Deep Neural Networks and Breast Cancer Detection: Model Architecture and Hyper Parameter Analysis

Tammas Hicks, Liza Richards, Vincent Huynh-Watkins

***Abstract*— The United States possesses a high rate of breast cancer missed diagnosis or misdiagnosis, estimated between 8-13%. Given the importance of early diagnosis for successful treatment, a machine learning model capable of accurately diagnosing breast cancer at rates above 99% would resolve this disparity. This study aims to investigate various approaches and models that could be used to produce such a network.**

Our group members have divided their efforts between examining the efficiency of pre-trained networks, and an untrained network built on PyTorch. The pre-trained networks included DenseNet201, NASNetMobile, and ResNet152. Our members tracked convolutional neural network efficacy by comparing accuracy for each epoch on both training and validation data over each epoch, as well as training and validation loss over each epoch. Comparisons were made between models based on overfitting, as well as ability to accurately diagnose cancer after the final training pass.

Our examination concluded that the pre-trained models were considerably more efficient at producing accurate diagnoses than the untrained model. Of those pre-trained networks, DenseNet121 generated the most accurate model, and was one of the lighter architectures tested. Notable among the pre-trained architectures was NasNetMobile, which failed to perform significantly better than random guessing. The untrained model suffered from overfitting. It is possible that with a much larger data set the untrained model would have been able to predict as efficiently as comparative pre-trained models. It was also found that a lower learning rate and batch size resulted in the highest accuracies produced by each of the models. As for the dropout rate, a value of 0.5 generated the most accurate models.

Keywords—machine learning, medical imaging, cancer detection, image classification

I. Introduction

A major factor in cancer treatment (and survival) is early recognition of disease onset. Machine Learning is then ripe for application to this domain (as has been steadily occurring for many years.) Models such as decision trees (DT), support vector machines (SVM), and artificial neural networks (ANN) have been applied to the field with success [1]. We have chosen to experiment in this domain to compare model accuracy on the same dataset and to understand where certain models may perform well and where others may fall short. Additionally, we seek to understand how architecture differences can affect model performance as well as experiment with hyperparameter tuning. Members of our group created their own deep learning models as well as used transfer learning on pre-trained deep neural networks in order to examine model accuracy, tune hyper parameters, and better understand machine learning architectures and how they may apply better to this problem. For instance, in one experiment we used keras to test different pretrained CNNs such as DenseNet201 and 121, NASNetMobile, and ResNet152. In other experiments we used PyTorch to create custom convolutional neural networks (CNNs) and experiment using the same data. Through these experiments we've been able to gather information regarding the performance of different models and draw some conclusions about the benefits and drawbacks thereof.

Chiefly evident is that pretrained neural networks which use transfer learning perform significantly better than networks trained solely on our training data. This is likely due to the complexity of CNNs and the fact that our dataset was not extraordinarily large in the context of machine learning. Our results also show that certain architectures lend themselves better to this task, including densely connected CNNs and residually connected CNNs whereas, unsurprisingly, with access to decent compute resources, mobile architectures were limited in comparison.

The remainder of this paper is as follows: Section II. gives a brief overview of computer vision as a field as well as the subfield of medical imaging, Section III. discusses our methodology, including how we conducted experiments and the data which we used/how it was handled, Section IV. covers the experiments we undertook, and some brief analysis on them, and Section V. summarizes our conclusions stemming from our experiments.

II. Background

A. Computer Vision

Computer Vision (CV) has its roots in the late 1950s, with researchers beginning to understand the neuron activation associated with vision in cats [2] and Russell Kirsch creating a mechanism to transform images into grids of numbers [3] both innovations laying the foundations for computer vision to begin to take shape over the next decades. Modern CV has come a long way from these humble roots, and our paper focuses on much more contemporary methods.

Since AlexNet won ImageNet in 2012, CV has enjoyed an explosion of use and research, with many new architectures of CNNs being explored as a result. Prior to this innovation, CNNs had been "prohibitively expensive to apply in large scale to high-resolution images" [4]. Today many researchers are focusing on improving CNN architectures, experimenting with different arrangements of feed-forward layers, residual networks and so on. We have chosen to focus on better understanding and testing these models because they are cutting edge and currently hold a large place in computer vision combined with the fact that cancer detection is a highly relevant and important subfield to be working in.

Today, some common applications of computer vision include:

- Optical Character Recognition (OCR)
- Face Detection
- Object Recognition
- Vision-based biometrics
- Medical Imaging

Our focus will be on medical imaging, and specifically classification of medical images. Further discussion of this subfield follows.

B. Medical Imaging

Medical imaging is a burgeoning subfield of computer vision which seeks most commonly to leverage machine learning to aid in medical diagnosis (also known as computer aided diagnosis or CAD [5].) This can be seen in the growing literature and research [1], [6], [7] which aims to improve and further integration of machine learning into medical diagnosis.

