NEURAL NETWORK BASED DIAGNOSIS OF BREAST CANCER USING THE BREAKHIS DATASET

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Ross Dalke

June 2022

COMMITTEE MEMBERSHIP

TITLE:	Neural Network Based Diagnosis of Breast Cancer Using the BreakHis Dataset
AUTHOR:	Ross Dalke
DATE SUBMITTED:	June 2022
COMMITTEE CHAIR:	Jane Zhang, Ph.D. Professor of Electrical Engineering
COMMITTEE MEMBER:	Xiao-Hua (Helen) Yu, Ph.D. Professor of Electrical Engineering
COMMITTEE MEMBER:	Wayne Pilkington, Ph.D. Professor of Electrical Engineering

ABSTRACT

Neural Network Based Diagnosis of Breast Cancer Using the BreakHis Dataset

Ross Dalke

Breast cancer is the most common type of cancer in the world, and it is the second deadliest cancer for females. In the fight against breast cancer, early detection plays a large role in saving people's lives. In this work, an image classifier is designed to diagnose breast tumors as benign or malignant. The classifier is designed with a neural network and trained on the BreakHis dataset. After creating the initial design, a variety of methods are used to try to improve the performance of the classifier. These methods include preprocessing, increasing the number of training epochs, changing network architecture, and data augmentation. Preprocessing includes changing image resolution and trying grayscale images rather than RGB. The tested network architectures include VGG16, ResNet50, and a custom structure. The final algorithm creates 50 classifier models and keeps the best one. Classifier designs are primarily judged on the classification accuracies of their best model and their median model. Designs are also judged on how consistently they produce their highest performing models. The final classifier design has a median accuracy of 93.62% and best accuracy of 96.35%. Of the 50 models generated, 46 of them performed with over 85% accuracy. The final classifier design is compared to the works of two groups of researchers who created similar classifiers for the same dataset. This will show that the classifier performs at the same level or better than the classifiers designed by other researchers. The classifier achieves similar performance to the classifier made by the first group of researchers and performs better than the classifier from the second. Finally, the learned lessons and future steps are discussed.

Keywords: Breast cancer, BreakHis dataset, Histopathological images, Convolutional neural network, Deep learning, Medical imaging, Breast cancer recognition

ACKNOWLEDGMENTS

For providing education, guidance, and expertise, I am greatly appreciative of my thesis advisor, Dr. Jane Zhang. I would also like to extend my thanks to my thesis committee for their time and feedback on my project. I am grateful for the knowledge that they have passed down to me and for always being pleasant to work with.

I would also like to thank my family for being my main support network. Specifically, I am deeply thankful for my brother, Trevor, for his unbreakable positivity and professional insight; he has always been my biggest supporter. My parents, Trevor, and Seif and his wife have been there for me every step of the way.

Thanks should also go to my partner, Maia, for their encouragement and emotional support. Lastly, I would like to acknowledge my friends and classmates for helping me in this endeavor.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

LIST OF CODE BLOCKS

Code Block Page

CHAPTER 1. INTRODUCTION

1.1 Problem Statement

Breast cancer is the most common type of cancer globally, accounting for 12% of newly identified cancers each year. Of those cases, 85% of patients have no family history of breast cancer. In the U.S., around 1 in 8 females and 1 in 833 males will develop invasive breast cancer over their lifetime. While technology has improved our ability to detect and diagnose breast cancer more effectively, it is still killing a tragic number of people. For females in the US, breast cancer has the second highest death rate of all other cancers; it is projected that breast cancer will kill over 43,000 of them in 2022 [1].

Humans have known about breast cancer and have been trying to treat it since at least 3,000 to 2,500 B.C.E. Back then, breast cancer was typically diagnosed through surgery or post-mortem autopsy. It was not until the mid 19th century that more modern methods of breast cancer detection started forming. In 1847, the invention of the microscope led to advances in the field of histopathology: the study of tissues related to disease [2]. Microscopy allowed for a more detailed look at the cells within different types of tumors, making it possible to diagnose them more accurately. Later, in 1895, the first X-ray was taken, which would eventually lead to the use of mammograms for a non-invasive method for imaging cancer. These technological advances made it much more possible to detect cancer in earlier stages and study its development.

There are now many modalities used for breast cancer imaging and detection. They include, mammography, ultrasound, thermography, MRI, positron emission tomography (PET) scans, CT scans, excisional biopsy, etc. While many of these methods are very good at identifying where a tumor exists, they are not as useful when it comes to making a diagnosis of how dangerous the tumor is. In the field of histopathology, samples from excisional biopsies are used to take microscopic images of tumor cross sections for diagnosis. Since the microscopic images are taken directly from a tumor and provide a lot of detail, they are extremely useful for differentiating types of tumors and their severity.

Though there have been many advancements in the detection, diagnosis and treatment of breast cancer, it is still the second deadliest cancer for females. Fortunately, these advancements have made it so that deaths from breast cancer have decreased by 1% per year from 2013 to 2018 [1]. Early detection and diagnosis of breast cancer is crucial for the survival of patients. When breast cancer is detected and properly diagnosed at an early stage, the patient's five year relative survival rate is 93% [3]. With evidence that a proper diagnosis can drastically reduce the chance of death due to breast cancer, it is important that diagnosis algorithms, such as image classifiers, continue to improve their accuracy so that the death rates keep dropping. If early detection and diagnosis technologies continue to improve, dying from breast cancer could become a thing of the past. This paper discusses the design of an image classifier that aims to improve the accuracy of early diagnosis of breast cancer so that more lives can be saved.

1.2 Purpose of Study

The purpose of this work is to develop an image classifier that is capable of diagnosing breast tumors as benign or malignant. The classifier uses labeled histopathological images taken from excisional biopsies because they are the most informative image type for classifying tissues. The Breast Cancer Histopathological Image Classification (BreakHis) dataset [4] was designed exactly with this purpose in mind, so it is used for this project. The dataset was created in 2016, so it is fairly recent.

Since there is an abundance of data in the BreakHis dataset, the classifier is designed with convolutional neural networks. The project focuses on improving the initial classifier design so that it achieves a higher classification accuracy. This involves studying how image resolution and changing images to grayscale affect the classification accuracy. This work also studies how the number of training epochs influences the performance of the final model. Several neural network architectures including VGG16, ResNet50 and custom architectures are compared to see which structure improves the classifier. Finally, a data augmentation method is investigated to see if it can improve the classification accuracy.

The project does not focus on image segmentation or tumor detection. Segmentation is typically performed to extract the location of a tumor from an image. Since the images are tissue samples taken directly from a tumor, the segmentation step is not necessary.

The final classifier is compared to the results of two other groups of researchers that created similar classifiers using the same dataset. The learned lessons and next steps are discussed in the final chapters of this report. It is hoped that this study will provide information to other researchers exploring neural network structures or working with the BreakHis dataset. Specifically, it should help them determine the best way to develop an algorithm for breast cancer classification. Ideally, this work will contribute to the next 1% decrease in breast cancer's death rate.

CHAPTER 2. BACKGROUND

2.1 Breast Cancer

Tumors have two main classifications: benign or malignant. Benign tumors are characterized by distinct, smooth, regular borders. They do not spread to other areas of the body, so they are considered to be safe when found in breast tissue. Malignant tumors tend to have irregular borders and grow faster than benign tumors, sometimes spreading to other parts of the body and invading nearby tissue [5].

The BreakHis dataset (described further in **Chapter 2.2**) has eight subclassifications to describe more specific tumor diagnoses. Benign tumors include adenosis, fibroadenoma, phyllodes tumor and tubular adenoma. Malignant tumors (breast cancer) include ductal carcinoma, invasive lobular carcinoma, mucinous carcinoma, and papillary carcinoma.

Benign tumors are shown in **Figure 2.1.1**. Adenosis (**Figure 2.1.1 a**) is when the lobules (milk-producing glands) become enlarged and more plentiful. This is a benign condition that has low likelihood of becoming malignant [6]. Fibroadenomas (**Figure 2.1.1 b**) are benign tumors that consist of glandular and stromal (connective) tissues. These benign tumors are more common than the other benign cases [7]. Phyllodes tumors (**Figure 2.1.1 c**) are typically benign, but about 1 in 4 cases will be malignant [8]. Since malignant phyllodes tumors are made within the connective tissues of the breast, rather than the milk ducts, they are classified as a cancer of connective tissues (sarcoma) instead of breast cancer [9]. Since phyllodes tumors are very rare, most likely benign, and less deadly than breast cancer, they are classified as benign in this dataset. The last type of benign tumor in the dataset is tubular adenoma or pure adenoma (**Figure 2.1.1 d**). These are rare tumors that are characterized to have microscopic tubular structures that don’t vary much in size [10].

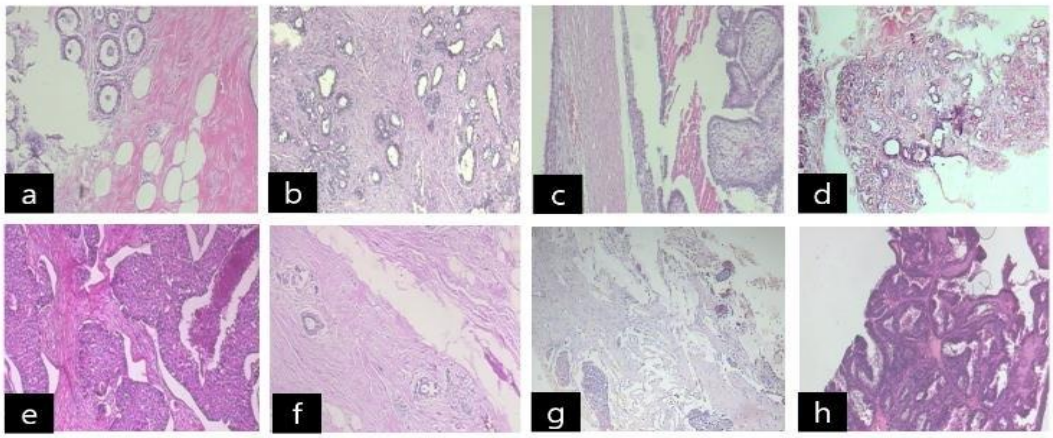


Figure 2.1.1: Benign Samples from BreakHis dataset: **a)** Adenosis, **b)** Fibroadenomas, **c)** Phyllodes Tumor, **d)** Tubular Adenoma

Malignant tumors are shown in **Figure 2.1.2**. Ductal carcinoma (**Figure 2.1.2 e**) is the most common type of breast cancer; about 80% of invasive breast cancers in females and 90% in males are invasive carcinoma. This type of breast cancer starts in the cells that line the milk ducts [11]. The next most common type of breast cancer is lobular carcinoma (**Figure 2.1.2 f**), which is cancer that has leaked through the lobule and invaded the tissues of the breast. This accounts for about 10% of breast cancers [12]. Mucinous Carcinoma (**Figure 2.1.2 g**) is a rare subcategory of invasive ductal carcinoma. The tumor is a cluster of abnormal cells that seem to 'float' in pools of mucin [13]. Papillary Carcinoma (**Figure 2.1.2 h**) is the most rare type of invasive ductal breast cancer: accounting for fewer than 1% of all breast cancers. This carcinoma is characterized by "finger-like projections, or papules," that can be seen under a microscope [14].

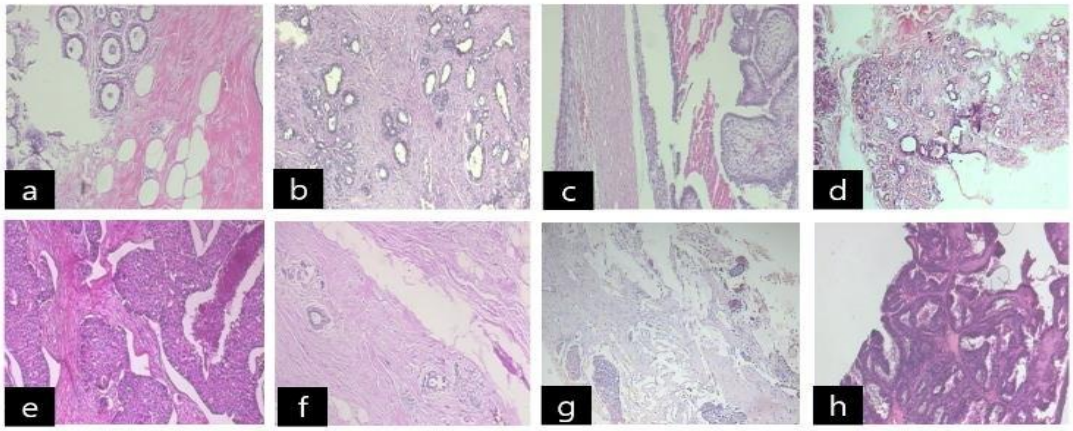


Figure 2.1.2: Malignant Samples from BreakHis dataset: **e)** Ductal Carcinoma, **f)** Lobular Carcinoma, **g)** Mucinous Carcinoma, **h)** Papillary Carcinoma

Breast cancer is common and very dangerous. As of 2021, breast cancer was the most common type of cancer globally, accounting for 12% of newly identified cancers each year. Of those cases, 85% of them have no family history of breast cancer. In the U.S., around 1 in 8 females and 1 in 833 males will develop invasive breast cancer over their lifetime. For females in the US, breast cancer has the second highest death rate of all other cancers; it is projected that breast cancer will kill over 43,000 of them in 2022 [1].

2.2 BreakHis Dataset Description

The Breast Cancer Histopathological Image Classification (BreakHis) dataset—built by the Pathological Anatomy and Cytopathology Laboratory in Parana, Brazil—is used for this project. Histopathology is the study of diseases of the tissues; histopathological images are microscopic images of tissues that can be examined to help diagnose tissue diseases and abnormalities [15]. For this dataset in particular, the tissue samples (like the one in **Figure 2.2.1** [16]) were taken from suspicious lumps or bumps in breast tissue and imaged with a microscope.

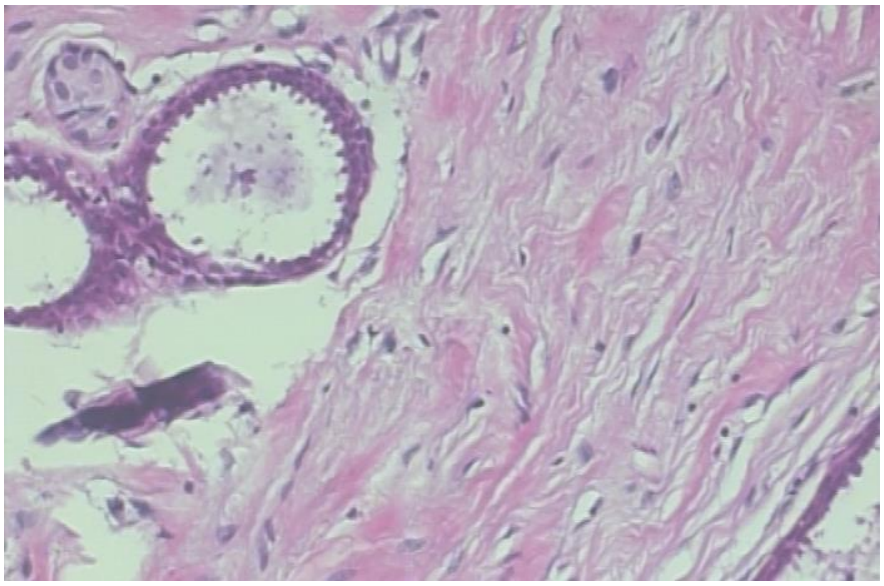


Figure 2.2.1: Histopathological Image from BreakHis Dataset [16]

The BreakHis dataset has professional annotations that label each tissue sample as 'Benign' or 'Malignant' tissue. The dataset also has eight subclassifications to describe more specific diagnoses, shown in **Figure 2.2.2**. Benign tumors include adenosis (**a**), fibroadenoma (**b**), phyllodes tumor (**c**) and tubular adenoma (**d**). Malignant tumors (breast cancer) include ductal carcinoma (**e**), invasive lobular carcinoma (**f**), mucinous carcinoma (**g**), and papillary carcinoma (**h**). These were defined in **Chapter 2.1**.

