: PMC2910932.

[4] Priya C V L, V G B, B R V, Ramachandran S. Deep learning approaches for breast cancer detection in histopathology images: A review. Cancer Biomark. 2024;40(1):1-25. doi: 10.3233/CBM-230251. PMID: 38517775; PMCID: PMC11191493.

[5] Spanhol FA, Oliveira LS, Petitjean C, Heutte L. A Dataset for Breast Cancer Histopathological Image Classification. IEEE Trans Biomed Eng. 2016 Jul;63(7):1455-62. doi: 10.1109/TBME.2015.2496264. Epub 2015 Oct 30. PMID: 26540668. (BreakHis dataset)

[6] Xie J, Liu R, Luttrell J 4th, Zhang C. Deep Learning Based Analysis of Histopathological Images of Breast Cancer. Front Genet. 2019 Feb 19;10:80. doi: 10.3389/fgene.2019.00080. PMID: 30838023; PMCID: PMC6390493.

[7] Araújo T, Aresta G, Castro E, Rouco J, Aguiar P, et al. (2017) Classification of breast cancer histology images using Convolutional Neural Networks. PLOS ONE 12(6): e0177544. https://doi.org/10.1371/journal.pone.0177544

[8] Mahmoud, H.. (2022). Modern architectures convolutional neural networks in human activity recognition. Advances in Computing and Engineering. 2. 1. 10.21622/ace.2022.02.1.001.

[9] Kolobaev, Maxim & Prytyka, Daniil & Mikhailova, Daria & Voevodina, Elena & Yurchenko, Alexander. (2023). MODERN ARCHITECTURES OF CONVOLUTIONAL NEURAL NETWORKS USED FOR IMAGE CLASSIFICATION TASKS. SOFT MEASUREMENTS AND COMPUTING. 8. 69-78. 10.36871/2618-9976.2023.08.005.

[10] Zheng, W. (2024). Comparison of transfer-learning for lightweight pre-trained model on image classification. *Applied and Computational Engineering*, *53*(1), 56–63. https://doi.org/10.54254/2755-2721/53/20241244

[11] Selveraju et al. (2017) Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. https://arxiv.org/abs/1610.02391

[12] BreakHis dataset download: https://web.inf.ufpr.br/vri/databases/breast-cancer-histopathological-database-breakhis/

[13] Alom, Md. Zahangir & Yakopcic, Chris & Taha, Tarek & Asari, Vijayan. (2018). Breast Cancer Classification from Histopathological Images with Inception Recurrent Residual Convolutional Neural Network. 10.48550/arXiv.1811.04241.

[14] Sayın, İ., Soydaş, M. A., Mert, Y. E., Yarkataş, A., Ergun, B., Sözen Yeh, S., & Üvet, H. (2023). Comparative Analysis of Deep Learning Architectures for Breast Cancer Diagnosis Using the BreaKHis Dataset. arXiv preprint arXiv:2309.01007. Retrieved from https://arxiv.org/abs/2309.01007

[15] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[16] https://medium.com/data-science-collective/implementing-cnn-in-pytorch-testing-on-mnist-99-26-test-accuracy-5c63876c6ac8

[17] Aflak, O. [The Independent Code]. (2023, March 4). *Creating Neural Networks from Scratch in Python | Deep Learning with Numpy* [Video]. YouTube. https://www.youtube.com/watch?v=Lakz2MoHy6o. Github. https://github.com/TheIndependentCode/Neural-Network

[18] Bottou, L., & Bousquet, O. (2012). The tradeoffs of large-scale learning. In S. Sra, S. Nowozin, & S. J. Wright (Eds.), *Optimization for machine learning* (pp. 351–368). MIT Press.

[19] CS231n Convolutional Neural Networks for Visual Recognition, https://cs231n.github.io/neural-networks-1/#actfun

[20] He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification*. In Proceedings of the IEEE International Conference on Computer Vision (ICCV) (pp. 1026–1034).

[21] Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. *International Conference on Artificial Intelligence and Statistics*, *15*, 315–323. http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

[22] Alessandro Saviolo. CNN-from-Scratch. https://github.com/AlessandroSaviolo/CNN-from-Scratch

[23] Masters, D., & Luschi, C. (2018). Revisiting Small Batch Training for Deep Neural Networks. *arXiv: Learning*. http://export.arxiv.org/pdf/1804.07612

[24] Jason Brownlee. (2021). Code Adam Optimization Algorithm From Scratch https://machinelearningmastery.com/adam-optimization-from-scratch/

[25] pickle python library. Github. https://github.com/python/cpython/blob/3.13/Lib/pickle.py Documentation. https://docs.python.org/3/library/pickle.html

[26] Géron, A. (2019). Min–max normalization (usually called scaling to a range). In Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (1st ed.). O'Reilly Media. Retrieved April 28, 2025, from https://www.oreilly.com/library/view/hands-on-machine-learning/9781788393485/fd5b8a44-e9d3-4c19-bebb-c2fa5a5ebfee.xhtml

[27] Cancer Research UK. Breast cancer statistics: Breast cancer incidence (invasive) (2017–2019) [Internet]. London: Cancer Research UK. Available from: https://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/breast-cancer

[28] Elmore JG, Longton GM, Carney PA, Geller BM, Onega T, Tosteson AN, Nelson HD, Pepe MS, Allison KH, Schnitt SJ, O'Malley FP, Weaver DL. Diagnostic concordance among pathologists interpreting breast biopsy specimens. JAMA. 2015 Mar 17;313(11):1122-32. doi: 10.1001/jama.2015.1405. PMID: 25781441; PMCID: PMC4516388.