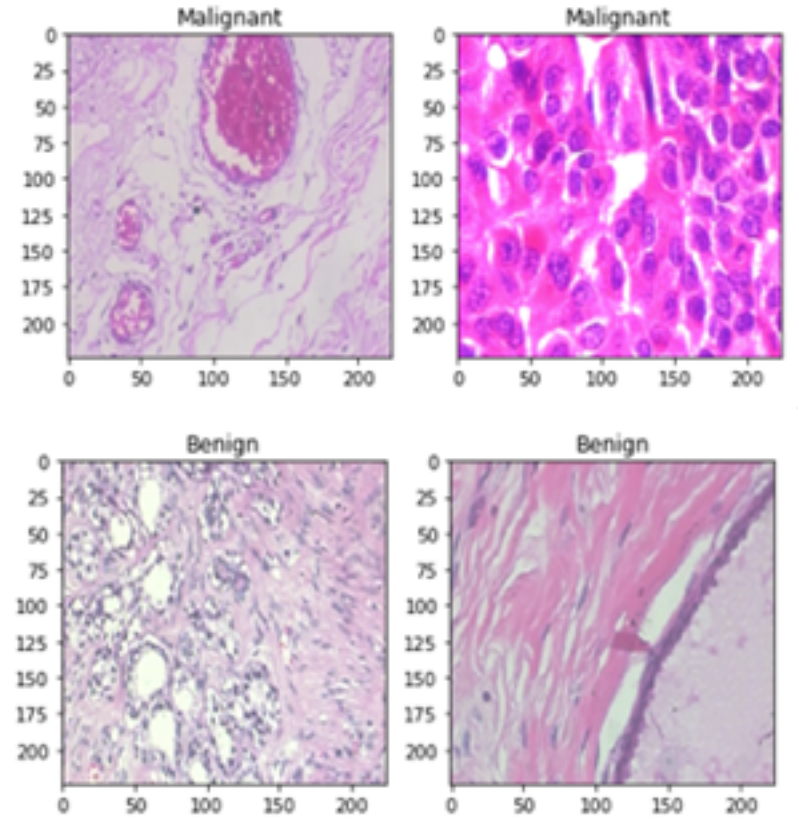
Our work seeks to provide some clarity on factors which may affect the quality of cancer diagnosis when using convolutional neural networks to assess tissue (specifically in our experiments breast tissue) and to act as a follow up on Rakhlin et. al's 2018 article on deep neural networks and breast cancer [8].

III. Methods

Here we will discuss the methodology we used for this paper. This includes how and where we found and handled our data. We will also discuss the different architectures we experimented with.

A. Data

We sourced data from the Breast Cancer Histopathological Image Classification which is composed of over 9,000 microscopic images of breast tumor tissue collected using different magnifying factors, ranging from 40x to 400x. The dataset includes 2,480 benign examples and 5,429 malignant examples. Examples of malignant and benign images can be seen in Figure 1. As noted by the curators of the data, benign refers to "a lesion that does not match any criteria of malignancy– e.g., marked cellular atypia, mitosis, disruption of basement membranes, metastasize…slow growing and remains localized" [9]. In order to use this data with Google Colaboratory notebooks, we uploaded the entire dataset to Google Drive under categories benign and malignant, as indicated by the original archive file provided by The Laboratory of Vision, Robotics and Imaging (VRI) from Brazil's Federal University of Parana. We then used the scikit learn train_test_split function in our notebooks to split the data into train and validation data, which we saved as numpy files in order to



B. Model Architectures

Group member 1, Hicks experimented with implementing an architecture designed by Github user gaurav67890. Modifications were made to the types of layers, and some layers were added to produce greater end accuracy, as well as reduce the considerable overfitting the model was experiencing.

The model originally consisted of three distinct sections, followed by a final linear function. The original architecture will be listed below, in simple format:

Convolutional 2d Network

2d Batch Normalizer
ReLU
Pooling layer

Convolutional 2d Network
ReLU

Convolutional 2d Network
2d Batch Normalizer
ReLU

View Layer (This was used to transform the data into the correct shape for the final layer)
Linear Layer

Convolutional 2d networks are used for sliding over the data at a given step size, in this case a step size of 1, and summing the data from a given area into a single 'feature'. Repeated layers of convolutional networks are used to build combined features into larger features [10]. A single layer might identify edges, while multiple layers would identify complex features like faces. The batch normalizing layers are implemented to speed training time by standardizing the inputs given to the next layer. ReLUs, or 'rectified linear units', are linear functions which output the input directly if it is positive, and output zero otherwise. They are used to help prevent exponential growth in neural networks; important as additional layers are added, and complexity of the data being processed increases. The final linear layer is used to learn the average rate of correlation.