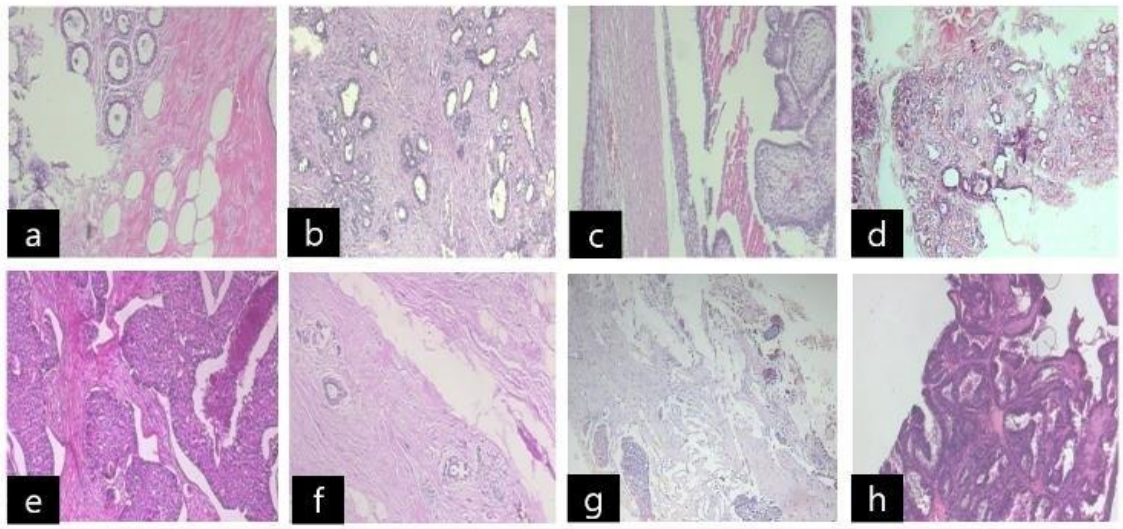


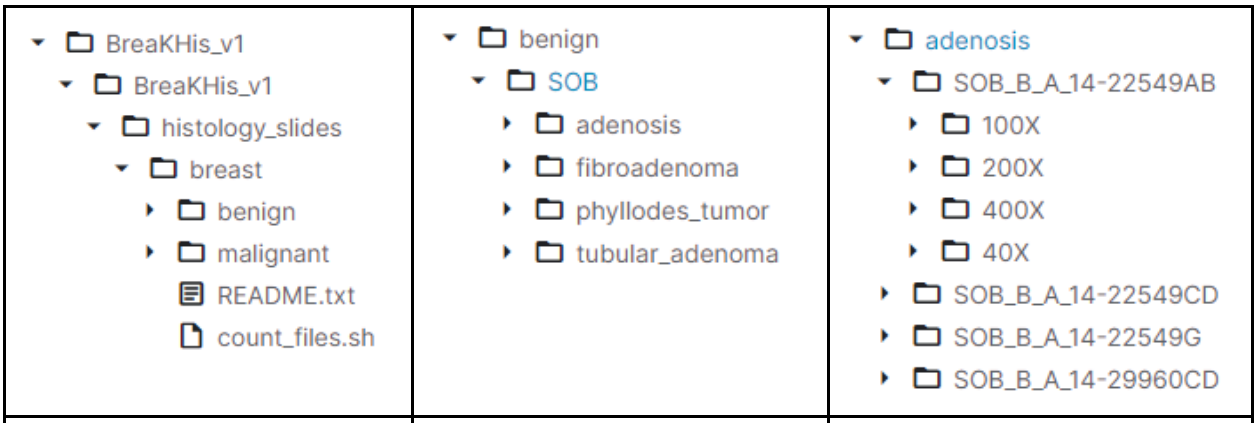
Figure 2.2.2: Examples from BreakHis dataset

The tissue samples were collected from 82 patients. Varying magnifications were used for imaging the samples, including 40X, 100X, 200X and 400X. There are 2,480 benign samples and 5,429 malignant samples for a total of 7,909 images. **Table 2.2.1** breaks down the BreakHis structure in more detail. Each image has a resolution of 700x460 pixels with 3-channel RGB and 8-bit resolution for each color channel [16].

Table 2.2.1: BreakHis 1.0 Structure [4]

Magnification	Benign	Malignant	Total
40X	652	1,370	1,995
100X	644	1,437	2,081
200X	623	1,390	2,013
400X	588	1,232	1,820
Total of Images	2,480	5,429	7,909

The main folder structure [16] for the BreakHis Dataset is shown in **Figure 2.2.3a**. Under each main classification folder (benign or malignant), the sub folders are split into the corresponding subclassifications. For example, the benign folder contains the samples for adenosis, fibroadenoma, phyllodes tumor, and tubular adenoma, as shown in **Figure 2.2.3b**. The SOB file is there because all samples were collected using the SOB method, also known as partial mastectomy or excisional biopsy. Each subclassification has folders for each patient, and further separates these samples into the magnification factors used for imaging. **Figure 2.2.3c** shows that the Adenosis subclass has 4 patients and separates the images into folders based on magnification factor. This effectively sorts the images by binary classification, collection method (all SOB), subclassifications, patient, and magnification.



a) Main folder structure	**b)** Subclassifications for Benign	**c)** Patient and Magnification

Figure 2.2.3: BreakHis Dataset Folder Structure

2.3 Limitations of the BreakHis Dataset

One downside with this dataset is that it uses histopathological images. This means that any classifiers built using this data will require biopsies to test patients. Since taking these images involves scraping tissues from lumps in or on breasts, the procedure can be very invasive. Other imaging techniques that can be used for breast cancer are far less invasive. For example, ultrasounds, mammograms and MRI don't require tissue samples to be removed from the patient [17].

Another downside is that it requires prior knowledge of where the lump is in the breast. If the lump is small or deep in the tissue, it may be more difficult to find. It is likely that another form of imaging, such as an x-ray, will be needed to find the correct location of the tumor before taking the sample. In that case, it would be convenient if the classifier worked with the same imaging type that is used to find the lump.

The dataset is also imbalanced [18], meaning that there are differing numbers of samples belonging to each class. Benign samples are only 31.36% of the total dataset whereas malignant samples make up the remaining 68.64%. A classifier trained with this data will be biased towards predicting a malignant label because of this imbalance. This may be desirable, depending on if it is

worse to return false positives or false negatives. However, this makes it harder to implement a classifier with no bias.

2.4 Advantages of the BreakHis Dataset

Since histopathological images are taken directly from areas of interest, it isn't necessary to perform segmentation and feature extraction before classification. Typically, segmentation is performed to detect where the potential tumor is, but this is not necessary for this dataset. Since segmentation does not need to be performed, it makes designing the classifier much simpler. Allowing for a smaller scope makes it so the project can delve further into the classifier design. Using histopathological images also makes it so that all of the data in the image is used for classification instead of just the segmented section. **Figure 2.4.1**, shows that the lump in a mammogram is a very small portion of the image; a classifier would not have as much data to work with if it used a mammogram. Using histopathological images makes it so that the classification process is simpler and more data can be used for making classification decisions.

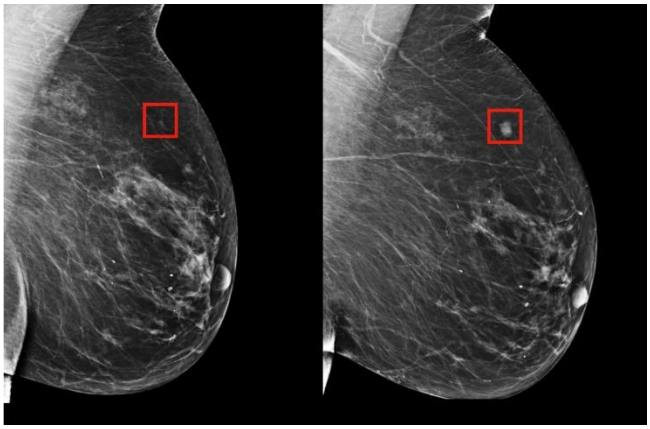


Figure 2.4.1: Example of a Mammogram with a tumor [19]

2.5 Comparing Deep Learning and Traditional Computer Vision

In "Deep Learning vs. Traditional Computer Vision," by Niall O' Mahony et al. [20], the authors explore the advantages and disadvantages of deep learning (DL) compared to traditional computer vision (CV) techniques. The paper also defines terms related to deep learning, problems that are solved by deep learning, and problems not suited for deep learning

Deep learning is a type of machine learning that heavily relies on artificial neural networks (ANNs), which were inspired by the neurons in the human brain. Each neuron is a computing cell that performs a simple operation and sends information to surrounding neurons. Through these interactions, the neural network can make decisions. By adding more layers of neurons, more intricate connections can be made, allowing for complex decisions to be made. In image processing, deep learning is used to solve difficult problems like image colorization, classification, segmentation, and detection.

Using deep learning algorithms to solve these issues has made it so that better results can be achieved. Another advantage is that the neural networks and models are retrainable for use in other datasets. Traditional computer vision techniques are less flexible, so they need more work from the engineers in order to work with another dataset. For example, assume that there is a model that successfully classifies images as a handwritten "0" or "1" and it needs to be adapted so that the algorithm classifies images as either "cat" or "dog." With a traditional algorithm, this adaptation would require an engineer to perform feature analysis on the images of cats and dogs to figure out which features to use. The program would then have to be changed to extract these features. With a deep learning algorithm, the dataset could be swapped out and the model retrained. There would be no extra steps like feature analysis for the engineer. When there are more classes involved, feature analysis becomes more burdensome, so it makes the classical algorithms harder to develop.

Mahony et al. provides **Figure 2.5.1** to show the difference between traditional computer vision workflow and deep learning workflow. Since the deep learning algorithm figures out the underlying patterns of features that are useful for classification, it removes the manual feature extraction that has to be done for traditional methods. For a deep learning model to work with new data, the model just has to be retrained with the new dataset.

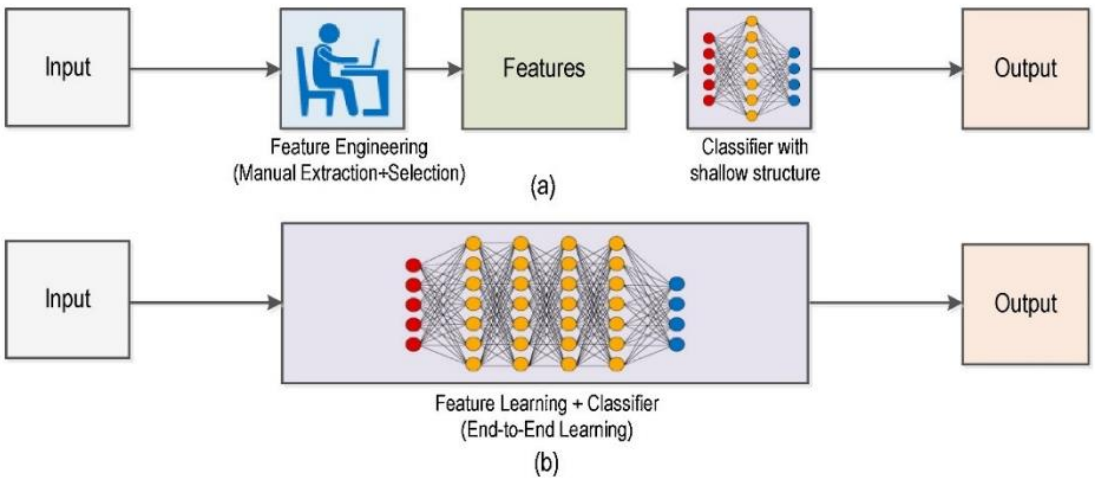


Figure 2.5.1: (**a**) Traditional Computer Vision workflow vs. (**b**) Deep Learning workflow [20]

While deep learning has many benefits, it can sometimes be overkill for certain projects. For example, if a classification problem can be solved by analyzing just a few features, it does not require a computationally intense algorithm to solve the issue. Traditional methods are capable of solving complicated problems in fewer lines of code, less cost, and much more efficiently than deep learning. Deep learning is considered very computationally intense and it requires a lot of time for training. It is often considered the brute force solution to computer vision because of its lack of elegance when compared to traditional methods. Deep learning algorithms are only viable when there is a need to analyze very large amounts of data in order to make a difficult decision.

Another issue with deep learning is that it is hard to tell how it came to a decision. Unlike traditional techniques, there isn't a way to know what features are being considered. It is also extremely difficult to manually tweak parameters because of the complex inter-relationships between hundreds or thousands of nodes. The black-box nature of deep learning makes it difficult to tweak performance, which is an issue for trying to fix issues like over-fitting. It is much easier to troubleshoot a traditional model because it has full transparency.

Mahony et al. concludes that deep learning has made a lot of traditional methods of computer vision irrelevant, but some traditional techniques are still useful for when deep learning is considered overkill. The authors also note that knowing only deep learning methods will greatly

restrain the kinds of solutions that can be made for solving computer vision problems. Whether or not deep learning is applied comes down to how complicated the decision is, how much computing power and time is available, and how much data is available for training.

2.6 Conclusions from Background

Even though breast cancer mostly only affects females, it is the most common type of cancer. For females, it has the second highest death rate of all cancers. Studies have shown that improvements in early detection have been able to decrease the death rate even as cases of breast cancer increase.

In order to decrease the number of deaths due to breast cancer, it is important that computer vision algorithms are made to diagnose breast cancer. After reading the paper by Niall O' Mahony et al., deep learning should be used for this project for several reasons. The BreakHis dataset has over 7,909 samples, so there is a large amount of data available for training the neural network. The diagnosis of cancer is complex: it's easier to make a neural network do the feature mapping and classification than it would be to design a traditional computer vision algorithm. With access to a remote server, there is easy access to large amounts of computing power, which is needed for deep learning. The project was performed over the course of an academic year, so there was not much of a time constraint. Most importantly, the higher performance achieved by the deep learning models is preferred for a diagnosis that could be the difference between life and death. The flexibility of neural network structures is another important factor, as this will allow others to build off of the results of this project, even if they are using a different dataset.

CHAPTER 3. LITERATURE REVIEW

3.1 Exploring the Effects of Neural Network Structures on Classifier Performance

In "Breast Cancer Classification from Histopathological Images with Inception Recurrent Residual Convolutional Neural Network," by Md Zahangir Alom et al. [21], the authors test their neural network structure against other deep learning structures. The BreakHis dataset is one of two histopathological datasets that are being used to test the performance of the classifier. In the paper, the authors describe their neural network, data augmentations, and results. They also compare testing accuracy differences for multi-class and binary classification.

The authors' goal was to create a deep convolutional neural network (DCNN) that combines the strengths of the Inception Network (Inception-v4), the Residual Network (ResNet), and the Recurrent Convolutional Neural Network (RCNN). The resulting network is called the Inception Recurrent Residual Convolutional Neural Network (IRRCNN) model.

Using the BreakHis dataset, they split the data into training, validation and testing datasets. For the training dataset, the samples were used to create more training samples. They did this by creating 21 copies of each image and applying augmentations like rotation, shift, shear, zooming, and flipping to each image to create an augmented training dataset that was 22 times as large as the original training dataset. Noise was also added in some parts of the images to help reduce the likelihood of overfitting the data. **Figure 3.1.1** shows examples of the augmented images compared to the original images.

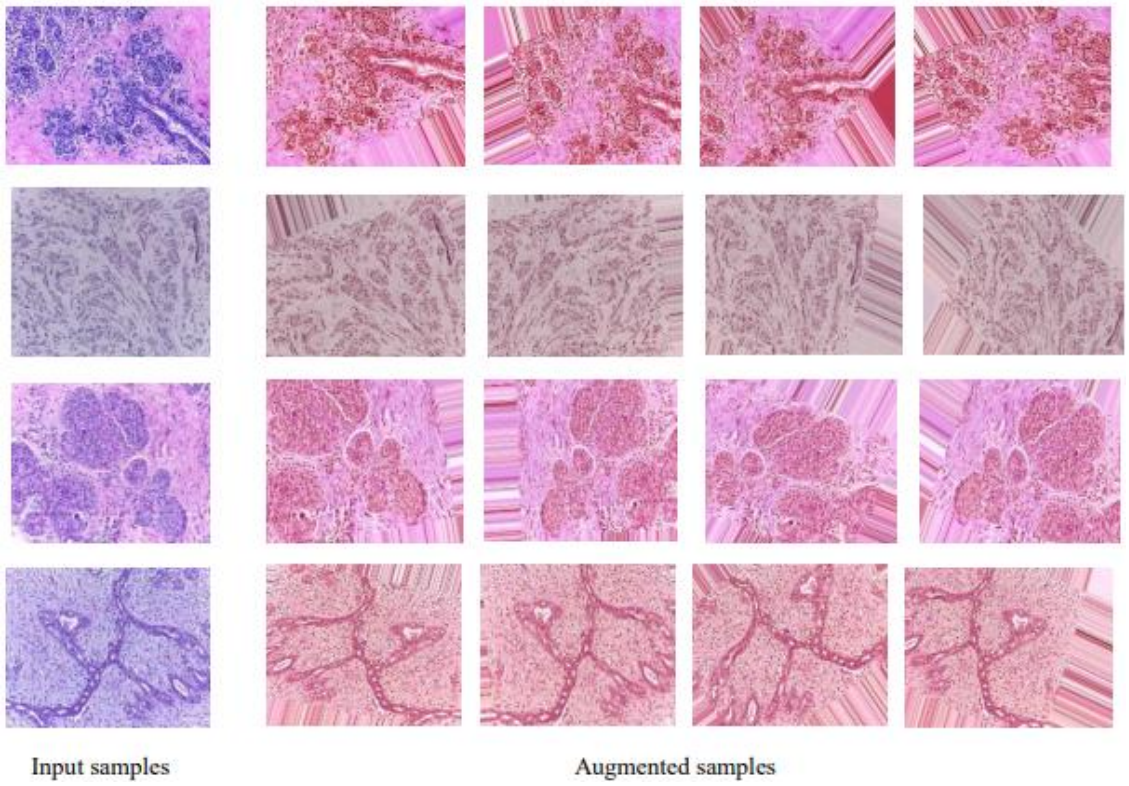


Figure 3.1.1: Four Example Images with Corresponding Augmented Images [21]

The validation dataset was used for an unbiased evaluation of how well the model performed as it was being trained. Typically, the entire training set is passed through the model in a fitting stage, this is defined as 1 epoch. The validation set is then passed through the model for an unbiased test of how well the model is currently classifying data. As more epochs are run, the classification rates for the training data and validation data should both increase, showing that the model is improving. However, if the validation data stops improving, it could mean that the model is overfitting the training data. Overfitting the model would make it so that the classifier does not classify new data as well, decreasing the performance. It is also possible that the model converges to a classification rate, so training over more epochs will not improve the results and waste time. For this paper, the authors showed the testing and validation accuracies over 150 epochs. **Figure 3.1.2** shows that validation and testing accuracies eventually converged, but there was an unexpected degradation to the accuracy around 50 epochs.

