# Classifying Ultrasound Images for Breast Cancer Detection Using a Convolutional Neural Network

# Jesse St. John and Margaret Barker 2024 | COMP4531

# Abstract

As of 2023, breast cancer is the most common occurring cancer in the world and is the second leading cause of cancer-related deaths among women in the United States (1). While mammograms are typically the first step and gold standard when it comes to screening for breast cancer, there are times when an ultrasound is needed. If the breast tissue is too dense, if the patient is pregnant (and therefore needs to avoid the radiation from a mammogram), of if the patient is under 25 are among some of the reasons that an ultrasound may be used as an additional screener for breast cancer. In addition, an ultrasound may be ordered after a mammogram to gain extra insight on masses discovered during the mammogram (2).

Ultrasounds make use of high-frequency sound waves to capture an image of the body’s internal organs in real time. This type of imaging can show specialists movement of the organs and blood flow which can help them distinguish between solid masses or fluid-filled cysts (3). Like mammograms, these procedures are done by a trained technician. The images are then forwarded to a specialist to interpret, and then depending on the set up of the patient’s healthcare team, that information may get passed through other necessary channels before getting to the patient. As one can imagine, after already receiving a mammogram and now waiting on yet more information for a crucial diagnosis, this can be a stressful, frustrating time for both patients and providers.

This project seeks to develop a deep learning model that can accurately classify these types of breast ultrasound images for cancer detection. Utilizing a convolutional neural network (CNN) to analyze the images and recognize the inherent patterns can help reduce the time it takes to get critical information to the patient, minimize human diagnostic error, and refocus provider resources into patient care.

# Problem Statement

A CNN will be trained to recognize and classify breast ultrasound images into three different categories: normal, benign, or malignant. The images were obtained from the [Kaggle Breast Ultrasound Images Dataset](https://www.kaggle.com/datasets/aryashah2k/breast-ultrasound-images-dataset) (4). The library contains 788 images, which are already separated into their respective categories. We will use a pre-built **EfficientNetB0 (developed by Google in 2019)** to obtain a benchmark accuracy value to compare our custom CNN model against. We will then evaluate our custom CNN model’s accuracy and identify ways to optimize the model regarding classification accuracy and overall performance.

# Problem Set Up

The images in the Kaggle library are separated into their proper classification categories. Additionally, each ultrasound image is accompanied by a high contrast image (indicated by a file name with a suffix of ‘\_mask’), as seen below. The high contrast images were created by the Kaggle dataset authors using Matlab and will not be used for this classification problem (4).

A collage of images of a person's body

Description automatically generated

Pictured from top to bottom are the classes: benign, malignant, and normal. The left column shows the raw ultrasound image. The right column shows the high contrast mask image.

We will only be using the raw ultrasound images for all modeling and training purposes. Thus our a sample of our data set looks like the following:

A collage of images of a person's uterus

Description automatically generated

Each ultrasound image in the library is 500 x 500 pixels in PNG format. The images were resized to 224 x 224 pixels to increase time efficiency. Each image was also normalized prior to being split into training, validation, and test sets. Labels for classification (normal, benign, or malignant) were passed with each image into the model for proper training. Training took place over 25 epochs, each with a batch size of 32.

The goal of this CNN model is to load in a series of single 500 x 500 PNG images, resize them to 224 x 244 pixels, normalize the images, feed each image through the CNN, and then have the model output the image with its predicted classification label compared to its true classification.

The images were split into train/validation/test sets as follows:

Training set: 467 images

Validation set: 195 images

Test set: 118 images

# Problem Exploration

## Data Exploration:

Our first step was to find a suitable dataset for modeling. The Kaggle Breast Ultrasound Images Dataset was used due to the hundreds of images provided in an already categorized manner. According to the website,

*“The data collected at baseline include breast ultrasound images among women in ages between 25 and 75 years old. This data was collected in 2018. The number of patients is 600 female patients. The dataset consists of 780 images with an average image size of 500\*500 pixels. The images are in PNG format.”*

Mammography and/or additional testing were used by doctors and other professionals to correctly classify the images as either normal, benign, or malignant. As stated previously, the masked images were generated using the raw ultrasound images and were thus discarded for our modeling purposes.

We created a Custom Data Generator to load and pre-process the ultrasound images. Essentially, this generator loads in only the ultrasound images, assigns the images the appropriate classification label based on their directory, and creates batches and subsets of our data. This Custom Data Generator also allows us to shuffle the order of the filenames (with their corresponding labels) at the end of each epoch to ensure the model does not learn the order of the data.