The following changes were made to the network to both improve its accuracy and maintain a reasonable training time while doing so. The multiple ReLU layers were reduced to a single ReLU layer, which was reused for each section, and transformed into a LeakyReLU layer. The LeakyReLU increases the speed at which the data is processed, and prevents the 'dying ReLU' problem, wherein a large number of outputs return 0 [11].

Given the relative complexity of identifying cancerous cells, due to the numerous features such as hairiness, clumping, etc., researcher Hicks chose to add convolutional layers and batch normalizing layers to the network. The intent was to increase the complexity and size of features that the network was capable of recognizing.
Linked below is the ipynb file used. Credit for the original code goes to gaurav67890.
https://colab.research.google.com/drive/12s6G_imFEZgF93QhtdwtrBGeogiTxOfw?usp=sharing

Group member 2, Richards experimented with an architecture designed by Abhinav Sagar [22] with modifications made to several hyperparameters involved in multiple pre-trained deep neural networks. Adjustments made to the hyperparameters have effects on the accuracies produced by the different neural networks.

In machine learning, hyperparameters are values that cannot be estimated from the data because it is external to the model. Hyperparameters control the overall behavior of the model [16]. Tuning hyperparameters is done in order to reach the model's highest performance as well as minimize the loss function without changing the actual model. Some of the most common hyperparameters are the dropout rate, learning rate, batch size, and the number of epochs.

Dropout is used to address the common problem of overfitting in deep neural networks. It refers to ignoring random hidden and visible units during the training phase [17]. As a result, they are not included to the activation of units that are downstream from them and produces a somewhat thinned network from it. Dropout tuning works better on larger networks because there is room for the model to learn more independent representations [17]. When tuning the dropout rate, values between 0.2 and 0.8 tend to be the best to choose from. A value that is too low will have a very minimal effect on the model performance and a value that is too high can result in network under-learning [17].

The learning rate is considered to be the most important hyperparameter in the configuration of a neural network. The learning rate controls how quickly the neural network updates its parameters [18]. It also controls how much a model should change or adapt based on the loss gradient. Values between .01 and .0001 tend to be the optimal range of choices. A learning rate that is too high can lead to the model converging too quickly, failing to converse, or even diverging [19]. A learning rate that is too low can cause the process to get stuck because the gradient descent is too slow.

Batch size refers to the number of training examples that are used in one iteration before updating internal node parameters [20]. It can be compared to a for loop that iterates over a number of samples and makes predictions. The data set is often divided into a certain number of batches because the entire set of data cannot be passed into the neural network at once [21]. Larger batch sizes tend to do a worse job at generalizing to the data. Smaller batch sizes lead to faster convergence to better solutions, but it is not guaranteed that it will converge to the global optima [21]. Batch sizes usually start at 16 but it can also 32, 64, 128, or 256.

The number of epochs is the number of times the learning algorithm passes through the training dataset. One epoch is equivalent to when a dataset as a whole is passed forward and backward through the neural network once [20]. The number of epochs can represent how diverse the data is. Often times, the larger the number of epochs the better. Large numbers of epochs can reduce the amount of error and increase the accuracy produced by the model. There is no limit to the number of epochs, however the larger the number the longer the algorithm will take to run. In addition, too large of a number of epochs may lead to the model over-fitting the training data [20].

Group member 3, Huynh-Watkins experimented with transfer learning and various pre-trained deep neural networks, including, Densenet201, Densenet121, MobileNet, NASNetMobile, and RestNet152.

Both densenet implementations are densely connected convolutional neural networks, as initially explored by Huang, Liu, Maarten, and Weinberger [12]. The general idea of a densely connected CNN is that they can be deeper, more accurate, and more efficient to train if they contain shorter connections between layers close to the input and close to the output, e.g., their connections are denser [10]. This is illustrated in Figure 2. DenseNet connects each layer to every other subsequent layer using feed-forward connections. This solution addresses the vanishing gradient problem, strengthens feature propagation, encourages feature reuse, and reduces the number of parameters [10].

ResNet152 is a deep neural network with 152 layers which learn residual functions with reference to the layer inputs [13] and are easier to optimize than other deep neural networks. The residual network architecture adds in “shortcut connections” which are feedforward connections, which make the network easier to optimize and therefore more efficient and less prone to the degradation problem than other implementations, such as VGG19 [11].

Figure 1

MobileNets are streamlined neural networks designed to be used for mobile and embedded vision applications [14]. The MobileNet architecture is based on depth wise separable convolutions [12] and has a smaller model size and lower complexity (compared to other deep neural networks) which makes it an ideal candidate for applications with low access to computing resources. Proposed applications include object detection, fine grain classification, and large-scale geo-localization.