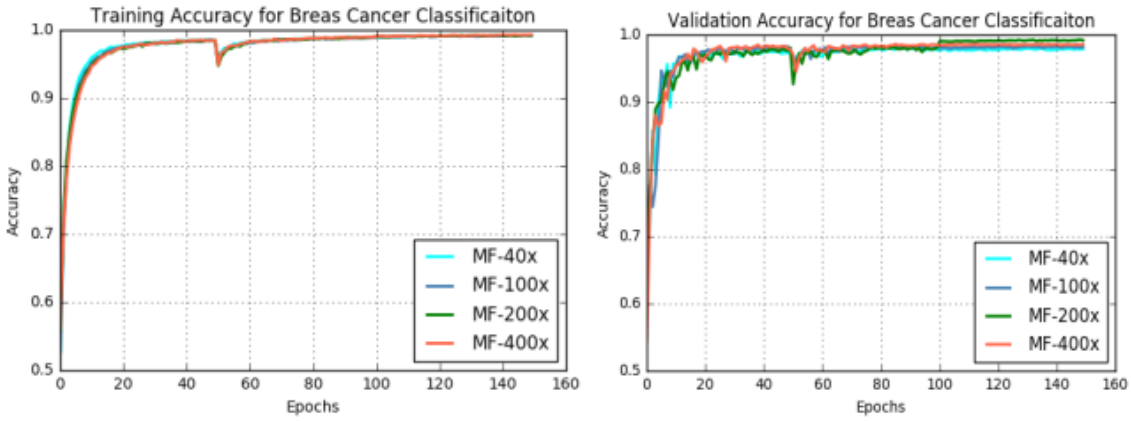


Figure 3.1.2: Training and Validation Accuracy for Multi-class IRRCNN Classifier at Different Magnification Factors [21]

After training the model using varying structures, the test dataset was used to get the performance for each magnification. The IRRCNN model was also trained without the augmented data to show the improvements made by adding more data. To make a final diagnosis for a patient, the diagnosis for each of the patient's samples was taken into account. By diagnosing patients using all of the samples taken from them, the patient-level accuracy was calculated. The multi-class classifier results are shown in **Table 3.1.1** and the binary classifier results are shown in **Table 3.1.2**. In both cases, the IRRCNN structure out-performed all of the other structures that it was tested against, achieving an image-level classification rate around 95 to 97% on average. The binary classifier performed slightly better than the multi-class classifier. Adding augmented data to the training set improved the results of the IRRCNN by roughly 1% to 1.5%.

Table 3.1.1: Breast Cancer Classification Results for Multi-Class [21]

	Methods	Year	Classification Rate (100R) at Magnification Factor			
			40×	100×	200×	400×
Image Level	CNN+patches [15]	2016	85.6 ± 4.8	83.5 ± 3.9	83.1 ± 1.9	80.8 ± 3.0
	LeNet + Aug [18]	2017	40.1 ± 7.1	37.5 ± 6.7	40.1 ± 3.4	38.2 ± 5.9
	AlexNet + Aug [18]	2017	70.1 ± 7.4	75.8 ± 5.4	73.6 ± 4.8	84.6 ± 1.8
	CSDCNN + Aug [18]	2017	92.8 ± 2.1	93.9 ± 1.9	93.7 ± 2.2	92.9± 2.7
	IRRCNN +w/o Aug.	2018	95.69 ± 1.18	95.37 ±1.29	95.61± 1.37	95.15 ± 1.24
	IRRCNN + w Aug.	2018	97.09 ±1.06	97.57 ±0.89	97.29 ±1.09	97.22 ±1.22
Patient Level	LeNet + Aug [18]	2017	48.2 ± 4.5	47.6 ± 7.5	45.5 ± 3.2	45.2 ± 8.2
	AlexNet + Aug [18]	2017	74.6 ± 7.1	73.8 ± 4.5	76.4 ± 7.4	79.2 ± 7.6
	CSDCNN + Aug [18]	2017	94.1 ± 2.1	93.2 ± 1.4	94.7 ± 3.6	93.5± 2.7
	IRRCNN +w/o Aug.	2018	95.81 ± 1.81	94.44 ± 1.3	95.61 ± 2.9	94.32± 2.1
	IRRCNN + Aug.	2018	96.76 ± 1.11	96.84±1.13	96.67±1.27	96.27±0.87

Table 3.1.2: Breast Cancer Classification Results for Binary Classification [21]

	Method	Year	Classification Rate at Magnification Factor			
			40×	100×	200×	400×
Image Level	CNN +fusion(sum, product, max) [15] (highest results)	2016	85.6 ± 4.8	83.5 ± 3.9	83.6 ± 1.9	80.8 ± 3.0
	AlexNet + Aug [18]	2017	85.6 ± 4.8	83.5± 3.9	83.1 ± 1.9	80.8 ± 3.0
	ASSVM [28]		94.97	93.62	94.54	94.42
	CSDCNN + Aug [18]	2017	95.80± 3.1	96.9 ± 1.9	96.7 ± 2.0	94.90 ± 2.8
	IRRCNN	**2018**	**97.16 ± 1.37**	**96.84 ±1.34**	**96.61± 1.31**	**95.78 ± 1.44**
	IRRCNN + Aug	**2018**	**97.95± 1.07**	**97.57± 1.05**	**97.32± 1.22**	**97.36± 1.02**
Patient Level	CNN +fusion (sum, product, max) [15]	2016	90.0 ± 6.7	88.4 ± 4.8	84.6 ± 4.2	86.10 ± 6.2
	Bayramoglu et al. [14]	2016	83.08 ± 2.08	83.17 ± 3.51	84.63 ± 2.72	82.10 ± 4.42
	Multi-classifier by Gupta et al. [13]	2017	87.2 ± 3.74	88.22 ± 3.23	88.89 ± 2.51	85.82 ± 3.81
	CSDCNN + Aug [18]	2017	92.8 ± 2.1	93.9 ± 1.9	93.7 ± 2.2	92.90 ± 2.7
	IRRCNN +wo aug.	2018	96.69 ±1.18	96.37 ±1.29	96.27±1 .57	96.15 ±1.61
	IRRCNN + w. Aug.	2018	**97.60± 1.17**	**97.65± 1.20**	**97.56± 1.07**	**97.62± 1.13**

For the BreakHis dataset, the authors concluded that their work achieves 1.05% and 0.55% improvement in average performance against the highest accuracies reported for image and patient level analysis in their sources. For the other dataset, they achieved 100% testing performance by using a winner takes all method to produce the final results per patient.

3.2 Binary vs Multicategory Classification and Limitations of BreakHis Dataset

In "BreakHis based breast cancer automatic diagnosis using deep learning: Taxonomy, survey and insights," by Yassir Benhammou et al. [18], the authors explored the strengths and limitations of the BreakHis dataset. They covered common issues with datasets and discussed the structure and organization of the BreakHis dataset. The authors explored the results of designing

classifiers based on magnification dependence/independence as well as multi-category/binary classification.

The dataset has two main categories (benign/malignant), eight subcategories, and images of varying magnifications (40X, 100X, 200X, 400X). This means that many different types of classifications can be made. The authors analyzed Magnification-Specific Binary (MSB), Magnification-Independent Binary (MIB), Magnification-Specific Multi-category (MSM) and Magnification-Independent Multi-category (MIM) classifications. They decided that Magnification-Independent Multi-category (MIM) classifications provide the most information to practitioners, so most of their testing used MIM classification.

According to the authors, one of the issues of the BreakHis dataset is data imbalance. Since there are more malignant samples than benign, it makes it so that the classifier is biased towards making a malignant diagnosis. There are also imbalances between all of the subcategories for both benign and malignant. Ideally, there would be an equal amount of samples taken for every subcategory, which would make the decision unbiased for choosing one class over the other.

The other issue in the BreakHis dataset is label noise. For the patient with ID 13412, some of the images contain both ductal carcinoma and lobular carcinoma. This makes it so that the image appears in both subcategories. This can confuse the model during a multi-category classification task as it tries to establish the difference between the two subcategories.

To try to solve the data imbalance, they decided to add augmented data to the training data. Ideally, more benign images would be added to the training dataset so that the data would become more balanced. The final augmentations used were random rotation (Rot), random flip (Flip), and stain normalization (SN). As shown in **Table 3.2.1**, most of the improvements were obtained by just rotating and flipping.

Table 3.2.1: The Results of Each Stage of MIB and MIM Model [18]

Model	Without DA and SN	with DA(Rot,Flip)	with DA(Rot,Flip) and SN(Macenko)	with DA(Rot,Flip) and SN(Reinhard)
Binary	78.1 ± 2.5	80.3 ± 3.0	75.0 ± 1.3	76.3 ± 0.7
	83.5 ± 1.5	**88.1 ± 2.1**	87.2 ± 1.4	**88.9 ± 2.5**
Malignant multi-category	62.3 ± 1.5	61.2 ± 2.0	60.1 ± 1.2	60.31.3
	57.4 ± 2.0	60.1 ± 1.7	56.0 ± 2.3	**63.6 ± 2.2**
Benign Multi-category	33.8 ± 1.9	38.4 ± 2.3	35.0 ± 2.4	37.2 ± 1.5
	47.6 ± 1.6	**52.7 ± 2.3**	39.0 ± 1.5	41.3 ± 0.9

From this data, the authors concluded that the dataset is not ready to be used for MIM applications. However, the MIB classifier worked well. The best way to improve the dataset would be to balance the data and add more labels so that it makes it easier to merge with other datasets. This paper also showed the importance of adding augmented data, and showed that simple augmentations such as rotation and flipping are enough to see significant improvements.

3.3 Conclusions from Literature Review

The paper by Md Zahangir Alom et al. has the greatest impact on the design of the deep learning algorithm. The results showed that binary classification performed slightly better than the multi-class classifier. This was also backed in the paper by Yassir Benhammou et al., which showed much greater performance by the binary classifier. Though binary classification loses some valuable information for making a diagnosis for the subclasses, the simpler design and higher classification accuracy are the most important factors for this project.

Alom et al. also showed that creating 21 new images using data augmentation could improve the accuracies achieved by a small amount. However, it is important to note that there is no rule of thumb for how many augmented images to create. Since increasing the size of the training set will cause longer run times, this project experiments with smaller augmented datasets. For example, augmentations increase the amount of training data by 20% instead of 2200%. This will show other researchers which approach is better, so that they have a better idea of what to do in their own designs. The data augmentation suggested by Yassir Benhammou et al. are easier to implement than the augmentation presented by Alom et al.. The simpler augmentations consist of just rotation and flipping, and achieve similar improvements in performance from just those

changes. To keep it simple, rotation and flipping are the only augmentations made to the training dataset for this project.

The results from Alom et al. also provide references for “good” classifiers; it seems that classifiers performing with upper 80’s or lower 90’s are good classifiers, whereas everything above that is a great classifier. While it is possible to create a classifier with accuracy in the upper 90’s, it probably takes very complicated structures such as the IRRCNN structures to be built. Exploring and implementing all of these structures would take a lot of time. In order to allow more time to explore more aspects of deep learning, fewer structures are used in this project. Another important lesson that Alom et al. teaches is that the number of training epochs plays a big role in the performance of the classifier. It is important to test over many epochs to achieve high accuracy, but only so long as the validation accuracy keeps improving.

CHAPTER 4. CLASSIFIER DESIGN

4.1 Acquiring and Preprocessing Data

The first step in building the classifier is downloading the dataset that is used for training and testing the model. TensorFlow Core's tutorial on loading and preprocessing images with keras [22] is used for guidance in this process. **Code Block 4.1.1** shows how the data is downloaded and the datapath is defined. When Docker (a tool used for installing the libraries and running scripts on the server) closes, it deletes any folders that aren't associated with the main folder. Unfortunately, the keras get_file method shown below will automatically save the dataset to a temporary folder outside of the main folder. So anytime that Docker is closed or the server is disconnected, it will delete the data folder. This means that everytime Docker is initially opened after logging off the server, it has to redownload the data folder. So the get_data function is written so that it will check if the temporary folder exists. If it does, then the data is already downloaded and the datapath is returned. Otherwise, the file path will be created and the BreakHis dataset will be downloaded.

```
def get_data():
    # Check if path exists
    path = "/tmp/.keras/datasets"
    if not os.path.isdir(path):

        # Create the dataset path in the Docker container
        os.makedirs(path)

        # Define Dataset url (Download Link)
        dataset_url = "http://www.inf.ufpr.br/vri/databases/BreaKHis_v1.tar.gz"


        # Download data and define directory path
        data_dir = tf.keras.utils.get_file(fname=None, origin=dataset_url, untar=True)
        data_dir = pathlib.Path(data_dir)/'histology_slides/breast/'

    # Data is already downloaded. Define Path
    else:
        print('\nData Directory Exists\n')
        data_dir = '/tmp/.keras/datasets/BreaKHis_v1/histology_slides/breast'

    return data_dir
```

Code Block 4.1.1: Downloads BreakHis Dataset

Now that data is temporarily stored on the server, the dataset is accessed with the image_dataset_from_directory method from keras, as shown in **Code Block 4.1.2**. This method creates a dataset with class labels based on the folders found under the data directory defined earlier. Since the goal is binary classification, the data directory opens the "breast" folder, which has access to the "Benign" and "Malignant" folders. This returns a TensorFlow Dataset object that has all of the images and their associated labels: 0 for a benign sample and 1 for a malignant sample.

```
# Access dataset from directory.
ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    color_mode = "rgb",
    seed=123,
    image_size=(700, 460),
    batch_size=32,
    shuffle=True)
```

Code Block 4.1.2: Returns an Object Containing the Dataset

It is important to note that this method is also responsible for preprocessing the whole dataset. For example, changing the color_mode parameter from "rgb" to "grayscale" converts all of the images from 3 channel RGB to 1 channel grayscale. This is also where all images can be rescaled. Since the images are 700x460, this is the initial size used because it maximizes the amount of data available per image. Since the data is well organized in the folder structure, it is important to shuffle the data. This makes it so that the model is more properly exposed to all classes and subclasses during training and testing. If the dataset is not shuffled, it would be detrimental to the performance of the classifier. A seed is provided so that the data is consistently shuffled every time.

According to Jason Brownlee [23], another important parameter for the dataset is the batch size, which has an effect on how the model is trained. The first step in training is using the current state of the model to make a prediction for the class label of an image. That prediction is then compared to the true label values using a loss function as an estimate of the error gradient. The error gradient is used to update the model weights to improve the model and the process is repeated. This process is also known as the stochastic gradient descent optimization algorithm. The error

gradient is a statistical estimate, so using a larger set, or batch, of training samples per estimate will represent the error gradient more accurately. With a better estimate, it is more likely that the weight adjustments made will improve the model. In practice, smaller batch sizes are used because it reduces the amount of space needed in memory. Typically, a batch size of 32 is a good default value [23]; this is also the default parameter value given by keras.

The next step in preprocessing the data is partitioning the dataset into three smaller datasets, as shown in **Code Block 4.1.3**. The training dataset is the largest of the three because it is used for training the model. The test set is used for an unbiased evaluation of the final model's performance. The validation set is used for "unbiased" evaluation at the end of each training epoch. This evaluation is also used to tune the model's hyper parameters between epochs. Hyperparameters are values that affect training, but cannot be changed during training. So with more uses, the model will slowly become slightly biased towards the validation set [24]. Aside from tuning hyper parameters, the validation set is also used for assessing if the model is overfitting the training data or if the model stops improving. A train:validate:test ratio of 80:10:10 is used.

```
def get_dataset_partitions(ds,ds_size,train_spl=0.8,aug_spl=0,val_spl=0.1,test_spl=0.1):

    # Check that spl values sum to 1.
    assert (train_spl + test_spl + val_spl) == 1

    # spl the data: 80% training, 10% Validation, 10% testing
    val_ds = ds.take(val_size)
    test_ds = ds.skip(val_size).take(val_size)
    train_ds = ds.skip(val_size).skip(val_size)

    return train_ds, val_ds, test_ds
```

Code Block 4.1.3: Partitions Dataset into Training, Validation, and Testing Dataset

4.2 Defining and Training the Model

Now that the data is split into training, validation and testing datasets, the model can be defined using keras. The initial model (shown in **Code Block 4.2.1**) comes from the TensorFlow Core tutorial [22] that shows how to train a model after creating the dataset.

```
# Example from https://www.tensorflow.org/tutorials/load_data/images
def define_model():

    '''Create model (Original)'''
    model = tf.keras.Sequential([
        tf.keras.layers.Rescaling(1./255),
        tf.keras.layers.Conv2D(32, 3, activation='relu'),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Conv2D(32, 3, activation='relu'),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Conv2D(32, 3, activation='relu'),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(2)
    ])

    '''Compile the Model'''
    model.compile(
        optimizer='adam',
        loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=['accuracy'])

    return model
```

Code Block 4.2.1: Defining and Compiling Model

The model has a layer for rescaling the magnitude of the image data to be within a range of [0, 1] instead of [0,255]. This is a way of normalizing the data so that calculations do not grow out of that range. This is followed by three convolutional blocks with a max pooling layer for each of them. The convolutional block uses 32 filters, which means that this layer outputs 32 feature maps. These feature maps are the result of convolving the 2D input image with smaller 2D kernels. Feature maps might contain information such as vertical/horizontal edges, curves, flat areas, etc., but it is more likely that the features are more abstract than these examples. The kernel size is set to 3, meaning that a 3x3 (minimum size) kernel will be used in the convolution. After defining the feature maps, they are used as the input to a max pooling layer, which simplifies the representation of the feature map. As shown in **Figure 4.2.1**, it does this by selecting the max element from the section of the image that is covered by the max pooling filter [25]. Max pooling filter size is 2x2 by default.