Before we began training our model, we created a custom callback to display the images, defined a learning rate scheduler, and created the data generators to split our data into the training, validation and test sets. The learning rate scheduler was used to assist in adjusting the learning rate during training. This scheduler modifies the learning rate at the end of each epoch depending on the learning rate value of the epoch that came before. The custom callback was a crucial part of our training process as it allowed us to visualize the model’s performance. At the end of each epoch, the ImageDisplayCallback retrieves a batch of images and labels from the validation data, makes predictions using the model, and then displays the images with both their true and predicted label. The output will vary from epoch to epoch, but the general layout looks like the following:

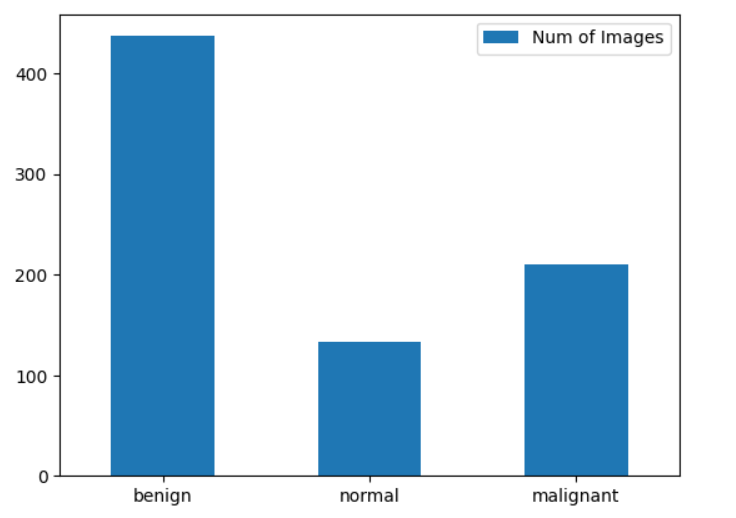
A close-up of an ultrasound

Description automatically generated

Then we employed callbacks to automate the fine tuning of the training process for both the EfficientNetB0 as well as our Custom CNN. These three additional callbacks were:

* EarlyStopping callback prevents overfitting by stopping the training process when no improvement is seen in validation loss.
* ModelCheckpoint callback saves the best model during training based on validation loss.
* ReduceLROnPlateau callback adjusts the learning rate dynamically to ensure better convergence.

A key feature of our dataset that came into play during our model development was the distribution of the classes, as indicated in the figure below. There are more cases of benign in our dataset than the other two classes combined. As we developed our model, this was an issue we worked to address.



## Model Development

### EfficientNetB0 Architecture

To ensure proof of concept, we employed a pre-built model called **EfficientNetB0.** EfficientNet is used for image recognition, was developed by Google in 2019, and has achieved high end performance on benchmark datasets, such as ImageNet. It offers several advantages for image recognition due to its design principles, which balance accuracy and computational efficiency. EfficientNetB0 has fewer parameters compared to other image recognition models like ResNet or Inception to scale down computational requirements. This is achieved using mobile inverted bottleneck convolutions and compound scaling within the model. The resulting squeeze-and-excitation optimization helps the model to adaptively recalibrate its feature maps, making it more robust to variations in the input data. EfficientNetB0 is the baseline model in a family of models referred to as the EfficientNet family. This means that the overall model can be scaled up or down, depending on the requirements of the task. This model also includes regularization techniques, such as batch normalization and drop out, which can help improve generalization capabilities (6, 7).

Due to its practical applications, we chose to use EfficientNetB0 as our base model to show that what we were attempting was feasible. This also gave us invaluable insight into how our own custom CNN model should work. The architecture of EfficientNetB0 can be seen below (5).

A diagram of a company

Description automatically generated with medium confidence

A diagram of a diagram

Description automatically generated with medium confidence A diagram of a process

Description automatically generated

Figures: EfficientNetB0 architecture overview (top). Process inside blocks from top figure (bottom left). Process inside each module from bottom left figure (bottom right).

### EfficientNetB0 Performance

On our initial run of the EfficientNetB0 model with our training and validation datasets, we connected this base model to a global pooling layer, a dense fully connected layer with 512 neurons and a ReLU activation function, followed by a dropout layer (set at 0.5), and then finally the SoftMax layer to achieve the predictions for the three classes; benign, normal, and malignant. The model was then compiled utilizing the Adam optimizer. The below plots show EfficientNetB0’s accuracy and loss on the training and validation data.