NASNet is an architecture which automates the network architecture engineering, e.g., it searches on a small dataset for the best model for a given task which can then be extended to a larger dataset [15]. This method finds the best convolutional layer and then stacks together more copies of that layer to design a convolutional architecture.

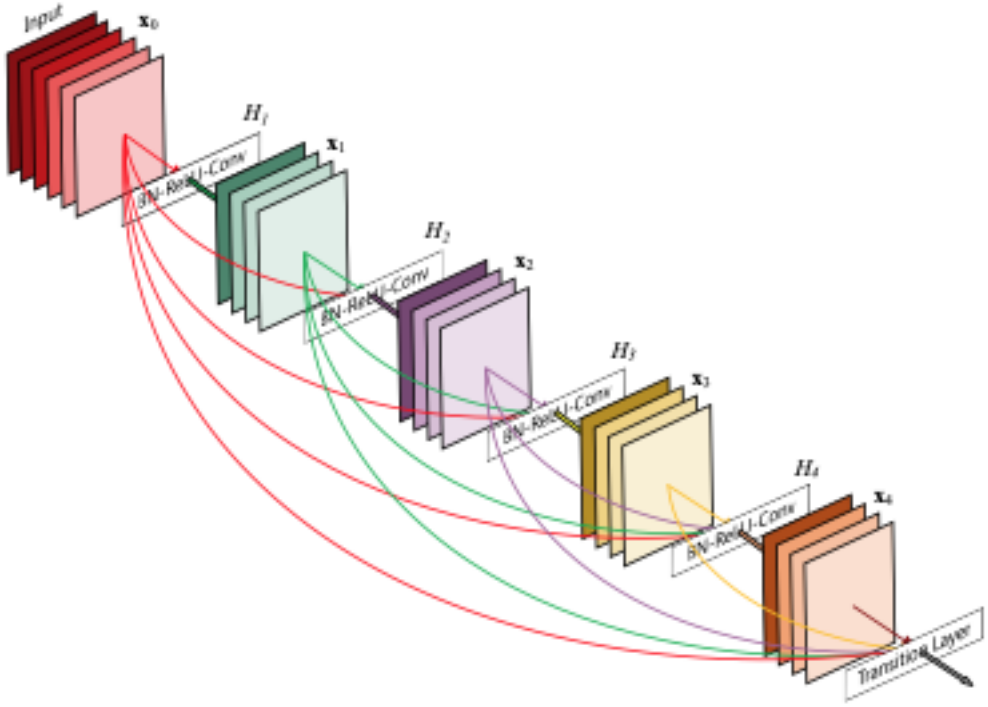


Figure 2 [3]

IV. Experiments

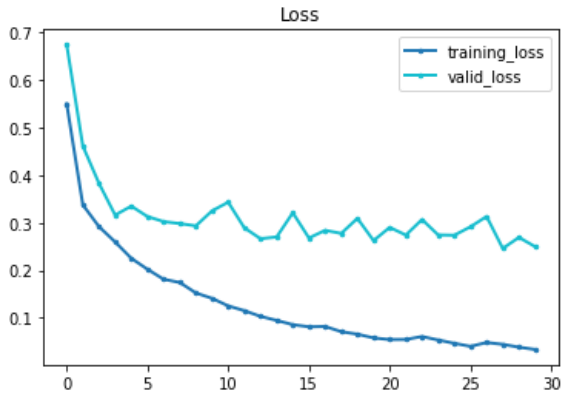
The original construction of the neural net produced relatively low results, identifying the correct target group on the validation set around 78% of the time. Training loss was also high, at 33% after the final epoch. Originally the construction was set to 10 epochs, and a learning rate of 0.001. While the training accuracy was close to 95%, the evaluation accuracy fluctuated wildly during the last few epochs.

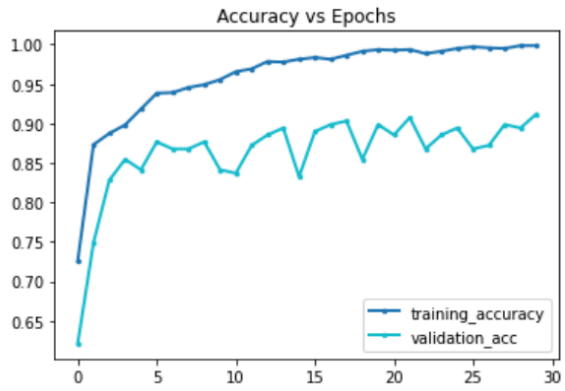
The un-altered model was suffering significantly from overfitting. If the number of epochs was increased significantly the training accuracy would eventually reach 100%, while the evalutaion data began to thrash around 85%. Attempts to reduce the learning rate and increase the number of epochs to solve the issue were unsuccessful. The model still experienced thrashing at later epochs, and would not achieve above 87%. Altering the structure of the model by changing the ReLU to a LeakyReLU, as well as by adding additional sections of stacked convolutional, batch normalization, and ReLU layers. As discussed above, the LeakyReLU can be a solution to dropout problems. The hypothesis around the additional convolutional layers, as well as the accompanying batch normalization layers was that the features which define malignant or benign cells are relatively complex, the repetition of convolution would allow the model to identfiy these features. The hypothesis proved correct as, when it was wiped and trained again, the model experienced peaks in validation performance above 91%. However, the model was still thrashing considerably during the last ~30% of its training epochs.