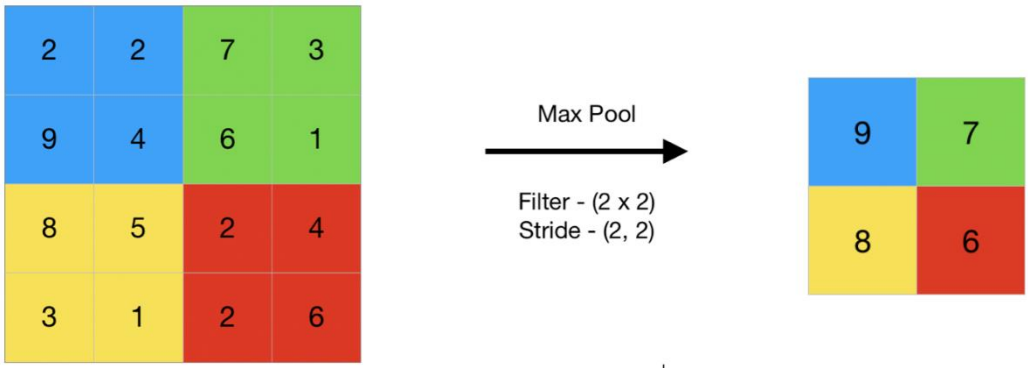


Figure 4.2.1: Max Pooling Example [25]

The simplified feature maps are then used as input for a flattening layer. The flattening layer converts the 2D feature maps into a single linear vector. The first dense (or fully connected) layer receives the vector and uses every point in the vector as input to a layer of 128 neurons. Each neuron computes a calculation with the inputs. The final dense layer uses all of the 128 nodes from the previous layer to calculate the output layer, which is only 2 nodes. The value contained by each node will be related to the probabilistic strength that the original image belonged to the class represented by that node. A similar structure with the same components is shown in **Figure 4.2.2** for reference.

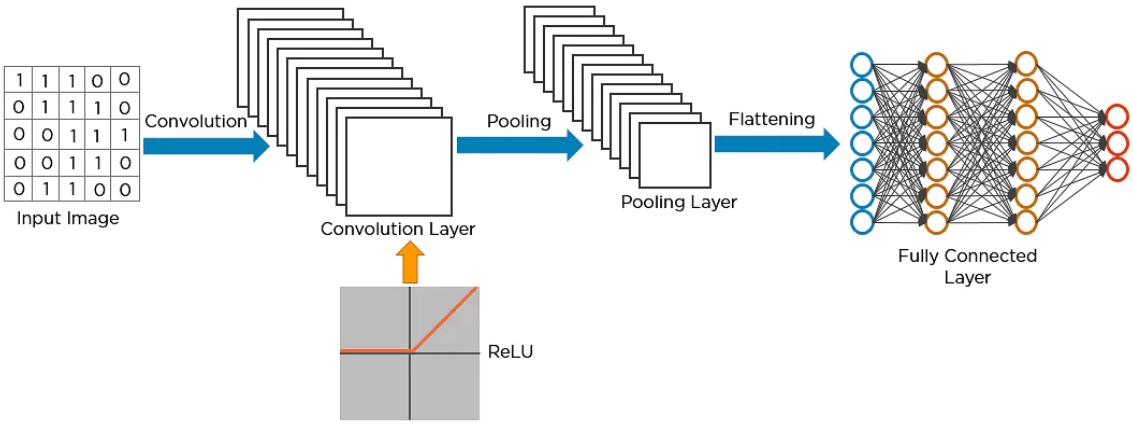


Figure 4.2.2: Visual for Convolutional, Pooling, Flattening, and Fully Connected Layers [26]

The compile method uses three parameters. The first parameter is the optimizer, which was chosen to be the Adaptive Moment Estimation (Adam) Optimizer. Adam is an optimization technique for gradient descent that combines the 'gradient descent with momentum' and Root Mean Square Propagation (RMSP) algorithms [27]. Adam combines and builds upon the strengths of

these optimizers to create an algorithm that is efficient in both time and memory. The loss function can also be specified for the compiler. The loss function is used to calculate the error gradient for gradient descent. For binary classification, cross entropy is often used for the loss function. Sparse categorical cross entropy is typically used for applications where there are multiple labels per image [28], but it can also be used in this case despite only having one label. The metrics parameter determines the metrics that are returned from the model during training and testing. For this project, the accuracy will be tracked so that the performance of the model can be observed.

Keras made it so that training the model is very simple. As shown in **Code Block 4.2.2**, the model is trained by fitting the training data to the model over several epochs. For now, the training data is run through the model three times. After each epoch, the validation set will be used to give an estimate on the performance of the classifier and tweak hyper parameters to improve the performance. The fit method returns a history of any metrics and loss values that were calculated during training and validation. This makes it so that accuracy and loss can be plotted over epochs.

```
def train(model,train_ds,val_ds):
    start = time.time()

    # Fit data to model using 3 epochs
    history = model.fit(
        train_ds,
        validation_data=val_ds,
        epochs=3
    )

    # Display the time that it took to train the model
    end = time.time()
    training_time = end-start
    print("\nTraining took:%0.2f" %training_time,"seconds\n")

    return model, history, training_time
```

Code Block 4.2.2: Training the Model

4.3 Predicting Class Labels and Assessing Predictions

Now that the model is trained, it can be used to predict the class labels for the test dataset. **Code Block 4.3.1** shows how true class labels and predicted labels are collected. Since the test

dataset is broken into batches, the test dataset has to be made into an iterable object so that the images and true labels from each batch can be accessed easily. This also allows for an exact copy of the dataset to be made, which will be useful in **Chapter 4.4**. For each batch, the true labels are collected and appended into a single array. The model is then used to predict the labels for the corresponding images.

```
def predict_labels(model,test_ds):

    # Initialize Lists
    labels = []
    pred_labels = []


    # Make the test dataset an iterable object
    test_ds = iter(test_ds)

    # For each batch, append labels to both the true label array and predicted label array
    for images, temp_labels in test_ds:

        # Collect True Data Labels
        temp_labels = tf.constant(temp_labels).numpy()
        labels.extend(temp_labels)


        # Collect Prediction Labels
        prediction = model.predict(images,batch_size=32)
        temp_pred_labels = prediction.argmax(axis=-1)
        pred_labels.extend(temp_pred_labels)

    return labels, pred_labels
```

Code Block 4.3.1: Collects True Class Labels and Predicted Labels

By comparing each predicted label against the true labels, the number of true positives, true negatives, false positives, and false negatives can be determined. The function in **Code Block 4.3.2** shows how these metrics, along with the accuracy rate, are calculated. It is possible to use keras' evaluation method to predict the labels and perform these calculations, but it is difficult to confirm that the predicted labels actually closely matched the true labels. Though not shown, tests were performed to show that the evaluation accuracy metric matches the accuracy metric calculated

below. **Figure 4.3.1** shows that the prediction labels and true labels nearly match for a test batch. This proves that the classifier is making reasonable predictions based on the images.

```
def calculate_metrics(labels, pred_labels):
    TP = 0
    TN = 0
    FP = 0
    FN = 0

    total = len(labels)
    # Count for Confusion Matrix
    for i in range(total):
        if labels[i] == 1 and pred_labels[i] == 1:
            TP += 1     # True Positive
        elif labels[i] == 0 and pred_labels[i] == 0:
            TN += 1     # True Negative
        elif labels[i] == 0 and pred_labels[i] == 1:
            FP += 1     # False Positive
        elif labels[i] == 1 and pred_labels[i] == 0:
            FN += 1     # False Negative

    # Calculate Accuracy and return results
    accuracy = (TN + TP)/total
    return [accuracy, TP/total, TN/total, FP/total, FN/total]
```

Code Block 4.3.2: Calculates Accuracy and Metrics Based on True Labels and Predicted Labels

```
True Labels
 [1 0 0 1 1 1 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 0 1 0 0 0 1 1]
1/1 [==============================] - 0s 93ms/step
Predicted Labels
 [1 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 0 0 0 1 1]
```

Figure 4.3.1: True Labels and Predicted Labels for the Same Test Batch

4.4 Stochastic Modeling

Up to this point, a model has been defined, trained, and used to predict labels for the test dataset. Performance metrics were calculated from the predicted labels. In other words, a functional model is complete.

The process of training the model has some random processes involved, so the quality of models trained with the same data will vary. In theory, once a model is trained, using the same test dataset should be deterministic: using the same model on the same test data should always return

the same labels. **Code Block 4.4.1** was used to confirm that the testing results are consistent when using the same model. A lesson learned from this experiment is that the test dataset has to be copied each iteration to be reused. Otherwise, the data would be rebatched, leading to slightly varying results.

```
# Define model
model = define_model()

# Train the Model
model, history, training_time = train(model, train_ds, val_ds)

# Test model 50 times to show that it is deterministic
results = []
test_ds_copy = iter(test_ds)
num_iterations = 50
for i in range(num_iterations):

    # Create copy of the datasets so that it can be reused
    test_ds, test_ds_copy = itertools.tee(test_ds_copy)

    # Extract True Labels from dataset and Predict Labels from images
    true_labels, predicted_labels = predict_labels(model,test_ds)

    # Calculates TP, TN, FP, FN and total accuracy
    results = calculate_metrics(true_labels, predicted_labels)
    accuracy = results[1]

    print(accuracy)
```

Code Block 4.4.1: Used to Confirm that the Testing Accuracy Is Deterministic

Making copies of the test dataset removes some of the randomness of the overall process. Another way that randomness is reduced is confirming that the datasets are consistently split from the original dataset. Back in **Chapter 4.1**, a seed was provided when shuffling the data. This makes it so that the training, validation and testing datasets always have the same images in them.

Now the only stochastic process involved in the algorithm is training. Training is non-deterministic because the dataset is shuffled after each epoch and again if retrained. Also, the initial weights for the model are randomized. The stochastic nature of the model before being trained can actually be a strength, so making the training process deterministic would not necessarily be an improvement to the classifier. For example, shuffling the training set between epochs is used to

prevent overfitting. Randomizing the initial weights for training is used to make sure that each neuron is performing its own calculation, otherwise known as breaking the symmetry [29].

Since a single classifier design has variability in the models that it produces, it is important to record the distribution, range, mean and median of the accuracy produced from multiple models. **Code Block 4.4.2** shows how 50 models are defined, trained, and tested to collect data for a single classifier. If the classification rate of a model is higher than the previous best, the model will be stored. The best performing model will be saved and returned. This makes it so that the return value of the algorithm is the best performing model from all of the runs. The accuracy of the best performing model and median performing model are the main metrics used to compare different classifier designs. The true positive, true negative, false positive, and false negative counts (in percentage) will be kept for analysis of the final model. The average training time and testing time will also be collected for comparing different classifiers.

```
def main():

    '''Collect and Preprocess Data'''
    # Download Data and return directory that contains the data
    data_dir = get_data()

    # Preprocess Data (Data Augmentation and datasplit splits)
    train_ds, val_ds, test_ds = preprocess_data(data_dir,aug_split = 0)

    # Make the test_dataset an iterable object
    test_ds_copy = iter(test_ds)

    '''Retrain and Test Model Many Times'''
    results = []
    best_accuracy = 0
    num_iterations = 50
    for i in range(num_iterations):

        # Create copy of the datasets so that it can be reused
        test_ds, test_ds_copy = itertools.tee(test_ds_copy)

        # Define and Train the Model
        model = define_model()
        model, history, training_time = train(model, train_ds, val_ds)
```

```
    # Extract True Labels from dataset and Predict Labels from images
    true_labels, predicted_labels = predict_labels(model,test_ds)

    # Calculates TP, TN, FP, FN and total accuracy
    results.append(calculate_metrics(true_labels, predicted_labels))

    # Keep track of the best performing model.
    if results[i][0] > best_accuracy:
        best_accuracy = results[i][0]
        best_model = model
        best_history = history

# Return the best model
return best_model
```

Code Block 4.4.2: Main Function. Trains and Tests 50 Models and Returns the Best One

4.5 Baseline Results

The baseline classifier design has now been fully defined. It currently uses 3 training epochs, images with 700x460 pixel resolution and 3 channel RGB, an architecture from the TensorFlow Core tutorial [22] (will be referred to as Model 1.0), and no data augmentations. All of these factors make the training and performance of this model unique, so all of these parameters will be changed to try to improve the final results. The processing time depends on computer hardware, which is described in **A.3**.

Table 4.5.1 shows the baseline results. In future adaptations, the goal will be to try to improve the classifier accuracy from this baseline. **Figure 4.5.1** shows the accuracy distribution for the current classifier. The distribution shows that the higher performing models occur more often than poorly performing models. The best performing model achieves an accuracy of 88.67%. In future classifiers, it would be better if the mean and median accuracies increased so that there is a higher likelihood that the models perform very well. The standard deviation on the model accuracy is very high. Ideally, the standard deviation would be less than 2%, so decreasing the variance is another goal for future classifiers.

Table 4.5.1: Results for Model 1.0

Structure	Model 1.0
Modifications	None
Accuracy (%)	**82.91** ± 5.65
Median Accuracy (%)	**84.57**
Accuracy Range (%)	[69.14, **88.67**]
TP (%)	62.87 ± 5.32
TN (%)	20.03 ± 7.82
FP (%)	10.83 ± 7.82
FN (%)	6.27 ± 5.32
Avg. Training Time (s)	**81.98** ± 1.55
Avg. Testing Time (s)	6.96 ± 0.23
Total Time (s)	4455.16

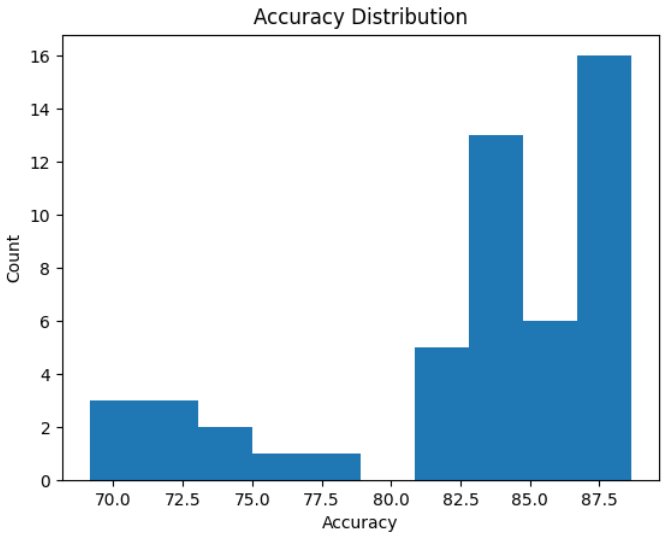


Figure 4.5.1: Accuracy Distribution for Model 1.0

4.6 General Modifications

There should be modifications that can be made to improve a classifier with any structure. This section will attempt to make general modifications that increase the classification accuracy of any classifier. These changes involve changing image resolution, number of epochs, and color channels.

In order to run the first test, it took 4,455 seconds, or 74 minutes and 15 seconds. In an effort to save time, the images in the dataset are going to be rescaled from 700x460 pixels to 256x256 pixels. This resolution is the default size parameter for extracting the dataset with keras image_dataset_from_directory method, so that is why it was chosen. **Table 4.6.1** shows the results from decreasing the resolution. Instead of using 50 iterations, this test performed 150 iterations and only took 2,888 seconds, or 48 minutes and 8 seconds. Using 150 iterations for this test made the distribution of the classifier's performance more accurate. This was also done to prove the significance that the resolution plays for time efficiency. If performed for just 50 iterations, it would have taken roughly 16 minutes. With lower resolution, the classifier performed significantly better on average. The standard deviation of the data is still higher than desired. The best case improved by 0.78%, which is not significant, but still an improvement. The median improved by 2.41%. The distribution in **Figure 4.6.1** shows that the best performing models have a high chance of occurring. It also shows that the lowest case (58.2%) is an outlier. It is possible that the larger features in the images were more important for classifying the tissues. By lowering the resolution, the smallest features in the image were likely removed, which could explain the improved performance. Moving forward, 256x256 pixel resolution will continue to be used.

Table 4.6.1: Results from Reducing Image Resolution (Model 1.0)

Structure	Model 1.0	Model 1.0
Modifications	None	256x256 pixel resolution
Number of Runs	50 (Standard)	150
Accuracy (%)	**82.91** ± 5.65	**85.18** ± 4.79
Median Accuracy (%)	**84.57**	**86.98**
Accuracy Range (%)	[69.14, **88.67**]	[58.2, **89.45**]
TP (%)	62.87 ± 5.32	62.85 ± 5.58
TN (%)	20.03 ± 7.82	22.33 ± 5.18
FP (%)	10.83 ± 7.82	8.53 ± 5.18
FN (%)	6.27 ± 5.32	6.29 ± 5.58
Avg. Training Time (s)	**81.98** ± 1.55	**15.76** ± 0.44
Avg. Testing Time (s)	6.96 ± 0.23	3.44 ± 0.16
Total Time (s)	4455.16	2887.76

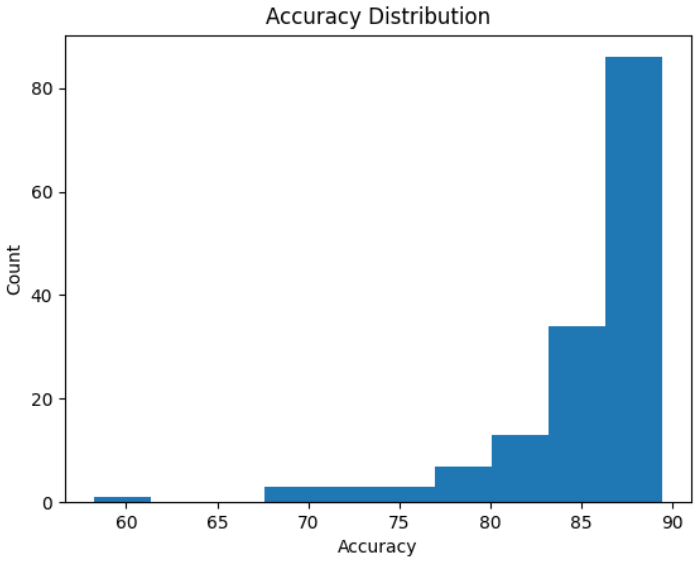


Figure 4.6.1: Accuracy Distribution for Model 1.0 with Image Resolution 256x256

In the paper by Md Zahangir Alom et al. [21], the authors wrote about the effects of increasing the number of epochs used for training. They showed that classifiers can perform poorly for the first few epochs, but they should continue to improve until they are overfitting. The classifier that the authors were using seemed to stop improving much beyond 20 epochs, but it did not degrade beyond that point. Since the current classifier only trains models over three epochs, the next test will use 25 epochs.

