Breast Cancer Classification Using Convolutional Neural Networks

David Shimkus
800466279
dashimk@siue.edu

Ramani Janga
800728648
rjanga@siue.edu

Rupa Sai Mundru
800721726
rmundru@siue.edu

Abstract

Breast cancer is a frequently diagnosed cancer that affects many women all around the world and can also affect a subset of men. Especially with early detection, most types of breast cancer are treatable. If left untreated, various types of complications can arise including the spread of cancer to other organs. Pursuing different methods of cancer detection can be of vital importance to preventing serious health related symptoms.

Scientists are able to use a convolutional neural network to attempt to predict whether or not patients have breast cancer. Medical professionals can use infrared imaging techniques to obtain RGB image patches of breast tissue. After flagging each image as being breast cancer positive or negative, these images are able to be fed as training input into a learning model. After training the model, test data is evaluated against it to examine its accuracy. In this report it is shown how such a model can predict if a patch is breast cancer positive or negative with greater than 80% accuracy.

This accuracy result could be improved by adjusting parameters and overcoming other obstacles. By adjusting settings such as the batch size, learning rate, network layers, and more it can be observed how the accuracy fluctuates proportionately. Furthermore, it is shown how using additional processing resources and data could help increase accuracy of results.

Key words: breast cancer, IDC, invasive ductal carcinoma, identification, classification, convolutional neural network, CNN, accuracy, augmentation, Keras

1 Introduction

Cancer is a major cause of death worldwide, and breast cancer was the most common type of human cancer diagnosed in 2020 even though it is only prevalent in women [1]. More than half a million women die from breast cancer all over the world every year, and it is the number one cause of female cancer death as of 2018 [9]. While breast cancer can be fatal, it can also lead to various other complications such as the spread of cancer to other organs or mastectomies (removal of the breasts) [2]. The most common form of breast cancer is Invasive Ductal Carcinoma (IDC) [10]. Being able to detect this type of cancer in its early stages and begin treatment quickly improves patient prognosis significantly [3]. Therefore, improving upon computer aided detection (CAD) methods is important in minimizing patients' symptoms and can be seen as a worthwhile humanitarian effort.

Scientists and medical professionals can take images of breast tissue using near-infrared (NIR) light propagation techniques [3]. This is only one way to examine breast tissue for cancer, but its outputs are in an RGB image format that can be straightforward to analyze. Each image is then sliced up into patches, or smaller portions of the images partitioned to uniform dimensions. After these images have been gathered, they can be flagged as being IDC positive or negative. This type of

binary classification lends itself well to a deep learning model.

This project, IDCCheck, attempts to implement such a convolutional neural network (CNN) by building upon previous works and showing how prediction classification accuracy improves or degrades based upon adjustments to different parameters and methods. The input to this project's algorithm is a set of NIR patches of breast tissue. Data normalization and augmentation techniques are performed against these patches to curate data quality and enhance data quantity. The curated data is then fed into a CNN to train the model by updating the weights connecting each layer. The CNN is made up of multiple convolution layers with rectified linear unit (ReLU) activations. Max pooling, batch normalization, and dropout layers are also included in the CNN. At the end of the CNN is a dense (fully connected) layer which includes the "many-to-many" type of relationship that can be extremely effective at IDC positive or negative classification at the cost of additional overhead. The final output predicts whether the image is breast cancer positive or negative. Figure 1 shows the general flow of the input data to the final classification step.