[29] Rakhlin, Alexander & Shvets, Alexey & Iglovikov, Vladimir & Kalinin, Alexandr. (2018). Deep Convolutional Neural Networks for Breast Cancer Histology Image Analysis. 10.1101/259911.

[30] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. http://www.deeplearningbook.org

[31] Benhammou, Yassir & Tabik, Siham & Boujemâa, Achchab & Herrera, Francisco. (2018). A first study exploring the performance of the state-of-the art CNN model in the problem of breast cancer. 1-6. 10.1145/3230905.3230940.

[32] Abadi et al. (2016). TensorFlow: A system for large-scale machine learning. OSDI '16.

[33] NVIDIA Developer Docs – cuDNN. https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html

[34] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2

[35] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16, 321–357. https://doi.org/10.1613/jair.953

[36] Y. Xu, T. Mo, Q. Feng, P. Zhong, M. Lai and E. I. -C. Chang, "Deep learning of feature representation with multiple instance learning for medical image analysis," 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Florence, Italy, 2014, pp. 1626-1630, doi: 10.1109/ICASSP.2014.6853873.

[37] Frid-Adar, M., Klang, E., Amitai, M., Goldberger, J., & Greenspan, H. (2018). *Synthetic data augmentation using GAN for improved liver lesion classification*. Neurocomputing, 321, 321–331. https://doi.org/10.1016/j.neucom.2018.09.013

Appendix A

Self-appraisal

This project was a very tough but rewarding learning experience for me, in both technical development and independent research. Creating a convolutional neural network from scratch using only NumPy presented significant challenges, namely, how to structure my project (i.e., making new classes, which layers to include or implement, which class to put into which module etc.), which architecture to use, and the lack of GPU support for NumPy and high computational cost. These limitations pushed me to research and make careful architectural and optimization decisions, such as implementing aggressive down-sampling early in the architecture and applying early stopping to halt training if validation accuracy was not improving, because not doing so would result in my model taking too long and being infeasible to train. While this restricted the model's final performance compared to the large pre-trained model's benchmarks, it provided valuable insights into model efficiency, batch processing, and the balance between accuracy and resource constraints.

Early in the development phase, I tested the model on a simpler dataset (MNIST), which taught me the importance of starting small. It made debugging and validating architectural choices far more manageable, especially compared to jumping straight into complex medical data. With more advanced datasets, it becomes harder to tell if the model is learning anything meaningful or just memorizing patterns. This step-by-step approach saved time and helped me gain confidence in each layer of the architecture. I also came to really value structured version control using GitHub. There were several moments when the codebase became too messy or unstable, and being able to git checkout a previous branch or commit saved me from losing hours of progress. It shows just how essential good version control is, not just for working in a group, but also for your own sanity during development. However, I acknowledged that the lack of reproducibility scripts, fixed random seeds, and checkpointed models limits the ability of others to replicate my results precisely; this is an area I would improve in future work.

One area I feel I underperformed was managing the project timelines. Because of the computational limitations, each training iteration was time-consuming, which led to a bottleneck in experimenting with more complex architectures or different variations of hyperparameters. In hindsight, I should have planned for this earlier and…

Legal, Social, Ethical, and Professional Considerations

Although this project was purely academic, it simulates a scenario where AI is used in high-risk cancer diagnosis applications. This raises ethical questions, such as a false negative result, where the model classifies a malignant tumor as benign, which could have major implications on the patient's life. Therefore, achieving a high recall on malignant cases (>90%) was a primary goal, even at the cost of increased false positives (it is better to be safe than sorry).

Another issue is model transparency and trust. In clinical settings, doctors must understand why a model made a decision and be able to verify it. While I included feature map visualizations, the absence of Grad-CAM is a limitation. Future versions must include such features for transparency and trust.

From a social and professional viewpoint, my model was designed with deployment in mind, especially for use in low-resource environments where hospitals may not have access to GPUs. This influenced my focus on keeping the model lightweight and fast.

Legally, while I used an open-access dataset (BreakHis), it's important to note that any real-world deployment would require compliance with data protection regulations such as GDPR or HIPAA to protect the patient's personal information from harmful third parties. This project does not currently address privacy, security, or clinical validation protocols, and would require substantial work before being suitable for deployment.

Appendix B

External Materials

The following materials were used as part of the project. These were references or inspirations for implementation but were not used directly in the final model code. All CNN architecture, training, data processing, and evaluation scripts were developed independently as part of this project unless stated otherwise.

Project Structure References:

- **Alessandro Saviolo – CNN-from-Scratch GitHub repository.**
  Used to understand project structuring and module organization in NumPy-based CNNs.
  GitHub: https://github.com/AlessandroSaviolo/CNN-from-Scratch
- **Aflak, O. (The Independent Code) – Creating Neural Networks from Scratch in Python (YouTube).**
  Referenced for initial guidance on CNN layer implementation (convolution, pooling,
  Activation functions) using NumPy. Also used to understand how CNNs work. And batched inputs.
  GitHub: https://github.com/TheIndependentCode/Neural-Network
  Video: https://www.youtube.com/watch?v=Lakz2MoHy6o&ab_channel=TheIndependentCode

Adam Optimizer:

- **Jason Brownlee (2021)—Adam Optimization Algorithm from Scratch**
  Link: https://machinelearningmastery.com/adam-optimization-from-scratch/

Batch Normalization Layer:

- **Huybik – CNN-from-Scratch GitHub repository.**
  GitHub: https://github.com/huybik/CNN-from-scratch

Appendix C

GitHub Repository Guide

The code of the project is available on GitHub, in the repository [0]. Different versions of the project are also available in the various branches of the repository; for instance, the SGD optimized code for the MNIST dataset is available under the branch “MNIST-SGD”. The project followed good version control, as every new version was pushed to the repository.