As shown in **Table 4.6.2**, increasing the number of epochs improves the average accuracy by 5.44% and the median accuracy by 3.97%. The best model performs with 94.4% accuracy (4.95% increase), but it occurs infrequently (see **Figure 4.6.2**) compared to the best models from using only 3 epochs. Using more epochs means that the training data was fed through the model more times than before, so it took much longer than before to run this test. On average, training a single model took about 124 seconds rather than 16 seconds. If the resolution wasn't already reduced, it would have taken much longer. Using more epochs will make it so that different architectures and modifications will be more easily comparable. It is possible that an architecture will learn more slowly than another or that a model has poor initialization weights. So it is more fair to make comparisons after many epochs than it is after just a few. Additionally, using just a few epochs has higher variance, so using more epochs should help get more consistent results. In general, this change is intended to improve the performance of any classifier that is defined later.

Table 4.6.2: Results from Increasing Number of Epochs (Model 1.0)

Structure	Model 1.0	Model 1.0
Modifications	256x256, 3 epochs	256x256, 25 epochs
Number of Runs	150	50 (Standard)
Accuracy (%)	**85.18** ± 4.79	**90.62** ± 1.90
Median Accuracy (%)	**86.98**	**90.95**
Accuracy Range (%)	[58.2, **89.45**]	[83.85, **94.40**]
TP (%)	62.85 ± 5.58	64.42 ± 1.79
TN (%)	22.33 ± 5.18	26.20 ± 2.14
FP (%)	8.53 ± 5.18	4.66 ± 2.14
FN (%)	6.29 ± 5.58	4.72 ± 1.79
Avg. Training Time (s)	**15.76** ± 0.44	**123.56** ± 1.27
Avg. Testing Time (s)	3.44 ± 0.16	3.45 ± 0.19
Total Time (s)	2887.76	6358.04

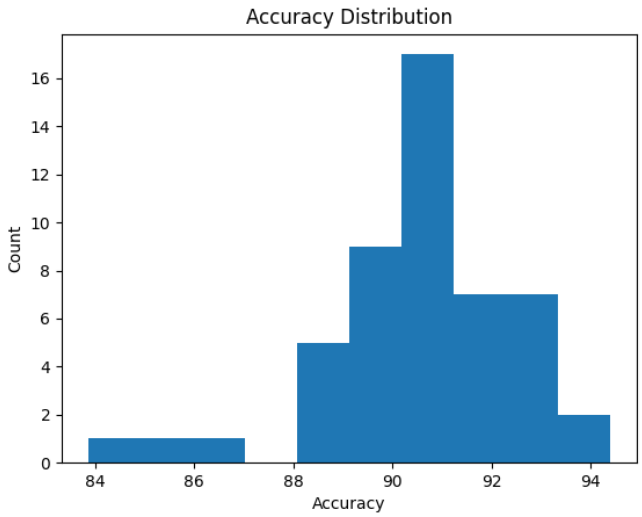


Figure 4.6.2: Accuracy Distribution for Model 1.0 Trained Over 25 Epochs

The last modification that will be made to try to improve all architectures is changing the number of color channels. Currently, the images are 3 channel RGB. For other image processing

techniques, it can sometimes be advantageous to convert to a single color channel (grayscale). Grayscale is useful when the classification is based on pixel intensity or feature shapes rather than color. It is similar to performing stain normalization like in the paper from Yassir Benhammou et al.

The results from performing classification with grayscale images are shown in **Table 4.6.3** and **Figure 4.6.3**. Changing the images to grayscale significantly degraded the performance of the classifier. The mean, median, and best accuracies are worse than the baseline classifier design. Moving forward, grayscale will not be used. The final version of a classifier defined with Model 1.0 uses 256x256 pixel resolution, 25 epochs, and 3 channel RGB. These changes will be used for all classifier designs moving forward.

Table 4.6.3: Results from Converting to Grayscale (Model 1.0)

Structure	Model 1.0	Model 1.0
Modification	RGB	Grayscale
Accuracy (%)	**90.62** ± 1.90	**77.21** ± 4.54
Median Accuracy (%)	**90.95**	**78.39**
Accuracy Range (%)	[83.85, **94.40**]	[68.10, **83.85**]
TP (%)	64.42 ± 1.79	60.73 ± 3.97
TN (%)	26.20 ± 2.14	16.47 ± 5.51
FP (%)	4.66 ± 2.14	14.39 ± 5.51
FN (%)	4.72 ± 1.79	8.41 ± 3.97
Avg. Training Time (s)	**123.56** ± 1.27	**105.60** ± 1.05
Avg. Testing Time (s)	3.45 ± 0.19	3.15 ± 0.16
Total Time (s)	6358.04	6056.84

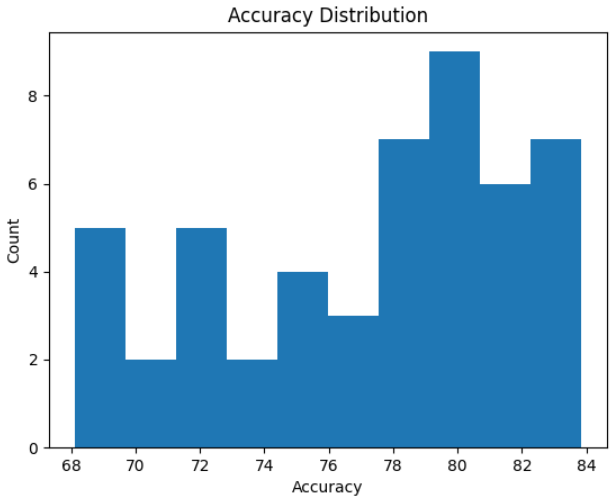


Figure 4.6.3: Accuracy Distribution for Model 1.0 Using Grayscale Images

4.7 Neural Network Structure Exploration

Most of the improvements to the classifier should be dependent on the neural network architecture used. This section will explore VGG16 and ResNet50 architectures as well as a custom architecture based on the existing Model 1.0.

4.7.1 VGG16

VGG is an older structure that is still used in some neural network projects for comparison to past models. So it is the first architecture to be tested against Model 1.0. VGG16 (structure shown in **Figure 4.7.1.1**) was identified as the best performing VGG model on the ImageNet dataset [30], so now it will be used on the BreakHis dataset to see how it performs. VGG16 works best when images are 224x224, so the image resolution will be further reduced to be used for this structure.

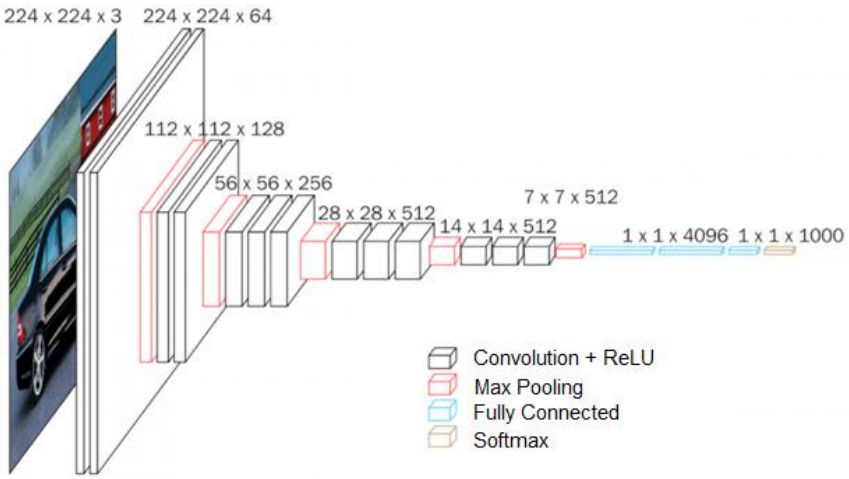


Figure 4.7.1.1: VGG16 Architecture [30]

The results for VGG16 are shown in **Table 4.7.1.1** and **Figure 4.7.1.2**. While the best classifier model performed very well (94.4%), the accuracy distribution shows that it is very unlikely to occur. In fact, 39 of the 50 tests resulted in models that only had an accuracy of 69.14%. It should also be noted that the false positives and true positives accounted for over 90% of all guesses. In other words, the classifier is guessing that almost all of the test images are malignant, so most of the models generated are worthless.

Table 4.7.1.1: Results Comparing Model 1.0, VGG16

Structure	Model 1.0	VGG16
Accuracy (%)	**90.62** ± 1.90	**73.04** ± 7.64
Median Accuracy (%)	**90.95**	**69.14**
Accuracy Range (%)	[83.85, **94.40**]	[69.14, **94.40**]
TP (%)	64.42 ± 1.79	68.26 ± 1.86
TN (%)	26.20 ± 2.14	4.78 ± 9.30
FP (%)	4.66 ± 2.14	26.08 ± 9.30
FN (%)	4.72 ± 1.79	0.88 ± 1.86
Avg. Training Time (s)	**123.56** ± 1.27	**743.79** ± 3.93
Avg. Testing Time (s)	3.45 ± 0.19	4.45 ± 0.12
Total Time (s)	6358.04	37540.91

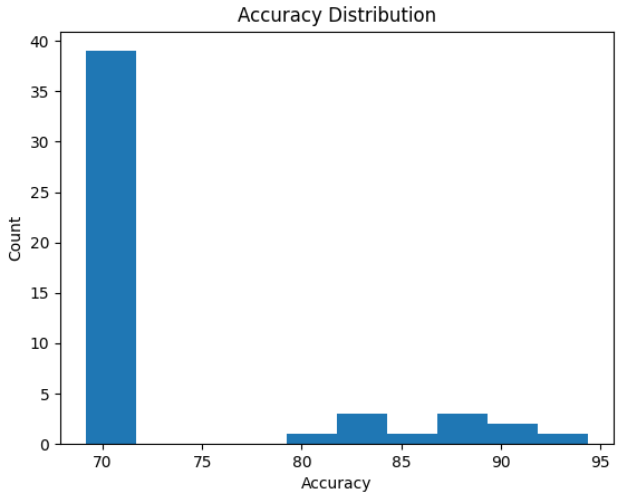


Figure 4.7.1.2: Accuracy Distribution for VGG16 Architecture

Upon further investigation, the VGG16 structure requires specific preprocessing of the images before training or testing the model. The preprocessing involves converting the input images from RGB to BGR and zero-centering each color channel based on the average pixel color. Unfortunately, preprocessing the images for VGG16 did not use the similar methods to other augmentations made later on. Since fixing this issue was not simple, there was not enough time to fix the issue and retest VGG16. VGG16 will not be the final model used.

4.7.2 ResNet50

The residual neural network (ResNet) was designed by Kaiming He et al. [31] and was used to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2015. It was designed with Residual Learning building blocks (see **Figure 4.7.2.1**) that are designed to overlay the feature maps onto previous layers of feature map or the original image. The weight layers are convolutional layers with relu activation functions. Using this building block is comparable to sharpening an image before further processing. Continuing with this analogy: by filtering the image, an edge map can be created, and by adding the edge map back to the image, sharper images can be made. This makes it so that the features are less abstracted between layers, so there will be some reference to the original image or previous blocks to use. Using these building blocks reduces

the issue of vanishing/exploding gradients (where the values become too small/large after many layers). This allows for deeper networks that do not significantly increase time complexity.

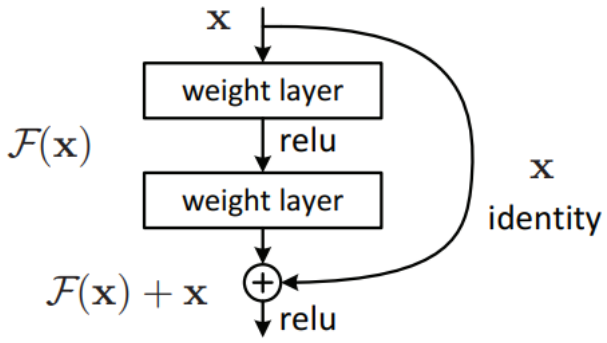


Figure 4.7.2.1: Residual Learning: a Building Block [31]

ResNet50 is a version of ResNet that is 50 layers deep. For this experiment, the ResNet50 architecture is used to show how it compares to the Model 1.0 structure. The ResNet50 architecture has a parameter for defining pre-trained weights for the initial model, so separate tests will be run for random initialization and pre-trained weights. The pre-trained weights are the weights that resulted in the best results for the ImageNet dataset.

Table 4.7.2.1 shows how ResNet50 compares to Model 1.0. Regardless of the weight initialization method, ResNet50 varied greatly compared to Model 1.0. The median models also perform worse than the median for Model 1.0. **Figures 4.7.2.2** and **4.7.2.3** show the distribution of accuracies for ResNet50 with and without pre-trained weights. This shows that models that were initialized with pre-trained weights are much more likely to create better models. It is likely that the pre-trained weights helped the classifier avoid local minimums in the gradient descent process. pre-trained weights should be used for weight initialization rather than random initialization.

Table 4.7.2.1: Results Comparing Model 1.0 and ResNet50

Structure	Model 1.0	ResNet50	ResNet50
Initialization Weights	Random	Random	ImageNet
Accuracy (%)	**90.62** ± 1.90	**63.11** ± 20.81	**79.46** ± 13.83
Median Accuracy (%)	**90.95**	**65.82**	**82.75**
Accuracy Range (%)	[83.85, **94.40**]	[30.86, **90.89**]	[35.42, **95.83**]
TP (%)	64.42 ± 1.79	38.30 ± 25.29	54.51 ± 15.84
TN (%)	26.20 ± 2.14	24.81 ± 8.46	24.95 ± 8.51
FP (%)	4.66 ± 2.14	6.05 ± 8.46	5.91 ± 8.51
FN (%)	4.72 ± 1.79	30.84 ± 25.29	14.63 ± 15.84
Avg. Training Time (s)	**123.56** ± 1.27	**576.55** ± 18.66	**569.45** ± 4.05
Avg. Testing Time (s)	3.45 ± 0.19	5.27 ± 0.35	5.16 ± 0.19
Total Time (s)	6358.04	29168.60	28844.59

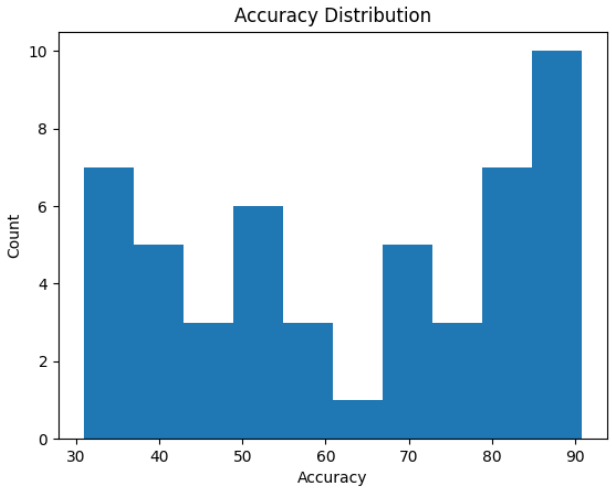


Figure 4.7.2.2: Accuracy Distribution for Randomly Initialized ResNet50 Architecture

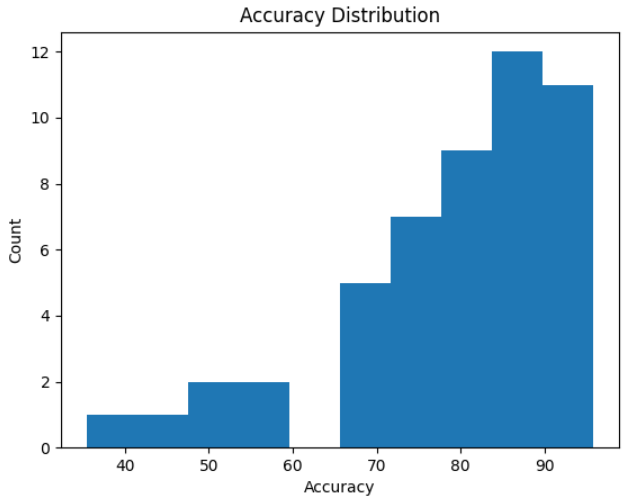


Figure 4.7.2.3: Accuracy Distribution for pre-trained ResNet50 Architecture

While the classifier with pre-trained weights and ResNet50 structure results in some very poor performing models, it also achieves the best model that has been seen so far (95.83% accuracy). Recall, Model 1.0 had few instances of the model working at the highest accuracy, but in most instances the models were at least performing above 88%. Model 1.0 is more likely to perform well if given few chances to train, but the ResNet50 model had the best performing model overall. For future testing, Model 1.0 will be used because ResNet50 is highly variable and takes a long time to train.

4.7.3 Custom Design

This section explores how simple modifications to the original architecture (Model 1.0) can affect the performance of the classifier. The number of layers, nodes per layer, and types of layers will be the parameters that are tweaked to make these changes. The model variants will be very shallow networks. Since the VGG16 and ResNet50 architectures are more complex, testing with a more shallow network can illuminate the importance of architecture depth. It could also reveal more about the complexity required in diagnosing cancer using neural networks.