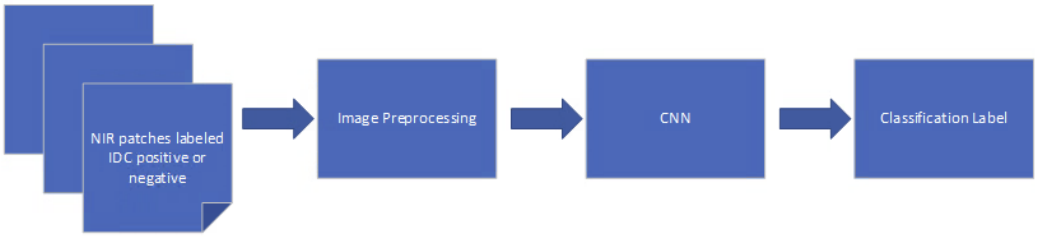


Fig. 1 - High level diagram of IDCCheck

2 Related Work

An example proof of concept project known as CancerNet has shown this type of model is able to classify breast tissue images as being IDC positive or negative with around 85% accuracy [4]. This project uses the Python programming language with the Keras, Tensorflow, Numpy, and related GPU enabled deep learning libraries to implement its solution. However, due to the ever-changing nature of this type of work, getting the now-outdated CancerNet project to work on modern and up to date versions of Python 3, Tensorflow, etc. proved to be a challenge and major roadblock. However, the core concepts and structure remain valid though and were useful inspiration for this IDCCheck project. The hidden layers of the IDCCheck CNN architecture itself were built with the general CancerNet makeup in mind with some modifications.

It is important to note that NIR techniques for analyzing breast tissue are not the most common forms of screening patients for cancer. Traditional mammograms (human breast x-rays) are still the most prevalent methods of screening and are endorsed by the Food and Drug Administration (FDA) [5]. However, this x-ray technique has a high rate of false negatives due to factors such as biological diversity and tissue density [6]. Two other pre-trained Convolutional Neural Networks known as Mobilenet and Nasnet have been tested using mammogram data with classification accuracy results of 78.4% and 74.3% respectively [6]. This alludes to the fact that there may be room for improvement using different types of data gathering techniques, such as NIR, for training.

Another related concept is the phenomenon of tissue superposition, in which different tissues of the breast are projected onto the same location in the mammographic image due to the angle in which the image is taken [9]. This can potentially cause good tissues to cover up bad tissues or vice versa, leading to incorrect evaluations. A workaround to this roadblock is to examine the breast at two different views (angles) and evaluate them together to arrive at a more confident conclusion [9]. While IDCCheck does not implement this due to the data used, it would be quite an interesting approach to add another dimension to the entire CNN for pairs of images. Previous work has indicated that including such a dimension can increase overall accuracy by as much as 5% in some cases [11]. Without loss of generality this concept could theoretically be expanded to multiple dimensions, potentially further increasing accuracy at the cost of exponentially increasing required processing.

3 Materials and Methods

The IDCCheck project had many initial challenges with its implementation. But the core ideas and concepts remained consistent throughout.

3.1 Datasets and Features

To train the IDCCheck model the primary data that was used is from a Kaggle repository that cites the data as originally coming from Case Western Reserve University [7]. The Kaggle data consists of 277,524 RGB images of 50x50 dimensions, of which 198,738 are IDC negative and 78,786 are IDC positive. These images are patches (slices) of a whole slide image of breast cancer specimens scanned at 40x. The patches are logically grouped by a unique patient identifier and separate folders indicating 0 for IDC negative and 1 for IDC positive.

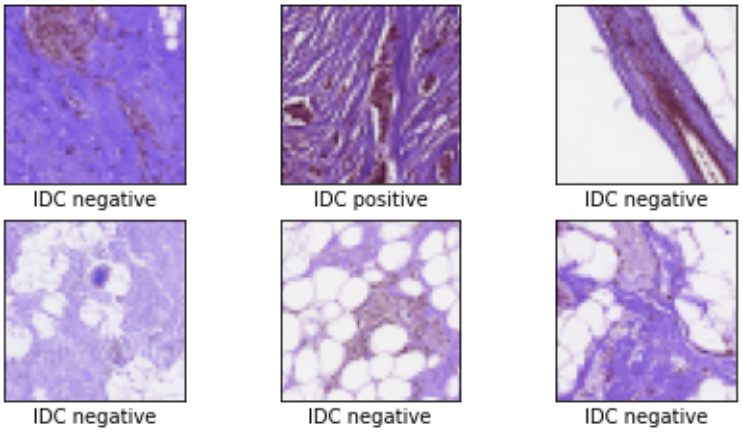


Fig. 2 - Example NIR breast tissue patches

This data from the Kaggle repository is considered “raw” and undergoes a series of preprocessing steps prior to being fed into the IDCCheck network. Each patch is first normalized where pixel intensities are rescaled and clamped to a decimal in a 0 to 1 inclusive range (where 0.0 would be white, and 1.0 would be black). This data set is largely unbalanced with IDC negative patches outnumbering IDC positive patches by more than two to one. Therefore, to balance the data and eliminate unnecessary bias prior to training the model there is an under-sampling step performed. The under-sampling step consists of discarding many IDC negative images at random, resulting in equal counts for each classification. After this is done there are a total of 157,572 images left.