To resolve this, the training code was modified, and a stop condition was added. If the model performed above a satisfactory value variable, a percentage determined by the experimenter, on the validation data, then the model would quit training early. While this did not solve the issue with thrashing, it did allow the training to be caught at the peak of a thrash. The drawback to this approach was that if the model would hypothetically have achieved some validation percentage value greater than what the experimenter had set the satisfactory performance value to during a later epoch, stopping early would miss this increase in performance. A solution to this is that repeated runs of the model on the training data gives the experimenter a reasonable certainty of what the highest achievable validation percentage was, and they could set the satisfactory performance parameter to that value.

However, results were not consistent. Runs would achieve the desired 91% validation threshold between 7 and 39 runs. There were still issues with thrashing in the validation data, and overfitting indicated by training success above 99% when the validation success finally achieved satisfactory results. It's likely that this is not a bad model for identifying malignant cells in breast tissue samples, but that it would be improved by a greater dataset size. The primary advantage that the other two approaches had above this is that they used pre-trained models with greater recognition ability induced by a dataset orders of magnitude larger than this one. It would be of interest to this researcher to collect further samples, and attempt training this model on that larger set.

Below are the results of the final run done with this architecture.





As you can see from the charts, while there is a general upwards trend in the average accuracy of the validation data, it experiences considerable thrashing. The high level of loss in the validation data, compared to the at of the training data also indicates a relatively inefficient system at time of completion.

A. Hyperparameter tuning

Modifications to the neural networks and their hyperparameters are made to the original CNN architecture. Firstly, applications were loaded from Keras in order to implement some pre-trained neural networks. Then the hyperparameters involved in these models were adjusted to test how the accuracies would change.

Looking at the results in the table below to the left, it is evident that for each model, a dropout rate of 0.5 produced the best validation accuracies. However, changing the dropout rates to 0.2 or 0.8 in each of the models led to different results in each. For example, in DenseNet121 and MobileNet, the accuracy produced when a dropout rate of 0.2 was used is the same as the accuracy produce by the 0.5 dropout rate model. However, in DenseNet201, the accuracy produced by the 0.2 dropout rate model is the same as the 0.8 model. In ResNet152 and NASNetMobile, the accuracies of the 0.2 and 0.8 dropout rate models are less than that of the 0.5 dropout rate model.

Model	Dropout	Accuracy
MobileNet	0.5	98.21%
MobileNet	0.2	97.85%
MobileNet	0.8	97.85%
NASNetMobile	0.5	68.46%
NASNetMobile	0.2	56.27%
NASNetMobile	0.8	66.67%
ResNet152	0.5	99.28%
ResNet152	0.2	98.93%
ResNet152	0.8	98.57%

Model	Dropout	Accuracy
DenseNet121	0.5	99.64%
DenseNet121	0.2	99.64 %
DenseNet121	0.8	98.21%
MobileNet	0.5	98.21%
MobileNet	0.2	97.85%
MobileNet	0.8	97.85%

Model	Dropout	Accuracy
DenseNet201	0.5	98.21%
DenseNet201	0.2	97.85%
DenseNet201	0.8	98.57%

Looking at the tables below, it can be concluded from each of the models that a higher learning rate resulted in lower validation accuracies. This is consistent with the fact that higher rates are worse at predictions. In all of the different models, an increased batch size of 32 from 16 also resulted in a decrease of the validation accuracy. From the results produced by each of the models, it is evident that there is a correlation between the learning rate and the batch size. A smaller learning rate and a smaller batch size allow for better network training and validation accuracy.