The first architecture modification (Model 1.1) experiments with removing two of the three convolutional layers and pooling layers. For reference, Model 1.0 and Models 1.1 are shown in **Code Block 4.3.7.1**. The hope is that the neural network will perform analysis on less complicated

features. It is possible that simpler features are better able to characterize the classes than more complicated and abstracted ones.

```
'''Model 1.0'''
model = tf.keras.Sequential([
    tf.keras.layers.Rescaling(1./255),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(2)])
```

```
'''Model 1.1'''
model = tf.keras.Sequential([
    tf.keras.layers.Rescaling(1./255),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(2)])
```

Code Block 4.7.3.1: Model 1.0 (Top), Model 1.1 (Bottom)

Removing convolutional layers improved the results in general. The mean, median and best models performed better with Model 1.1 than with Model 1.0. Removing the layers did not change the average training time significantly. The only issue with Model 1.1 is that the accuracies are distributed across a wider range than Model 1.0. Fortunately, according to the distribution shown in **Figure 4.7.3.1**, most of the models performed with high accuracy. So it is likely that if trained only once, the model returned would likely perform at a rate higher than 90%. Model 1.1 returned the best model so far, with an accuracy of 96.35%. From this experiment, it can be proposed that the images can be classified based on simpler features.

Table 4.7.3.1: Results Comparing Model 1.0 and Model 1.1

Structure	Model 1.0	Model 1.1
Accuracy (%)	**90.62** ± 1.90	**92.16** ± 4.38
Median Accuracy (%)	**90.95**	**93.62**
Accuracy Range (%)	[83.85, **94.40**]	[71.48, **96.35**]
TP (%)	64.42 ± 1.79	65.65 ± 2.08
TN (%)	26.20 ± 2.14	26.51 ± 3.84
FP (%)	4.66 ± 2.14	4.35 ± 3.84
FN (%)	4.72 ± 1.79	3.49 ± 2.08
Avg. Training Time (s)	**123.56** ± 1.27	**123.79** ± 1.95
Avg. Testing Time (s)	3.45 ± 0.19	3.50 ± 0.12
Total Time (s)	6358.04	6372.14

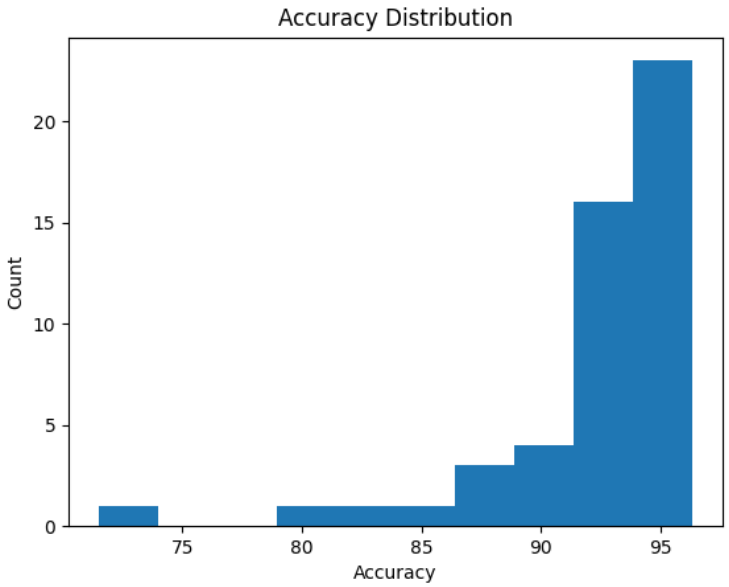


Figure 4.7.3.1: Accuracy Distribution for Model 1.1 Architecture

From Model 1.1, Model 1.2 was created. The only difference is that the flattening layer feeds into a larger dense layer: increased from 128 nodes to 256. The thought behind this is that doubling the number of calculations performed in the dense layer will double the amount of information used in the final decision. Unfortunately, the increased number of nodes did not affect

the results significantly (see **Figure 4.7.3.2**). The distribution of accuracy (**Figure 4.7.3.2**) does not show significant changes between the models. Based purely on the difference in training time, Model 1.2 will not be used to further develop the classifier.

Table 4.7.3.2: Results Comparing Model 1.1 and Model 1.2

Structure	Model 1.1	Model 1.2
Accuracy (%)	**92.16** ± 4.38	**92.13** ± 4.81
Median Accuracy (%)	**93.62**	**93.55**
Accuracy Range (%)	[71.48, **96.35**]	[73.83, **95.83**]
TP (%)	65.65 ± 2.08	66.01 ± 2.02
TN (%)	26.51 ± 3.84	26.12 ± 5.40
FP (%)	4.35 ± 3.84	4.73 ± 5.40
FN (%)	3.49 ± 2.08	3.14 ± 2.02
Avg. Training Time (s)	**123.79** ± 1.95	**153.64** ± 11.71
Avg. Testing Time (s)	3.50 ± 0.12	3.63 ± 0.23
Total Time (s)	6372.14	7871.10

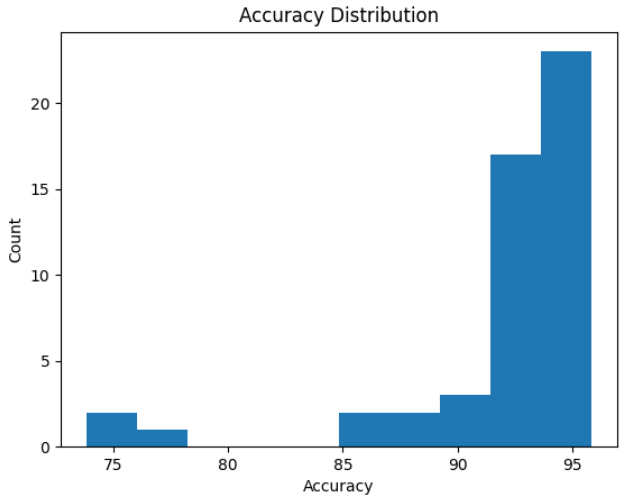


Figure 4.7.3.2: Accuracy Distribution for Model 1.2 Architecture

Model 1.3 is another alteration made from Model 1.1. Instead of changing the number of layers or number of nodes, Model 1.3 doubles the amount of filters used in the convolutional layer.

This will result in 64 feature maps that can be analyzed rather than 32. Though increasing the number of feature maps increased the amount of data passed through the rest of the structure, it did not seem to provide more useful information. **Table 4.7.3.3** shows that Model 1.1 and 1.3 worked about the same. The distribution for Model 1.3 (**Figure 4.7.3.3**) is similar to the distribution for Model 1.1, but higher performing models occurred slightly more frequently for Model 1.3. Since the algorithm picks the best model from all 50, Model 1.1 will continue to be used because it achieved the best performing model. It also takes less time to train.

Table 4.7.3.3: Results Comparing Model 1.1 and Model 1.3

Structure	Model 1.1	Model 1.3
Accuracy (%)	**92.16** ± 4.38	**92.00** ± 5.17
Median Accuracy (%)	**93.62**	**93.42**
Accuracy Range (%)	[71.48, **96.35**]	[63.02, **95.31**]
TP (%)	65.65 ± 2.08	64.89 ± 5.10
TN (%)	26.51 ± 3.84	64.89 ± 5.10
FP (%)	4.35 ± 3.84	3.75 ± 2.90
FN (%)	3.49 ± 2.08	4.25 ± 5.10
Avg. Training Time (s)	**123.79** ± 1.95	**173.22** ± 2.31
Avg. Testing Time (s)	3.50 ± 0.12	3.37 ± 0.17
Total Time (s)	6372.14	8837.20

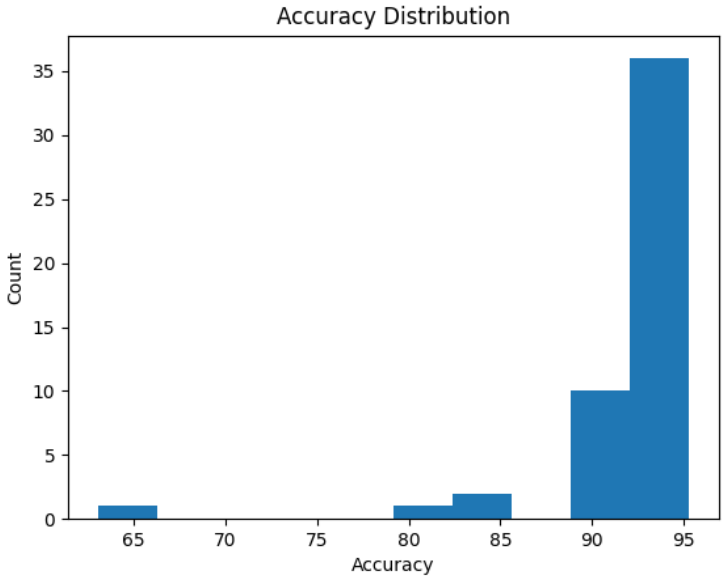


Figure 4.7.3.3: Accuracy Distribution for Model 1.3 Architecture

The final alteration of the original model will be Model 1.4, which is also building off of Model 1.1. For this model, there is an additional layer between the max pooling layer and the flattening layer (shown in **Code Block 4.7.3.2**). The dropout layer is designed to drop (set to 0) a percentage amount of values from each image during training [32]. So after the feature maps are pooled, 20% of the values are randomly selected and set to 0 before being sent to the flattening layer. This is similar to adding noise to a training set: it prevents overfitting because the data will not be exactly the same images each time. Between epochs, different pixels will be set to zero, making the training images unique every epoch. This strategy is often applied to datasets with small amounts of data so that the training set is more diversified.

```
'''Model 1.4'''
model = tf.keras.Sequential([
    tf.keras.layers.Rescaling(1./255),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(2)])
```

Code Block 4.7.3.2: Model 1.4

Including the dropout layer seemed to make the classifier vary less, while not changing the average performance. **Table 4.7.3.4** shows that the structures performed similarly, but Model 1.1 tends to work slightly better in general. More importantly, **Figure 4.7.3.4** shows that the accuracy distribution for Model 1.4 only has a few instances where the models perform at the highest level. Since Model 1.4 cannot be used to consistently achieve models with high accuracy, it will not be the structure that is used when moving forward.

Table 4.7.3.4: Results Comparing Model 1.1 and Model 1.4

Structure	Model 1.1	Model 1.4
Accuracy (%)	**92.16** ± 4.38	**92.12** ± 2.44
Median Accuracy (%)	**93.62**	**92.77**
Accuracy Range (%)	[71.48, **96.35**]	[82.94, **95.70**]
TP (%)	65.65 ± 2.08	65.53 ± 2.25
TN (%)	26.51 ± 3.84	26.59 ± 2.31
FP (%)	4.35 ± 3.84	4.27 ± 2.31
FN (%)	3.49 ± 2.08	3.61 ± 2.25
Avg. Training Time (s)	**123.79** ± 1.95	**130.50** ± 1.58
Avg. Testing Time (s)	3.50 ± 0.12	3.47 ± 0.17
Total Time (s)	6372.14	6706.43

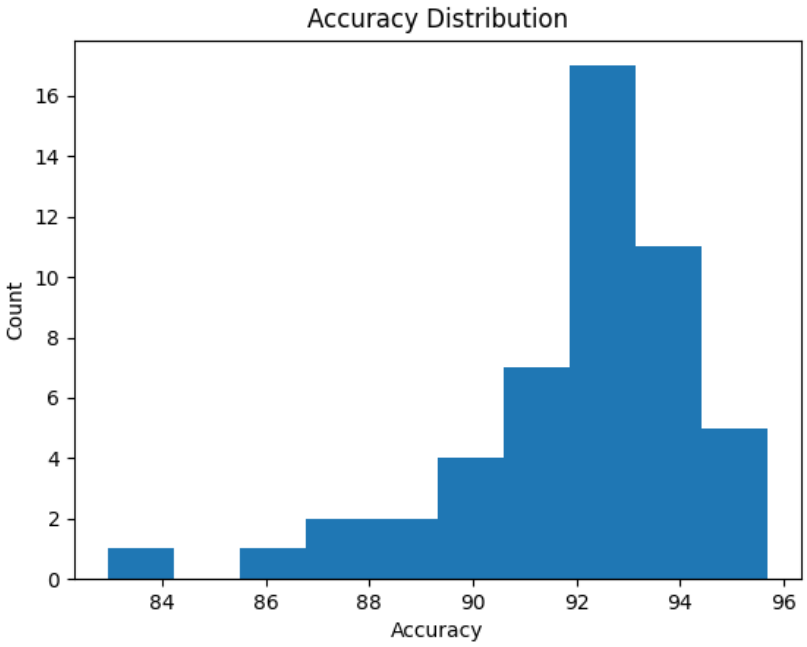


Figure 4.7.3.4: Accuracy Distribution for Model 1.4 Architecture

4.7.4 Neural Network Structure Conclusion

Model 1.1 produces the highest classification accuracy. It was able to achieve a model that correctly classified 96.35% of the test samples. Structures for Models 1.2, 1.3 and 1.4 also perform at similar levels and have a high likelihood of returning high performing models. ResNet50 had models that performed with high accuracy, but also had some of the worst models seen. The high variability of the ResNet structure in this use case made it undesirable. However, it should be noted that using pre-trained weights for this structure can improve the performance when compared to random initialization. Due to inconsistent bin sizing and ranges for the accuracy distribution plots, it is hard to tell exactly how some of the models compare to each other. Since Model 1.1 returned the best model and consistently resulted in other high performing models, it will be the structure that is used in future sections.

Based on the high performance of simple structures, it seems that shallow networks could be enough to classify the BreakHis dataset. Features that can be easily extracted from the first layer

were enough to perform very well. It seems plausible that classical image processing techniques could have been used to achieve similar classification rates for this dataset.

Unfortunately, the issue of preprocessing was not solved for VGG16. If given more time, a solution would have been found. However, VGG16 is an older network and would probably not work as well as a more recent architecture. So it seems unlikely that VGG16 would have been able to improve the classification rate anyway.

4.8 Data Augmentation

The final step in designing the classifier is adding augmented data. By increasing the amount of training samples available during training, it is possible to reduce the chance of overfitting the model to the data. As discussed in **Chapters 3.1** and **3.2**, other researchers have been able to use data augmentation as a last step in improving their classifiers. Alom et al. made 21 augmented copies of each sample from the training dataset [21]. Increasing the size of the training set will increase the time that it takes to train the classifier, so it was decided that a smaller amount of data will be used for augmentation to see if it improves the results.

The first step in augmentation is copying data from the training set. This process is performed when the dataset is partitioned, as shown in **Code Block 4.8.1**. Since there are fewer samples being augmented, it is possible to split the data to retain the same train:validate:test ratio of 80:10:10. More accurately, the data will be pulled from the dataset such that the ratio (train+augmented):validate:test = 80:10:10. To make sure that the ratio is maintained, the test and validation sets must take more samples from the original dataset. For example, if the augmentation split is 20%, that means it will augment 20% of the training data. The training data and augmented data should be 80% of the total, so the total dataset expands by a factor of (1 + 0.8*0.2) = 1.16. The testing and validation sets will each be 10% of this expanded size. This value is used for extracting the correct number of samples for the testing and validation sets, and the leftover will be the training set. The augmented data will be a copy of a subset of the training data. Prior to augmentation, the

batch ratio was 200:24:24. After partitioning with augmented data, the batch ratio is 231:28:28. This confirms that the ratio was maintained and the data size increased.

```
def get_dataset_partitions(ds, train_split=0.8, aug_split=0, val_split=0.1, test_split=0.1):

    # Expanded size accounts for adding the augmented data points
    expanded_size = 1 + train_split*aug_split
    val_size = int(val_split * expanded_size * len(ds))

# Partition Data
    val_ds = ds.take(val_size)
    test_ds = ds.skip(val_size).take(val_size)
    train_ds = ds.skip(val_size).skip(val_size)

    # Only perform augmentation if the range is valid
    if aug_split > 0 and aug_split <=1:
        # Define the size of each split based on the dataset size and the splits
        aug_size = int(train_split * len(ds) * aug_split)
        aug_ds = ds.skip(val_size).skip(val_size).take(aug_size)
    else:
        aug_ds = None

    return train_ds, val_ds, test_ds, aug_ds
```

Code Block 4.8.1: Partitions Dataset into Training, Augmented, Validation, and Testing Dataset

At this point, the augmented dataset is still just copies of some training samples. As discussed before, rotating and flipping the samples makes them unique enough to reuse. Rotation and random flipping are applied to the data, as shown in **Code Block 4.8.2**. The rotation consistently flips the images 126°, but there is not a simple way to make the flipping consistent. The rotation should be enough to make the images different from the training samples, so random flipping does not need to occur consistently. Flipping will be horizontal, vertical, both, or neither. Since only one augmented image will be made per training sample, there is no risk of repeating augmented images. The augmentations are applied in parallel, which significantly reduces the time that it took to train the augmented section of the dataset. Flipping can only happen in 4 directions and rotation is fixed, so there are only 4 augmentations possible (shown in **Figure 4.8.1**), but all of them will be different from the original. After performing the augmentation, the augmented data will be added to the training set.

```
def augment_data(train_ds,aug_ds):

    # Defines random rotation and flip
    rot_and_flip_aug = tf.keras.Sequential([
        tf.keras.layers.RandomFlip("horizontal_and_vertical",seed=123),
        tf.keras.layers.RandomRotation(factor=(.35,.35),seed=123)])

    # Applies rotation and flip to the data.
    AUTOTUNE = tf.data.AUTOTUNE
    aug_ds = aug_ds.map(
        lambda x, y: (rot_and_flip_aug(x, training=True), y),num_parallel_calls=AUTOTUNE)

    # Append the augmented data to the training data
    train_ds = train_ds.concatenate(aug_ds)

    return train_ds
```

Code Block 4.8.2: Performs Augmentation

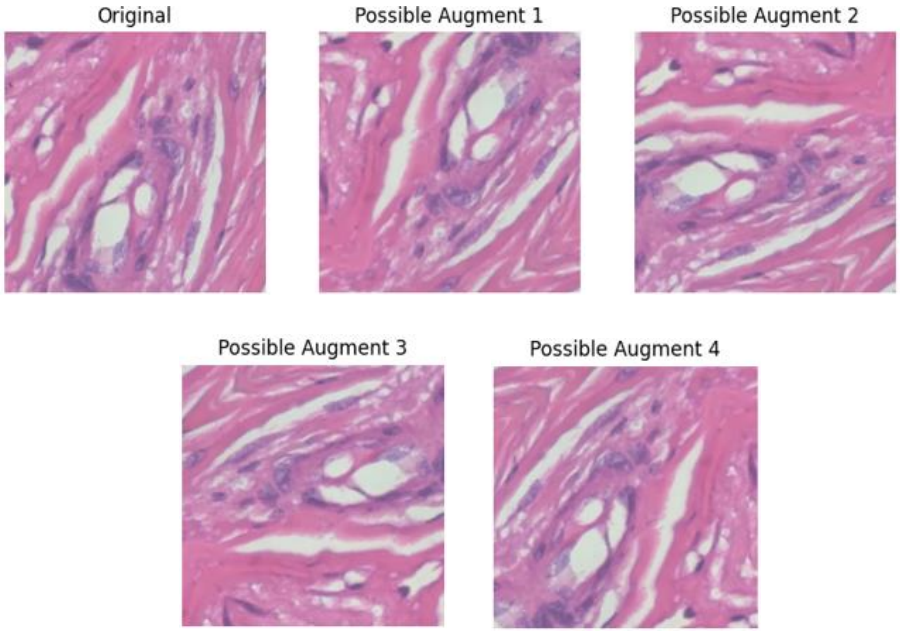


Figure 4.8.1: Original Image and Possible Augmentations

When using augmented data for training Model 1.1, it made it perform worse. See **Table 4.8.1** for comparison to Model 1.1 without augmented data and **Figure 4.8.2** for the accuracy distribution. The mean, median, and best performing model all decreased for the classifier trained with the augmented data.