The last preprocessing step is an intended data augmentation step, where each image is used to generate 32 additional images. These generated images take the original normalized image as input and apply randomized rotations, shifts, and flips to come up with additional data points. These images do not exist on disk and are generated in memory by the Keras libraries. The result is a set of 5,199,876 images that are still balanced with respect to positive versus negative images as the balancing was performed prior to the augmentation step. Figure 3 captures all of the steps it takes to get to this final data set.

Originally this data was being bucketed into an 80/10/10 split of training/validation/testing respectively. Ultimately, the IDCCheck code had this data split 70/30 into training/testing buckets for a baseline default and a specific validation set is not used.

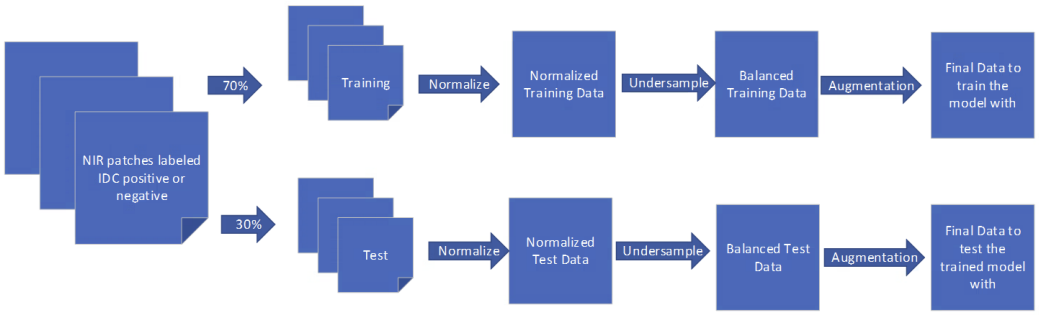


Fig. 3 - Data preprocessing flow

3.2 Methods

3.2.1 Architecture Details and Challenges

The IDCCheck system was implemented using Python 3 and various supporting libraries such as Tensorflow, Keras, Numpy, and other standard deep learning modules. However, the inspiration and groundwork for this project, CancerNet, was implemented using outdated libraries and was not able to run on current out of the box instances without forcing deprecated packages to install. The team spent many hours debugging and stepping through the CancerNet code and attempting to get a successful run. Ideally, the hours invested into debugging the CancerNet code would have been allocated to adjusting hyperparameters and other configurations. After heavy modification the CancerNet code was able to run without crashing in a Python 3 runtime, but the results were inaccurate and could not reflect the tutorial and stated classification accuracy of around 85%. At that time, it was decided to abandon the CancerNet code but retain the core concept such as the outlined CNN it contained.

Testing was originally performed using a university provided Linux server home.cs.siue.edu. This was impractical as this server is not GPU enabled. After this was discontinued, a personal NVIDIA GPU home Linux PC was utilized to moderate success to implement some of the core Keras and CUDA features which significantly reduced the time needed to train the model. However, using this PC was impractical however as it was only available to one team member and bringing the PC to campus and organizing group sessions was still thrown into some doubt as the COVID-19 pandemic restrictions persisted in the beginning of the semester.

Eventually, the project was migrated to Google Colab with a GPU enabled runtime. Additional challenges were encountered here with loading the data. A standard runtime only allows for 12 GB system RAM (not GPU RAM) and the image preprocessing method that was implemented required around 17 GB RAM. For the proof of concept, the available data is reduced to about two-thirds of the total data by default to mitigate this limitation. A team member purchased a Google Colab Pro license to run the IDCCheck network against the full dataset with consistently higher classification accuracies. Some of these results are captured in the conclusion section.