Model	Learning Rate	Accuracy
DenseNet121	0.0001	99.64%
DenseNet121	0.01	97.49%
DenseNet201	0.0001	98.21%
DenseNet201	0.01	97.85%
MobileNet	0.0001	98.57%
MobileNet	0.01	97.49%
NASNetMobile	0.0001	68.46%
NASNetMobile	0.01	58.78%
ResNet152	0.0001	99.28%
ResNet152	0.01	98.93%

Model	Batch Size	Accuracy
DenseNet121	16	99.64%
DenseNet121	32	97.85%
DenseNet201	16	98.21%
DenseNet201	32	98.21%
MobileNet	16	98.21%
MobileNet	32	97.85%
NASNetMobile	16	68.46%
NASNetMobile	32	63.08%
ResNet152	16	99.28%
ResNet152	32	98.93%

Based on the results in the table below, it can be concluded that there is a relationship between the number of epochs and the model performance. When the number of epochs is decreased to 10 from 20, the validation accuracy of each of the models decreased quite a bit. When the number of epochs was increased to 20, the accuracy increased with it. From this we can also assume that the loss decreased as the number of epochs increased as well.

Model	Epoch #	Accuracy
DenseNet121	20	99.64
DenseNet121	10	96.77
DenseNet201	20	98.21
DenseNet201	10	96.06
MobileNet	20	98.21
MobileNet	10	97.49
NASNetMobile	20	68.46
NASNetMobile	10	45.88
ResNet152	20	99.28
ResNet152	10	96.54

B. Transfer Learning Experiments

To test different CNN architectures against each other easily, we also used pre-trained networks and transfer learning in order to easily implement and subsequently measure performance on the aforementioned architectures. Using Keras, this process included writing the code to load data and doing model setup and then running training. [1]To compare the different models, it was as easy as changing which architecture to load from Keras.applications. Below is a table indicating model performance on validation data as well as model size. As seen in the table, the highest performing models were DenseNet 121 and ResNet152. The model that performed the worse was NASNetMobile. It is unclear why this model performed so poorly, as it overfit quite significantly, with a training accuracy of 91.41%. Some conjecture may be that the NASNet pretrained model was not well suited to this task, given that it learns on a small dataset first and then a CNN is created using the highest performing convolution layer from this initial dataset. Without more information on the pretrained model it is difficult to directly answer this question, however it seems plausible that the weights learned from pre-training on ImageNet do not represent the weights needed to do well on our problem and that the architecture learned by NASNet created an inflexibility which it was

Model	Validation acc	size
DenseNet201	98.21%	80mb
DenseNet121	99.64%	33mb
MobileNet	98.21%	16mb
NASNetMobile	52.69%	23mb
ResNet152	99.28%	232mb

unable to overcome in the given number of epochs (70.) Figures 3 and 4 demonstrate the accuracy and loss over epochs respectively for NASNetMobile, with the curves clearly displaying that the model was having difficulty learning. The accuracy curve shows signs of overfitting, with the model learning the training data well but not generalizing to validation nearly at all.

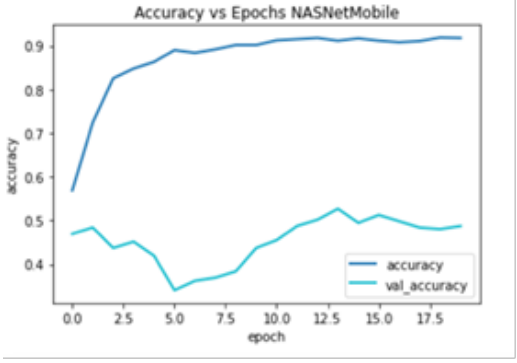


Figure 3

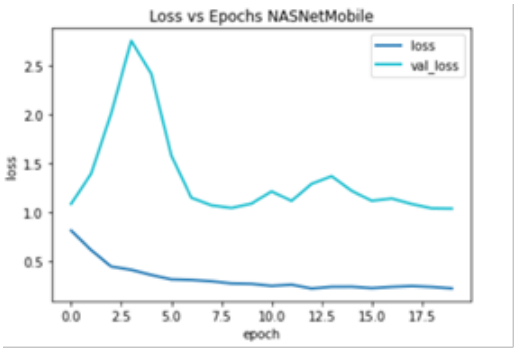


Figure 4

The confusion matrix for NASNetMobile seen in Figure 5 also highlights how poorly the model did on validation data, with a significant number of predictions being incorrect. Interestingly, the model was better at predicting when an image contained a malignant tumor than when an image contained a benign tumor, indicating that the model may have learned how to identify malignant tumors but that it had not yet at the conclusion of training learned how to differentiate between malignant and benign tumors well.