Table 4.8.1: Results Comparing Model 1.1 With and Without Augmented Training Data

Structure	Model 1.1	Model 1.1 (Augmented)
Accuracy (%)	**92.16** ± 4.38	**88.17** ± 4.17
Median Accuracy (%)	**93.62**	**89.29**
Accuracy Range (%)	[71.48, **96.35**]	**[**67.97, **92.75**]
TP (%)	65.65 ± 2.08	62.21 ± 3.20
TN (%)	26.51 ± 3.84	25.96 ± 4.92
FP (%)	4.35 ± 3.84	6.07 ± 4.92
FN (%)	3.49 ± 2.08	5.75 ± 3.20
Avg. Training Time (s)	**123.79** ± 1.95	**163.17** ± 2.29
Avg. Testing Time (s)	3.50 ± 0.12	4.11 ± 0.23
Total Time (s)	6372.14	8373.00

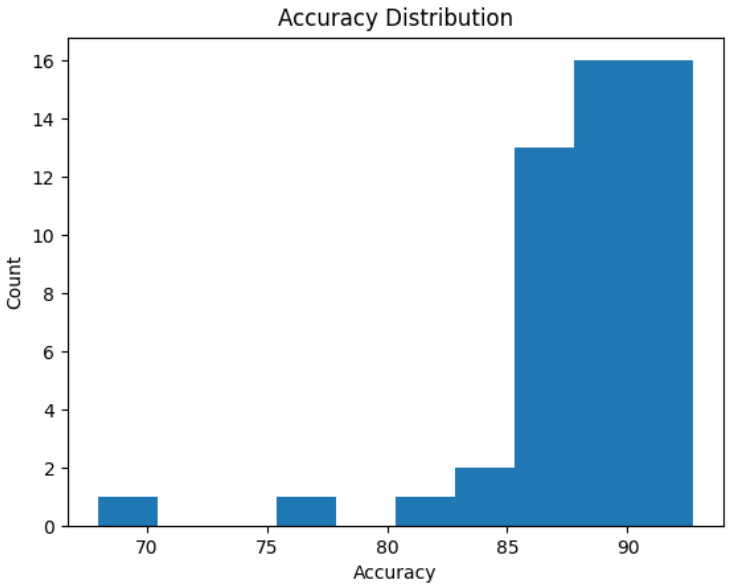


Figure 4.8.2: Accuracy Distribution for Model 1.1 With Augmented Training Data

There are a few ways that the augmentation algorithm could have been improved. When designing the augmentation portion of the program, it was designed to maintain the train:test:validation ratio. Since the validation and testing datasets had to come directly from the

original, it limited how much data could be added to the training set while maintaining the ratio. Instead, 20% of the original samples should have been set aside for validation and testing, then the remaining 80% would have been used to make many copies of augmented data. It would change the ratio, but there would still be enough data for testing and validation.

Another factor that limited the augmentation is that it only had four possible combinations. Ideally, many copies of the same image could be used to make several unique images. This could be done by either making a more complicated rule for determining the amount of rotation and flipping to apply, or by adding more augmentation options such as shift, shear, zoom, or stain normalization. This would allow the training dataset to grow immensely, and it would likely improve the results.

Due to time constraints, these issues could not be addressed. Rather than maintaining the ratio, focusing on increasing the training dataset size would probably have been a better solution.

4.9 Final Results

Once a viable classifier design was made, general modifications were made to improve the results. The modifications include reducing the resolution from 700x460 to 256x256 and increasing the number of training epochs from 3 to 25. The next step was testing multiple architectures to find the highest performing one. Unfortunately, VGG16 did not function correctly because the preprocessing was not defined properly. ResNet50 worked well some of the time, but the quality of models returned from this structure varied greatly. Using pre-trained weights instead of randomly initialized weights improved the results greatly, but ResNet50 still did not work as well as the custom structures. The custom structures worked well in general, but the best structure was Model 1.1. Model 1.1 reduced the number of convolutional and pooling layers from 3 to 1 so that it made decisions based on simpler feature maps. Adding augmentation to the data did not improve the classifier, but it might have helped if more samples were added.

Table 4.9.1 shows the baseline results and the final results. The average model for the final classifier performs 9.25% better than the original. The median model works 9.05% better than the

median model of the original model. The best model from the final classifier performed with 96.35% accuracy, which is 7.68% better than the best model from the original classifier. The final classifier only takes a little over 2 minutes to train, which is relatively fast compared to ResNet50 or other structures.

Table 4.9.1: Results Comparing the Original Model 1.0 and Final Model 1.1

Structure	Original	Final
Accuracy (%)	**82.91** ± 5.65	**92.16** ± 4.38
Median Accuracy (%)	**84.57**	**93.62**
Accuracy Range (%)	[69.14, **88.67**]	[71.48, **96.35**]
TP (%)	62.87 ± 5.32	65.65 ± 2.08
TN (%)	20.03 ± 7.82	26.51 ± 3.84
FP (%)	10.83 ± 7.82	4.35 ± 3.84
FN (%)	6.27 ± 5.32	3.49 ± 2.08
Avg. Training Time (s)	**81.98** ± 1.55	**123.79** ± 1.95
Avg. Testing Time (s)	6.96 ± 0.23	3.50 ± 0.12
Total Time (s)	4455.16	6372.14

The quality of the models generated from the final classifier varies less than the original classifier. **Figure 4.9.1** shows the distribution of accuracies for both classifiers. The final classifier design tends to output high end models more consistently than the original classifier design. Though the final criteria mostly comes down to the best model, it is also important that the best model is easily replicable.

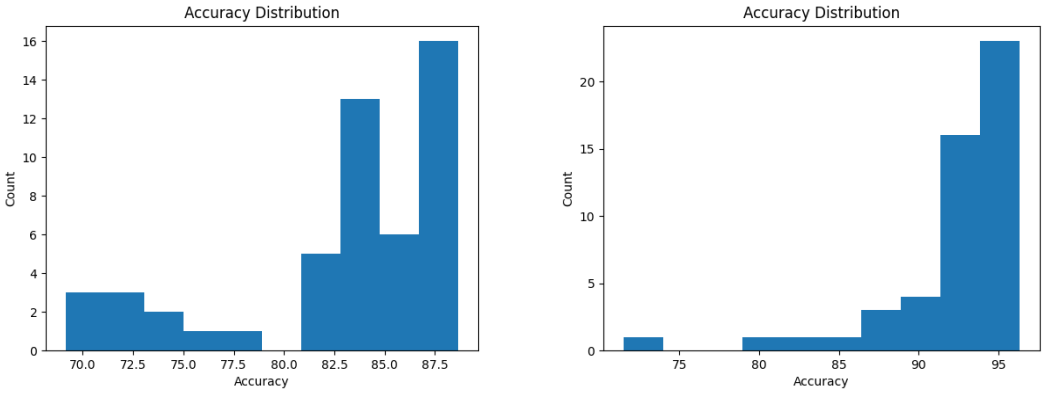


Figure 4.9.1: Accuracy Distributions for Original Model 1.0 (Left) and Model 1.1 (Right)

The confusion matrix is shown in **Table 4.9.2**. Due to the imbalance in the dataset, the classifier has a slight bias towards predicting a sample as malignant. This bias seems to be true, because there is a higher number of false positives than there is for false negatives despite having fewer negative samples. The false negative rate is 5.05% and the false positive rate is 14.10%. This means that the classifier is much more likely to misdiagnose benign samples.

Table 4.9.2: Confusion Matrix for Model 1.1 (Final Model)

	Malignant Sample	Benign Sample
Predicted Malignant	TP: **65.65%**	FP: **4.35%**
Predicted Benign	FN: **3.49%**	TN: **26.51%**

CHAPTER 5. DISCUSSION OF RESULTS

The final classifier from this report achieves an average accuracy rate of (92.16 ± 4.38)%. Yassir Benhammou et al. created a binary classifier with an average accuracy of (88.9 ± 2.5)%. Both of these classifiers are Magnification Invariant Binary (MIB) classifiers, so they are performing the same classification between Benign and Malignant samples from the same dataset. This means that the developed classifier had a higher average accuracy than the classifier designed by this group of researchers. The developed classifier had larger variance than the classifier designed by Benhammou et al., which is not ideal. However, assuming that the best model of 50 is kept—much like the algorithm in this paper—then the best performance of their classifier is unlikely to be as high as the best model (96.35%) in this paper. It should also be noted that Benhaummou et al. were successful in implementing augmentations that improved their results. If the augmentation algorithm for this project were fixed, it likely would have led to a higher performance of the average classifier.

Unlike the classifiers in this paper or in the paper by Benhammou et al., the performances of the classifiers in Alom et al.'s paper are separated by the magnification factor (**Table 5.1**). This makes it a bit more difficult to compare the performances between the classifiers. However, looking at the IRRCNN's average performances for each magnification factor, they tend to range from about 95% to 97%. Though their average is much higher, the classifier developed in this work had models working in this range. This is significant because the IRRCNN structure is much more complicated than the Model 1.1 structure. This shows that using a much simpler structure can still return models that perform at very high levels. Again, improving the augmentation would have benefitted the final results. Alom et al. augmented 21 new images for every image in the training set. If that were done properly for the developed classifier, it could have raised the average to more comparable levels.

Table 5.1: Breast Cancer Classification Results for Binary Classification (Alom et al. [21])

	Method	Year	Classification Rate at Magnification Factor			
			40×	100×	200×	400×
Image Level	CNN +fusion(sum, product, max) [15] (highest results)	2016	85.6 ± 4.8	83.5 ± 3.9	83.6 ± 1.9	80.8 ± 3.0
	AlexNet + Aug [18]	2017	85.6 ± 4.8	83.5± 3.9	83.1 ± 1.9	80.8 ± 3.0
	ASSVM [28]		94.97	93.62	94.54	94.42
	CSDCNN + Aug [18]	2017	95.80± 3.1	96.9 ± 1.9	96.7 ± 2.0	94.90 ± 2.8
	IRRCNN	**2018**	**97.16 ± 1.37**	**96.84 ±1.34**	**96.61± 1.31**	**95.78 ± 1.44**
	IRRCNN + Aug	**2018**	**97.95± 1.07**	**97.57± 1.05**	**97.32± 1.22**	**97.36± 1.02**
Patient Level	CNN +fusion (sum, product, max) [15]	2016	90.0 ± 6.7	88.4 ± 4.8	84.6 ± 4.2	86.10 ± 6.2
	Bayramoglu et al. [14]	2016	83.08 ± 2.08	83.17 ± 3.51	84.63 ± 2.72	82.10 ± 4.42
	Multi-classifier by Gupta et al. [13]	2017	87.2 ± 3.74	88.22 ± 3.23	88.89 ± 2.51	85.82 ± 3.81
	CSDCNN + Aug [18]	2017	92.8 ± 2.1	93.9 ± 1.9	93.7 ± 2.2	92.90 ± 2.7
	IRRCNN +wo aug.	2018	96.69 ±1.18	96.37 ±1.29	96.27±1 .57	96.15 ±1.61
	IRRCNN + w. Aug.	2018	**97.60± 1.17**	**97.65± 1.20**	**97.56± 1.07**	**97.62± 1.13**

As discussed in **Chapter 4.9**, the average model had a false negative rate of 5.05% and a false positive rate of 14.10%. Though only the best model is returned, this still highlights a potential issue of this classifier. Since there is an imbalance in the dataset and augmentation was not used to balance it out, most classifiers designs will have a bias towards diagnosing benign cancer as malignant. In this case, 14.10% of the images will be classified as malignant when there is nothing wrong with them. The simplest way to fix this is by balancing the dataset. By adding more augmentations for benign samples, the training set can be balanced.

CHAPTER 6. CONCLUSION

The final classifier design produces models that correctly diagnose benign and malignant breast cancer with up to 96.35% accuracy. Of the 50 models generated for testing, only 4 did not exceed 85% accuracy and over 20 models achieved accuracies above 93%. Compared to classifier designs by other researchers, the classifier developed in this work performed well. The final classifier design returned models that had a higher average accuracy than similar classifiers designed by Benhammou et al. And the best models were comparable to those from a more modern and complicated classifier design like IRRCNN that was designed by Alom et al.

Compared to pathologists, the classifier also performs with high accuracy. There are not many studies on the misdiagnosis rate of breast cancer, but it is speculated that breast cancer is misdiagnosed by pathologists anywhere from 5% to 28% of the time [33]. This does not apply to histopathological samples or the BreakHis dataset specifically, but it still gives a reference for the quality of breast cancer classification by pathologists.

One of the lessons learned from this project is that larger features are more important than smaller features for diagnosing breast cancer with the BreakHis dataset. When reducing the resolution, smaller features are lost while larger features remain. After reducing the resolution of all of the images, the classifier began performing with higher accuracies and it significantly improved the training time.

This paper also shows that using a simple neural network structure can be highly effective in diagnosing breast cancer from histopathological images. The final structure used for the classifier was the shallowest network of any of the architectures tested, but it had the highest performing models. Since the final classifier used fewer convolutional layers, the features used for analysis were simpler as well. This information shows that it is plausible that a classical computer vision classifier could have performed at a similar caliber. However, it is also important to note that the features being analyzed are more abstract than ones that would be used in classical computer vision.

Since picking features for a classical classifier may not be a simple task, it is probably preferable that a neural network is used for the classifier.

Another lesson learned is that choosing the right network architecture and proper number of training epochs has the most impact on the performance of the classifier. Being able to train for more cycles helps create the classifiers with the highest accuracy, so one of the goals of the architecture is to reduce the chance of overtraining. Data augmentation can also play a large role in this process, but it requires far more samples than were implemented in this project.

An additional factor for the success of a network architecture is whether or not pre-trained weights are available. Though the ResNet50 architecture was not used as the final structure for the classifier, an important lesson was learned about weight initialization. Testing the ResNet50 structure with random initialization and with pre-trained weights demonstrated that there is an advantage to using pre-trained weights for model initialization. Using pre-trained weights improves the average performance of the classifier and increases the proportion of models that achieve higher accuracies. When using a well established structure, pre-trained weights should be used for initialization, if they are available.

The most important takeaway is that the developed classifier is a viable design that is capable of diagnosing breast cancer in its early stages. The classifier has a relatively simple structure that allows for a short training period but still provides accurate classification. Acquiring a proper diagnosis at an early stage is an important step in improving the survivability of breast cancer. This classifier can be used as a way to help decrease the death rate of breast cancer. Hopefully, this paper will provide guidance to other researchers with similar goals, and the classifier will help save people's lives.

CHAPTER 7. NEXT STEPS

Due to the limit on time and the many directions that this project could have grown, there are many steps that can be taken to improve the classifier algorithm in the future.

The first set of improvements would help the performance analysis of a classifier. Since the BreakHis dataset is well organized and broken into separate categories, it is possible to group all images from the same patient. Using the separate diagnoses of each image, a prediction can be made on a patient level in addition to an image level. By collecting more data from a patient, a better diagnosis can be made. Md Zahangir Alom et al. used a winner-takes-all method to correctly diagnose 100% of their test patients [21]. This was performed on a different dataset, but it could still be applied to the BreakHis dataset.

Since the training process is stochastic, a higher number of samples would have helped the accuracy distribution plots. If the number of models trained and tested increased from 50 to 100, smaller bins could have been used on the histogram, leading to a more accurate representation of the distribution. Since there were only 10 bins and large variations in data, there were some plots that could be deceiving. For example, **Figure 7.1** shows the accuracy distribution for Model 1.1; there is a sample somewhere in the range of 79% to 83%, but it doesn't show exactly where. By keeping a fixed x-axis for each histogram and using more bins, the resolution of the distribution could have been increased. Having 100 bins spread from 0% to 100% would make it much easier to compare plots.