3.2.2 CNN Details

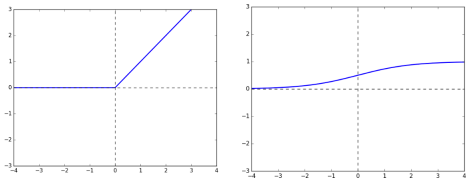
One of the most important parts of any CNN project is the makeup of its hidden layers. The IDCCheck network makes use of multiple types of layers including convolutions, batch normalizations, max pooling, and dropout components. Also used before the final classification are the flatten and fully connected layer steps. All convolutional layers use the Rectified Linear Unit (ReLU) activation function, and the final dense layer uses a sigmoid activation function right before the final classification.

```
cnn_model = Sequential()

cnn_model.add(Conv2D(32,(3,3),padding="same",activation="relu",input_shape=(50,50,3)))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(2,2))
cnn_model.add(Dropout(0.25))

cnn_model.add(Conv2D(64,(3,3),padding="same",activation="relu"))
cnn_model.add(BatchNormalization())
cnn_model.add(Conv2D(64,(3,3),padding="same",activation="relu"))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(2,2))
cnn_model.add(Dropout(0.25))

cnn_model.add(Conv2D(128,(3,3),padding="same",activation="relu"))
cnn_model.add(BatchNormalization())
cnn_model.add(Conv2D(128,(3,3),padding="same",activation="relu"))
cnn_model.add(BatchNormalization())
cnn_model.add(Conv2D(128,(3,3),padding="same",activation="relu"))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(2,2))
cnn_model.add(Dropout(0.25))

cnn_model.add(Flatten())
cnn_model.add(Dense(256,activation="relu"))
cnn_model.add(BatchNormalization())
cnn_model.add(Dropout(0.5))

cnn_model.add(Dense(1,activation="sigmoid"))
```

Fig. 4 - Default IDCCheck CNN in Google Colab

Examining figure 4 shows the input shape on the first line corresponds to the 50x50 3 channel RGB NIR tissue patches. The Conv2D function creates a convolution kernel that is convolved with the layer's input to send its output to the next layer. The first parameter is the dimensionality of the output space. The second parameter is the kernel size. A kernel is sometimes called a filter, and it specifies the height and width of the convolution window to be applied against the input. Note that there is no stride parameter specified so the Keras framework defaults the stride to (1,1). The "same" value for the padding parameter indicates that zeros are evenly padded surrounding the patch's sides when necessary.

Almost all of the CNN's activation functions are the ReLU with the notable exception of the very last layer being a sigmoidal (logistic) function. These functions are ways to force certain boundaries into the model so that the weights cannot explode or cause more harm than benefit. Figure 5 shows the visual representation and formulas of these activation functions.



Rectified Linear Unit (ReLU)

$$y = \max(0, z)$$

Logistic

$$y = \frac{1}{1 + e^{-z}}$$

Fig. 5 - ReLU and Sigmoidal (Logistic) activation functions

The next layer used in the CNN is a batch normalization layer with all default parameters. This layer applies a transformation that maintains the mean output close to zero and the output standard deviation close to one. In general, using a batch normalization layer right after a convolution or

dense layer is a good idea because it helps keep the output from that layer manageable. This helps keep values from spinning out of control in either direction as they propagate through the model.

The max pooling layers all have a (2,2) pool size parameter in this model. This layer "downsamples" the data by taking the maximum value over chunks of the input. This essentially summarizes portions of the image and helps reduce noise.

Dropout layers are occasionally applied which sets input units to zero based on a rate defined by the frequency parameter. Alongside this operation all of the other inputs are scaled up by 1/(1-rate). These layers essentially help "emphasize" certain portions of data while de-emphasizing other portions. For example, all dropout layers in this model except for the last one have the dropout rate set to 0.25. In terms of RGB images, one might be able to think of this as setting certain pixels to pure white, and darkening the other pixels.

The last layer types to mention are "flatten" and "dense" which do what their names imply. The flatten layers simply change the shape of the input object to appropriately match the input shape to the dense layer. The dense layer is the "many to many" type of layer where every single portion of the input is weighted against every single element in the layer. The dense operation can certainly help provide more accurate classifications but it is more costly. This is why convolution layers prior to dense layers can be helpful as they are usually less resource intensive.

By default the IDCCheck project sets the first Conv2D layer's output dimensionality to 32, with the following convolution layers doubling this value. The team experimented with adjusting these dimension values with varying results.

4 Results and Discussion

By using the entire data set available on Kaggle the IDCCheck project is able to correctly classify the test portion of the data with an average accuracy of > 80%. Adjusting the hyperparameters and amount of data certainly causes this accuracy to fluctuate as discussed below. Furthermore, training the model with 100 epochs almost always resulted in an overall training accuracy of around 85%. The baseline default parameters include a 70/30 train/test split of the data, a batch size of 32, and a learning rate of 0.001. The baseline results and small tweaks to these parameters are summarized in Table 1.