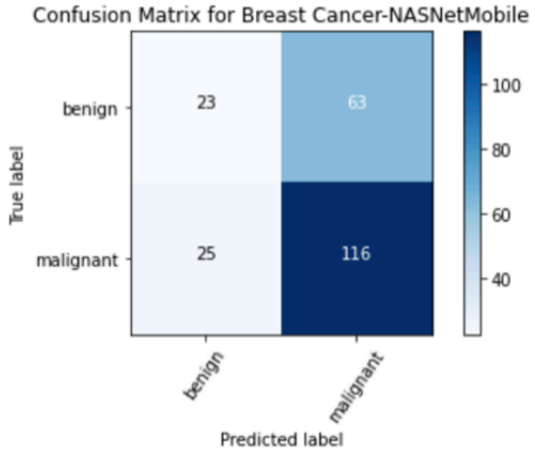


Figure 5

In contrast, the learning curves of the DenseNet 121 model (figures 6 and 7) show that the model continued learning throughout the training and did not seem to overfit. This would lead to the potential conclusion that the DenseNet architecture pre-trained on ImageNet was flexible enough to train and learn on our dataset of 9000 images and be able to learn to recognize tumors as well as to differentiate between benign and malignant.

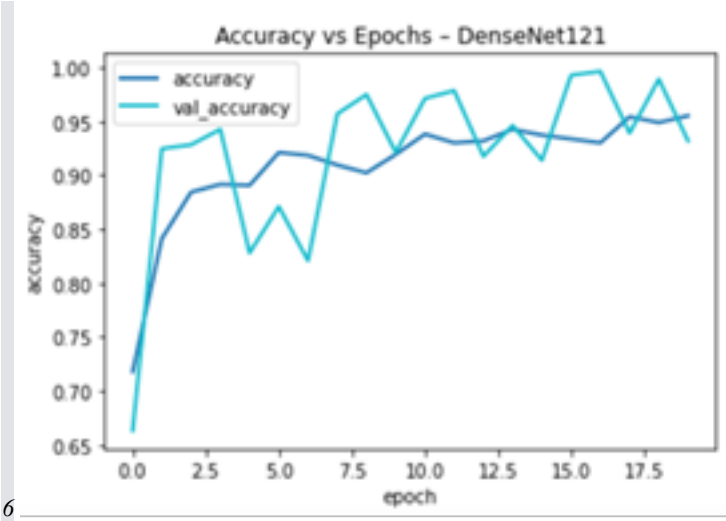


Figure 6

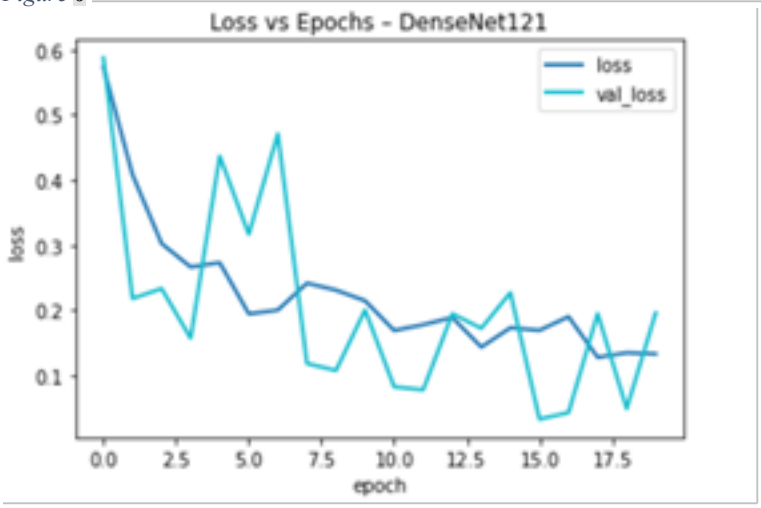


Figure 7

The confusion matrix for DenseNet121 in Figure 8 supports this claim, with the model doing quite well in predicting both benign and malignant tumors. It is notable, however, that similarly to NASNetMobile, DenseNet121 overpredicted malignant tumors compared to benign tumors. Although DenseNet faired far better than NAS net, this is notable in that it may be a result of an imperfectly balanced dataset, with over double the number of malignant examples. That said, this may make some sense in the context of attempting to diagnose cancer, with a false negative having potentially much larger consequences than a false negative. With a larger dataset and longer amount of time to train, the models may be able to overcome this imbalance. That said, it may be favorable to slightly rebalance the dataset by taking out a random portion of the dataset with the malignant label, creating a dataset with closer to 55% malignant 45% benign, instead of roughly 66% malignant and 33% benign.