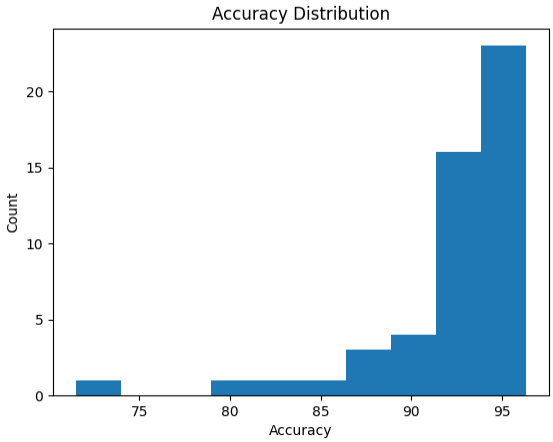


Figure 7.1: Accuracy Distribution for Model 1.1 Architecture

Another improvement for analysis would have been plotting the accuracy and loss over the number of epochs used while training. When training the model, the accuracy and loss per epoch was stored in a variable called “history;” after the best model was found, the history was used to plot the accuracy and loss over the epochs. Unfortunately, the training and testing metrics had stochastic properties. So the plots were difficult to read and unreliable (See **Figure 6.2**). A simple fix would have been to average the accuracy and loss across all of the models instead of using just the best model. That would have made the plots smoother and more informative. Unfortunately, due to time restriction, this was never implemented.

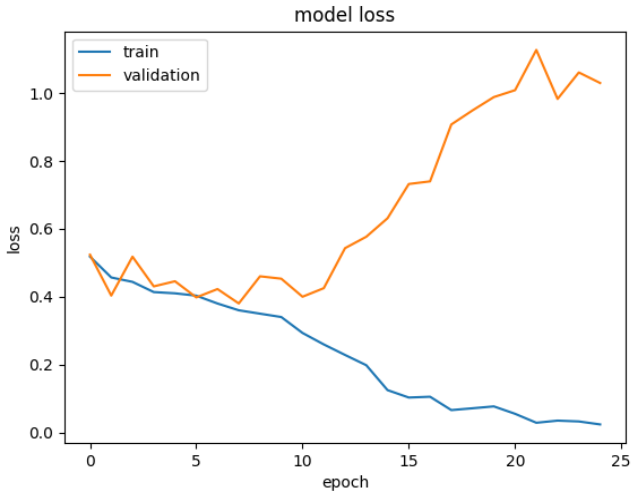


Figure 7.2: Model Loss During Training (Model 1.0)

The next set of improvements would improve the overall performance of the classifier itself. In **Figure 7.2**, the validation loss begins degrading around epoch 10. Training shouldn't continue if the validation loss or accuracy stops improving or starts degrading. When compiling the model with keras, it's possible to define callbacks; code was written to test the callbacks (**Code Block 7.1**), but it wasn't used because of lack of time. The goal of callbacks is to keep the weights from the epoch that performed the best. The Early Stopping callback monitors a metric or loss function of choice and will stop training if the monitored value begins to drop. It has a parameter for patience. Patience determines how many iterations the model can train without making an improvement. If the model stops learning properly, it will stop training and return the model that had the best weights for classifying. The Model Checkpoint callback is more general. But for this use case, it monitors the validation accuracy and saves the weights that achieved the max validation accuracy. The weights are saved to an h5 file and can be reloaded into the model from the temp folder that is generated. Again, the code for the callbacks has not been fully tested, and only one can be used at a time for now.

```
'''Fit data to model using 25 epochs and early stopping'''
# Early Stopping Call back will stop training if the model
# hasn't decreased the validation loss in the last 5 epochs.
# If it stops early, it will restore the weights that had the minimum loss.
earlystopping = tf.keras.callbacks.EarlyStopping(
    monitor ="val_loss",
    mode ="min",
    patience = 5,
    restore_best_weights = True)

# Fit Training Data to Model (Train Data)
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=25,
    callbacks =[earlystopping]
)
```

```
'''Fit data to model using 25 epochs and Model Checkpoint'''
# Create the checkpoint path in the Docker container
if not os.path.isdir(path):
    os.makedirs(path)
```

```
# Saves the best model
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    Filepath = path +'checkpoint.h5',
    save_weights_only=True,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

# Fit Training Data to Model (Train Data)
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=25,
    callbacks =[model_checkpoint_callback]
)

# Reload best weights from the file
model.load_weights(path + 'checkpoint.h5')
```

Code Block 7.1: Early Stopping Callback (Top), Model Checkpoint Callback (Bottom)

As discussed in **Chapter 4.8**, the augmentations that were made did not improve the results. In theory, training with more data should have improved the average classifier's performance. With the way that augmentations were performed, only a small amount of new images could be added to the training data. Ideally, many images would have been added per training sample, rather than just one image for some of the training samples. Limiting the data to always fit the train:validate:test ratio of 0.8:0.1:0.1 was a mistake. It should have been initially extracted from the dataset in that ratio, but adding augmentation should have changed the ratio significantly. Another issue with the augmentation algorithm is that it was limited in the number of ways that it could augment an image. Only 4 extra images could result from the current method. The rotation and flipping should have been based on a function of the number of augmentations. For example, if adding 2 images per training set, the first could rotate 120 degrees and the second 240 degrees. Allowing for more complex augmentations allows for more unique images to be generated.

To fix the imbalance issue, more augmentations could have been used on the benign samples. Since there are 2480 benign samples and 5429 malignant samples, creating twice as many augmentations for benign samples than for malignant samples would improve the balance.

Increasing the number of augmentations would likely improve the performance, and balancing the data would make it less biased towards classifying images as malignant.

If it is not preferable for false positive and false negative rates to match, it might also be possible to incorporate risk when defining the threshold between classes. This idea stems from the classical approach of Bayesian Risk. This would allow for adjustments to be made based on which classification mistake is more preferable. It might be that it is more preferable for the classifier to mistake benign samples as malignant than it is for it to mistake malignant as benign. It is difficult to determine which mistake is more costly because it might have ethical considerations. So this probably would not be implemented, but it could be.

Since the dataset is shuffled before being partitioned into training, validation and testing, some of the samples taken from a single patient will end up in each of the datasets. This means that samples taken from a single tumor can end up in both the training and testing sets. There might be a difference in magnification, or it might be a different section of the same tumor, but this means that the data is biasing the classifier. A potential fix to this is separating the training, validation and testing by patient rather than grouping randomly shuffled images into the datasets. So for example, 80% of patients could be used for training, making it so that those same patients will not appear in the test set.

Problems arise when trying to implement this strategy. For example, there are some cancers that are more rare than others. So it might be that only a few patients have papillary carcinoma. So the classifier will have a lack of exposure to this subclass and struggle to classify it. Even when using binary classification, having a lack of exposure to any of the subclasses during training could make the classifier struggle to correctly identify them as benign or malignant. Another special case are patients with multiple tumors belonging to separate sub-categories. When trying to implement this change, it might just be that the BreakHis dataset is not well suited for separating by patients. It is still an interesting and worthwhile experiment to conduct, as this could help remove bias from the classifier.

Another improvement to be made would be adapting the model to work for multi-category classification, not just binary. If the multi-category classification is able to achieve similar levels of accuracy as binary classification, then it provides more information to doctors about the specific type of tumor that the patient has. If it does not work as well as binary classification, it can still be used after binary classification to provide relevant information to the doctors. This change could help pathologists report the status of malignancy, as well as the proper course of action for treatment options and expected outcomes. Expanding the classifier to work for an arbitrary number of classes and multiple labels would be the ultimate goal, as this would improve the diagnosis and make it much easier to adapt for other purposes.

An interesting special case for classifying sub-categories is the phyllodes tumor. As mentioned in **Chapter 2.1**, about 1 in 4 phyllodes tumors are malignant [8]. The BreakHis dataset labels this as benign because it is less common and not technically breast cancer. It is considered a cancer of the connective tissues (sarcoma), which is not as deadly as breast cancer. Still, it is important that this cancer is caught in the screening process because it can be lethal. This is another case where multi-category classification would be useful for the pathologist. Otherwise, the malignant phyllodes tumors would pass unnoticed.

A final improvement to this report would be to return to VGG16 and finish the preprocessing for the algorithm. This is not a high priority because VGG16 is considered an older algorithm, so it probably would not perform as well as ResNet50 or other algorithms that could be implemented. Finishing the work for VGG16 would improve the completeness of the report and could give other researchers insight to which algorithms work best for the BreakHis dataset. A higher priority might be to implement more modern structures like IRRCNN to see their effectiveness or ways that they can be improved.

BIBLIOGRAPHY

[1] *Breast Cancer Facts and Statistics*. Breast cancer facts and statistics. (n.d.). Retrieved May 7, 2022, from https://www.breastcancer.org/facts-statistics

[2] Lakhtakia, R., & Chinoy, R. F. (2014). A Brief History of Breast Cancer: Part II - Evolution of surgical pathology. Sultan Qaboos University medical journal, 14(3), e319–e322

[3] Early detection is key. Carol Milgard Breast Center. (2022, March 21). Retrieved May 18, 2022, from https://www.carolmilgardbreastcenter.org/for-patients/facts-myths/early-detection-is-key/#:~:text=Women%20whose%20breast%20cancer%20is,year%20starting%20at%20age%2040.

[4] Spanhol, F., Oliveira, L. S., Perirjean, C., & Huette, L. (2019, October 25). *Breast cancer histopathological database (BreakHis) - Laboratório Visão Robótica e Imagem*. Laboratório Visão Robótica e Imagem - Laboratório de Pesquisa ligado ao Departamento de Informática. Retrieved May 7, 2022, from https://web.inf.ufpr.br/vri/databases/breast-cancer-histopathological-database-breakhis/

[5] *Benign tumor: Definition, types, Causes & Management*. Cleveland Clinic. (n.d.). Retrieved May 7, 2022, from https://my.clevelandclinic.org/health/diseases/22121-benign-tumor#:~:text=A%20benign%20tumor%20has%20distinct,other%20parts%20of%20your%20body

[6] *Adenosis of the breast: Sclerosing adenosis*. American Cancer Society. (n.d.). Retrieved May 7, 2022, from https://www.cancer.org/cancer/breast-cancer/non-cancerous-breast-conditions/adenosis-of-the-breast.html

[7] *Fibroadenomas of the breast*. American Cancer Society. (n.d.). Retrieved May 7, 2022, from https://www.cancer.org/cancer/breast-cancer/non-cancerous-breast-conditions/fibroadenomas-of-the-breast.html#:~:text=Fibroadenomas%20are%20common%2C%20benign%20(non,a%20woman%20goes%20through%20menopause

[8] *Phyllodes tumors of the breast*. American Cancer Society. (n.d.). Retrieved May 7, 2022, from https://www.cancer.org/cancer/breast-cancer/non-cancerous-breast-conditions/phyllodes-tumors-of-the-breast.html#:~:text=Phyllodes%20tumors%20(or%20phylloides%20tumors,any%20age%20can%20have%20them

[9] Potter, M. (2018, April 2). *Phyllodes tumors*. Johns Hopkins Kimmel Cancer Center. Retrieved May 7, 2022, from https://www.hopkinsmedicine.org/kimmel_cancer_center/cancers_we_treat/breast_cancer_program/treatment_and_services/rare_breast_tumors/phyllodes_tumors.html#:~:text=Unlike%20breast%20cancer%20which%20begins,or%20cancer%20of%20connective%20tissue

[10] *Tubular adenomas of the breast : American Journal of roentgenology : Vol. 174, no. 3 (AJR)*. American Journal of Roentgenology. (n.d.). Retrieved May 7, 2022, from https://www.ajronline.org/doi/10.2214/ajr.174.3.1740757#:~:text=Tubular%20adenomas%2C%20also%20termed%20pure,that%20vary%20little%20in%20size

[11] Parker, H. (n.d.). *Invasive ductal carcinoma (IDC) & ductal carcinoma in situ (DCIS) breast cancer*. WebMD. Retrieved May 7, 2022, fromhttps://www.webmd.com/breast-cancer/guide/ductal-carcinoma-invasive-in-situ#:~:text=Ductal%20carcinoma%20is%20a%20common,DCIS

[12] *Invasive Lobular Carcinoma (ILC)*. Invasive lobular carcinoma (ILC). (n.d.). Retrieved May 7, 2022, from https://www.breastcancer.org/symptoms/types/ilc

[13] Wells, D. (2018, September 18). *Mucinous carcinoma: Survival rate, recurrence, and treatment*. Healthline. Retrieved May 7, 2022, from https://www.healthline.com/health/mucinous-carcinoma

[14] Potter, M. (2018, April 2). *Papillary Breast Cancer*. Johns Hopkins Kimmel Cancer Center. Retrieved May 7, 2022, from https://www.hopkinsmedicine.org/kimmel_cancer_center/cancers_we_treat/breast_cancer_program/treatment_and_services/rare_breast_tumors/papillary_breast_cancer.html#:~:text=Papillary%20breast%20cancer%20is%20a,These%20are%20called%20papillomas.

[15] The Royal College of Pathologists. (n.d.). Histopathology. The Royal College of Pathologists. Retrieved May 7, 2022, from https://www.rcpath.org/discover-pathology/news/fact-sheets/histopathology.html#:~:text=Histopathology%20is%20the%20diagnosis%20and,clinicians%20manage%20a%20patient's%20care

[16] Bukun. (2020, March 10). *Breakhis*. Kaggle. Retrieved May 7, 2022, from https://www.kaggle.com/datasets/ambarish/breakhis

[17] Centers for Disease Control and Prevention. (2021, September 22). *How is breast cancer diagnosed?* Centers for Disease Control and Prevention. Retrieved May 7, 2022, from https://www.cdc.gov/cancer/breast/basic_info/diagnosis.htm

[18] Benhammou, Y., Achchab, B., Herrera, F., & Tabik, S. (2020). Breakhis based breast cancer automatic diagnosis using Deep Learning: Taxonomy, survey and insights. *Neurocomputing*, *375*, 9–24. https://doi.org/10.1016/j.neucom.2019.09.044

[19] Adam Conner-Simons and Rachel Gordon | CSAIL. (n.d.). *Using AI to predict breast cancer and personalize care.* MIT News | Massachusetts Institute of Technology. Retrieved May 7, 2022, from https://news.mit.edu/2019/using-ai-predict-breast-cancer-and-personalize-care-0507

[20] O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G. V., Krpalkova, L., Riordan, D., & Walsh, J. (2019). Deep learning vs. Traditional Computer Vision. *Advances in Intelligent Systems and Computing*, 128–144. https://doi.org/10.1007/978-3-030-17795-9_10

[21] Alom, M. Z., Yakopcic, C., Nasrin, M. S., Taha, T. M., & Asari, V. K. (2019). Breast cancer classification from histopathological images with inception recurrent residual convolutional neural network. *Journal of Digital Imaging,*. https://doi.org/10.1007/s10278-019-00182-7

[22] *Load and preprocess images: Tensorflow Core*. TensorFlow. (n.d.). Retrieved May 7, 2022, from https://www.tensorflow.org/tutorials/load_data/images

[23] Brownlee, J. (2020, August 27). *How to control the stability of training neural networks with the batch size*. Machine Learning Mastery. Retrieved May 7, 2022, from https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/#:~:text=Batch%20size%20controls%20the%20accuracy,stability%20of%20the%20learning%20process

[24] Brownlee, J. (2020, August 14). *What is the difference between test and validation datasets?* Machine Learning Mastery. Retrieved May 7, 2022, from https://machinelearningmastery.com/difference-test-validation-datasets/

[25] *CNN: Introduction to pooling layer*. GeeksforGeeks. (2021, July 29). Retrieved May 7, 2022, from https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/#:~:text=Pooling%20layers%20are%20used%20to,generated%20by%20a%20convolution%20layer

[26] Biswal, A. (2022, February 21). *Convolutional Neural Network tutorial [update]*. Simplilearn.com. Retrieved May 7, 2022, from https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network

[27] *Intuition of adam optimizer*. GeeksforGeeks. (2020, October 24). Retrieved May 7, 2022, from https://www.geeksforgeeks.org/intuition-of-adam-optimizer/#:~:text=Adam%20optimizer%20involves%20a%20combination,minima%20in%20a%20faster%20pace

[28] Team, K. (n.d.). *Keras documentation: Probabilistic losses*. Keras. Retrieved May 7, 2022, from https://keras.io/api/losses/probabilistic_losses/

[29] Barrera, F. Y. (2021, February 26). *Random initialization of weights in a neural network*. Baeldung on Computer Science. Retrieved May 7, 2022, from https://www.baeldung.com/cs/ml-neural-network-weights

[30] -, G. L. T. (2021, October 5). *Introduction to VGG16: What is VGG16?* GreatLearning Retrieved May 7, 2022, from https://www.mygreatlearning.com/blog/introduction-to-vgg16/

[31] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr.2016.90

[32] *Image classification: Tensorflow Core*. TensorFlow. (n.d.). Retrieved May 7, 2022, from https://www.tensorflow.org/tutorials/images/classification

[33] *How common are breast cancer misdiagnoses?* Hale & Monico. (2021, February 8). Retrieved June 6, 2022, from https://www.halemonico.com/2021/01/13/how-common-are-breast-cancer-misdiagnoses/

APPENDICES

A.1 Documentation for Researchers

When starting this project, a lot of work involved figuring out how to set up the coding environment and learning how to use tools such as Docker and the remote server. Since this process was very time consuming, it made it so that there was less time to dive into the project itself. To make it so that future researchers have an easier time getting started, a document was created on Google Drive that includes information on how to set up the coding environment, use docker, create bash files and start designing a classifier. It also includes useful links that were needed for installation, information, tutorials, and troubleshooting. The document can be accessed with the following link: https://tinyurl.com/2p8z692p

A.2 GitHub Link

All of the code used in this project can be found here: https://github.com/rdalke/Thesis

A.3 Computer Specifications

The time that the classifier takes to perform a task is dependent on the computer being used. This project used Cal Poly's Computer Science department's f35 server. The server has a total of 775 GB of RAM, 256 logical processor cores (CPU model AMD EPYC 7742 64-core), and two Nvidia Tesla V100s. Students that have permissions for this computer are typically allowed 5 GB. However, for projects that require more space, 10 GB can be allotted to a single student. This is a very powerful computer, so training and testing times are relatively short compared to using a personal computer.