A graph of a model

Description automatically generated with medium confidence

EfficientNetB0 initially learned quickly on our data, but then stalls around 55% accuracy. The model’s accuracy on the validation data remained the same throughout. During this run we discovered that this model is still computationally expensive and is not producing a high enough accuracy on our data. With additional tuning of hyperparameters, we could have possibly changed this outcome, however, we turned our focus to our customized model.

### Custom CNN Architecture

We designed our custom Convolutional Neural Network specifically for image classification tasks making use of Keras’ Sequential API. The architecture begins with an input layer that receives images of a specified height and width (244 x 244 pixels) with three color channels (RGB). This input is fed into a series of convolutional layers as seen below.

A diagram of a diagram

Description automatically generated

Each convolutional layer applies a set of filters to the input image; the number of filters is indicated in the diagram underneath each Conv2D layer. These filters detect various features in the image such as edges, textures, and more complex patterns as the depth of the network increases. Following the filters is a ReLU activation function which introduces non-linearity to the model, enabling it to learn from complex data. Each convolutional layer is followed by max-pooling layer to reduce the spatial dimensions of the feature maps, effectively summarizing the presence of features in patches of the feature map and making the detection process more efficient. This down-sampling of the feature maps helps reduce the computational load. Additionally, this can help control overfitting by making the model less sensitive to small translations in the input image.

This series of convolutional and max-pooling layers is repeated multiple times, with each successive layer learning increasingly abstract and complex representations of the input data. The final convolutional layer captures the highest-level features, which are essential for the classification task at hand. Before the model moves on to the fully connected (or dense) layers, the feature maps from the final max-pooling layer are flattened to a one-dimensional vector to serve as the input for these dense layers. The first dense layer consists of 1024 neurons, each applying a ReLU activation function, followed by a drop out layer to mitigate the overfitting by randomly setting a fraction of the input units to zero during training. Then L2 regularization is applied to the layer to further prevent overfitting and ensure the model will generalize well to new, unseen data. A second dense layer with 512 neurons, dropout, and L2 regularization follow. Both dropout layers are set at a rate of 0.5. These two dense layers further process the extracted features and learn to combine them in ways that are useful for classification.

The model concludes with an output layer consisting of three neurons, corresponding to the three classes in our classification task. This layer utilizes a SoftMax activation function, which converts the outputs to probabilities, allowing the model to predict the likelihood of each class for a given input image.

### Custom CNN Performance

Once we had our custom CNN built and callbacks defined, we were ready to train the model on our dataset. The plot for our initial run can be seen below.

A comparison of graphs and diagrams

Description automatically generated with medium confidence

Our model appears to be outperforming the out-of-the-box model. When we ran this model on the test set, the accuracy was 83%. The gap between the training and validation accuracy widens with each epoch. This could indicate that our model is possibly overfitting. The plots of the loss converge to similar values and suggest that our model is well tuned. We generated a confusion matrix analysis on our model’s performance on the test data to further evaluate performance.

A diagram of a number of different types of labels

Description automatically generated with medium confidence

Even though our overall accuracy percentage was better than EfficientNetB0, when we looked at the confusion matrix, we saw that our model was only predicting for classes of benign. This was likely due to the imbalance of classes in our dataset, something we noticed in the early stages of our data exploration.

Knowing we started with a relatively small, imbalanced dataset we also looked at the Precision-Recall Curve for our model. Understanding this information is crucial for further improvements of the model, especially when placed back into the context of the problem we are attempting to solve: breast cancer classification. It is important that the model be accurate and have high recall. The Precision-Recall Curve will visualize the tradeoff between precision and recall. High precision corresponds to a low false positive rate and high recall corresponds to a low false negative rate.

A graph of different colored lines

Description automatically generated

The plot above also gives us the Average Precision (AP) of each class: benign is 0.56, normal is 0.16, and malignant is 0.28. The scores for normal and malignant are quite low, indicating the model often fails to correctly identify these cases. The score for benign indicates the model has a moderate performance in identifying those cases. This makes sense given the information from the confusion matrix. Having a model that only learns one class well and then predicts all subsequent unseen data to be that class amounts to little more than a guessing game and is not sufficient. The 83% accuracy score means nothing when the model is obtaining that score only by chance. Thus, we needed to address this imbalance of classes in our dataset to improve model performance.