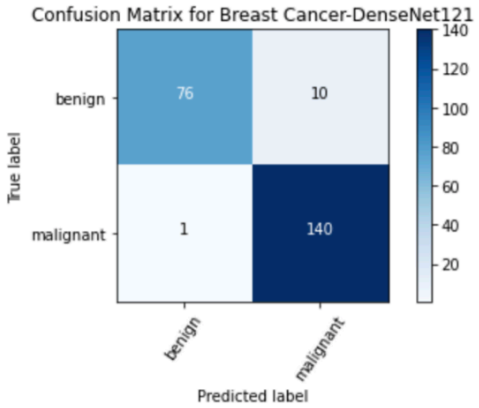


Figure 8

V. Conclusions

For group member 2, Richards can conclude that a decrease in both batch size and learning rate lead to better model performance for each of the different models with the evidence being the higher validation accuracies. Future research regarding these values with these networks could be to continue increasing the values until the data fails to converge. A dropout rate of 0.5 was also proven to produce the highest accuracies for each of the different neural networks. However, an increase or decrease in the dropout rate had different effects on the networks. Future research regarding the relationship between the change in dropout rate and the performance of the model can be done in order to determine why the same dropout rates have different effects on the different models. It was also concluded that a greater number of epochs leads to an increase in the validation accuracy of each of the models. For future research regarding the number of epochs, the number could continue to be increased until there is no longer an increase in accuracy. You can test how many epochs it would take to reach the limit of the model.

VI. References

[1] Kourou, "Machine learning applications in cancer prognosis and prediction," *Science Direct,* vol. 13, pp. 8-17, 2015.

[2] D. H. Hubel and T. N. Wisel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of Physiology,* vol. 148, pp. 574-591, 1959.

[3] "Fiftieth Anniversary of First Digital Image Marked," NIST, 24 May 2007. [Online]. Available: https://www.nist.gov/news-events/news/2007/05/fiftieth-anniversary-first-digital-image-marked. [Accessed 3 June 2021].

[4] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep neural networks," *NIPS,* 2012.

[5] M. N. Werrick, y. Yang, J. G. Brankov , G. Yourganov and S. C. Strother, "Machine Learning in Medical Imaging," *IEEE Signal Processing Mag. ,* vol. 27, no. 4, pp. 25-38, 2010.

[6] M. A. Myszczynska, P. N. Ojamies, A. M. Lacoste, D. Neil, A. Saffari, R. Mead, G. M. Hautbergue, J. D. Holbrook and L. Ferraiuolo, "Applications of machine learning to diagnosis and treatment of neurodegenerative diseases," *Nature Reviews Neurology,* vol. 16, pp. 440-456, 2020.

[7] J. G. Richens, C. M. Lee and S. Johri, "Improving the accuracy of medical diagnosis with causal machine learning," *Nature Communications,* vol. 11, 2020.

[8] A. Rakhlin, A. Shvets, V. Iglovikov and A. Kalinin, "Deep Convolutional Neural Networks for Breast Cancer Histology Image Analysis," *International Conference Image Analysis and Recognition,* 2018.

[9] "Breast Cancer Histopathological Database (BreakHis)," Laboratory of Vision, Robotics, and Imaging, [Online]. Available: https://web.inf.ufpr.br/vri/databases/breast-cancer-histopathological-database-breakhis/. [Accessed 25 May 2021].
[10] J. Brownlee, "Machine Learning Mastery," 17 Apr 2019. [Online]. Available: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/. [Accessed 6 Jun 2021].
[11] D. Liu, "Medium.com," 29 Nov 2017. [Online]. Available: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7#:~:text=Leaky%20ReLU%20has%20two%20benefits,to%200%20makes%20training%20faster..
[12] G. Huang, Z. Liu, L. Van Der Maarten and K. Q. Weinberger, "Densely Connected Convolutional Networks," *IEEE Conference on Computer Vision and Pattern Recognition,* pp. 2261-2269, 2017.
[13] K. He, X. Zhang, R. Shaoqing and S. Jian, "Deep Residual Learning for Image Recognition," *Microsoft Research,* 2015.
[14] A. G. Howard, M. Zhu , B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreeto and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *Google Inc. ,* 2017.
[15] B. Zoph , V. Vasudevan, J. Shlens and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," *Google Brain,* 2018.
[16] P. Radhakrishnan, “What are Hyperparameters? and How to tune the Hyperparameters in a Deep Neural Network?,” *Medium*, 18-Oct-2017.
[17] J. Brownlee, A Gentle Introduction to Dropout for Regularizing Deep Neural Networks,” *Machine Learning Mastery*, 06-Aug-2019.
[18] Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, *JMLR*, vol. 15, pp. 1929-1958, 2014.
[19] H. Zulkifli, “Understanding Learning Rates and How It Improves Performance in Deep Learning,” *Medium*, 27-Jan-2018.
[20] J, Brownlee, “Difference Between a Batch and an Epoch in a Neural Network,” *Machine Learning Mastery*, 25-Oct-2019.
[21] K.Shen, “Effect of batch size on training dynamics,” *Medium*, 19-Jun-2018.
[22] A. Sagar, “Convolutional Neural Network for Breast Cancer Classification,” *Medium*, 26-Nov-2019. https://towardsdatascience.com/convolutional-neural-network-for-breast-cancer-classification-52f1213dcc9.

Page Break