### Calculating Class Weights

We utilized SciKit Learn’s compute\_class\_weight module to calculate appropriate weights for each class. These weights are based off the distribution of the classes within our training set. After calculation, the weights are assigned to the appropriate class label within a dictionary that can then be passed to the model during fitting. The weights were assigned as followed (rounded to 4 decimal places here for convenience):

* Benign: 0.5914
* Normal: 1.9705
* Malignant: 1.2355

### EfficientNetB0 w/Class Weights

A second run with EfficientNetB0 was done to see the difference once implementing the class weights.

A comparison of a graph

Description automatically generated

With the addition of the class weights, EfficientNetB0’s training accuracy steadily decreases instead of improving. The validation accuracy starts off much higher than the training accuracy and then drastically drops. This could indicate early overfitting followed by underfitting. It could also point to issues between the training and validation data as well as poor learning rates. The accuracy of this model on our test data set was 55.2 %.

### Custom CNN w/Class Weights

The addition of the class weights to our custom CNN model and the effect it had on its performance can be seen below.

A graph of a model

Description automatically generated with medium confidence

The loss curves look like our first run, which indicates that our model is still well tuned. The curve for the training accuracy looks similar as well, perhaps a little smoother. Although it is worth noting that in this version of the model, both the training and validation accuracy start off lower before increasing. The gap between the training and validation accuracy is narrower, indicating that that we may have improved the overfitting issue. However, it is difficult to tell from this alone whether the model is still overfitting or has now plateaued. We again generated a confusion matrix analysis on our model’s performance on the test data to further evaluate performance. This confirms that after addressing the issue of class imbalance within our dataset, our model is now predicting for all classes.

A diagram of a diagram

Description automatically generated with medium confidence

We then re-ran the Precision-Recall Curve to further evaluate how our model was now performing on predicting each class.

A diagram of a graph

Description automatically generated with medium confidence

Our model with class weights has improved for all classes in terms of the AP score. This version has high performance with identifying the benign class and moderate performance identifying the malignant class. The model is still performing poorly when it comes to identifying the normal class.

The accuracy of our custom CNN model on the test set was 76%.

## Conclusion

In this project, we employed a multi-layer sequential model to refine the output and obtain better results. We utilized L2 regularization and dropout to reduce the amount of overfitting. Learning rate optimizers and model checkpoints were used to help with validation loss.

Multiple other optimization techniques were used throughout this process, some we kept and some we discarded. One such discarded technique was data augmentation. This avenue initially caused issues and created blank images for the model to train on. We were able to correct that, but ultimately removed it as it did not improve the accuracy of the model.

The biggest issue we encountered was the uneven distribution of the dataset which required us to utilize class weights as the predicted outcomes were doing nothing better than guessing. Despite having a smaller sized dataset, our model performed well against the prebuilt EfficientNetB0 model.

While our model does not yet meet standards for deployment into the medical sphere for a diagnostic tool, we have shown that deep model learning and convolutional neural networks have considerable potential for being used in such a way. We believe that with more time and more data, models can be built to not only assist in diagnosing breast cancer from ultrasounds, but in many more applications in the medical field.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | EFFICIENTnETB0 | EFFICIENTNETB0 W/CLASS WEIGHTS | CUSTOM CNN | CUSTOM CNN W/CLASS WEIGHTS |
| accuracy | 0.5521 | 0.5521 | 0.8333 | 0.7604 |
| loss | 1.7069 | 1.7519 | 0.6951 | 1.0208 |
| AP Normal | - | - | 0.16 | 0.30 |
| ap benign | - | - | 0.56 | 0.83 |
| ap malignant | - | - | 0.28 | 0.63 |

# References

1. https://www.breastcancer.org/facts-statistics

2. https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/breast-ultrasound

3. https://www.mayoclinic.org/diseases-conditions/breast-cancer/diagnosis-treatment/drc-20352475#:~:text=Advertising%20%26%20Sponsorship-,Mammogram,at%20the%20area%20more%20closely.

4. https://www.kaggle.com/datasets/aryashah2k/breast-ultrasound-images-dataset. Al-Dhabyani W, Gomaa M, Khaled H, Fahmy A. Dataset of breast ultrasound images. Data in Brief. 2020 Feb;28:104863. DOI: 10.1016/j.dib.2019.104863.

5. <https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142>

6. [1905.11946] EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (arxiv.org)

7. https://keras.io/api/applications/efficientnet/