Total IDC negative Images	Total IDC positive Images	Training data split	Testing data split	Batch Size	Learning Rate	Number of Epochs	Runtime (rounded down)	Test Loss	Test Accuracy
198738	78786	0.7	0.3	32	0.001	100	13 minutes	0.4073	0.8212
198738	78786	0.7	0.3	16	0.001	100	14 minutes	0.5845	0.7979
198738	78786	0.7	0.3	32	0.01	100	13 minutes	0.4368	0.8158
198738	78786	0.8	0.2	32	0.001	100	16 minutes	0.3629	0.8504

Table 1 - Initial classification results

Seeing these types of figures is a strong indicator that this IDCCheck project was working similarly to the precursor project CancerNet. The CancerNet stated classification accuracy was also around 85% for the same dataset so this project was on the right track. It was noticed that the data split of 80/20 led to a better test accuracy so this split was retained for future training. The team also observed that the learning rate and batch size adjustments led to a worse test accuracy classification,

so changing these values in their respective directions was discontinued.

Overall, these differences in parameters were not too significant for the loss or accuracy while training the model. Figure 6 below summarizes how the accuracy and loss was adjusted per each epoch. It is interesting to note that the learning rate of 0.01 has wild spikes on certain epochs. This makes sense as a strong learning rate can sometimes be deleterious in that it can push a model towards inaccurate results until corrected. However, these adjustments did not influence the overall curve significantly, partially due to the normalization, pooling, and dropout that occurs in the CNN hidden layers.

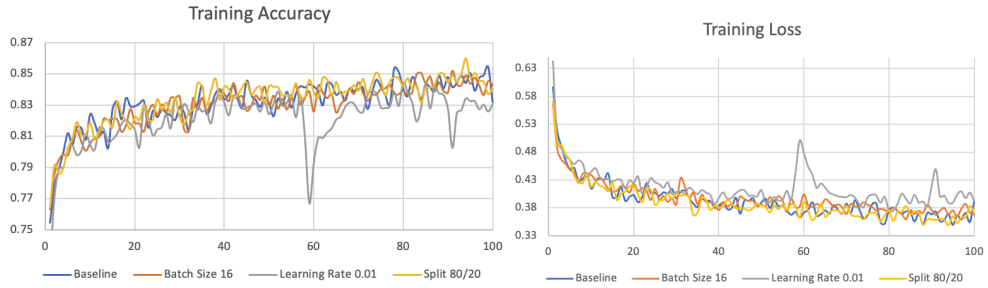


Fig. 6 - Training Accuracy and Loss with different parameters

Generally speaking, the more epochs one is able to perform, the more the overall accuracy of the training set will improve as well. However, in this case doubling the epochs to 200 and making the 80/20 split in the data between training and validation led to similarly favorable results of a test accuracy of 84%. Figure 7 shows the training accuracy and loss as displayed in the Colab notebook using the Matplotlib library and illustrates the overall trend increasing and decreasing respectively.

Fig. 7 - 200 epoch run using default configurations with an 80/20 split of the data

Another hyperparameter to consider is the number of hidden layers in the network itself. Referring back to Figure 4, it can be seen how there are multiple copies of the same type of convolution that are repeated in different groupings. In an attempt to determine the relevancy of these layers they were commented out and another execution was run with all other parameters set to the baseline to compare the training accuracy and loss. This slimmed down model has been denoted as the “less layer” model and is shown below in Figure 8.

```
cnn_model.add(Conv2D(32,(3,3),padding="same",activation="relu",input_shape=(50,50,3)))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(2,2))
cnn_model.add(Dropout(0.25))
cnn_model.add(Conv2D(64,(3,3),padding="same",activation="relu"))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(2,2))
cnn_model.add(Dropout(0.25))
cnn_model.add(Conv2D(128,(3,3),padding="same",activation="relu"))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(2,2))
cnn_model.add(Dropout(0.25))
cnn_model.add(Flatten())
cnn_model.add(Dense(256,activation="relu"))
cnn_model.add(BatchNormalization())
cnn_model.add(Dropout(0.5))
cnn_model.add(Dense(1,activation="sigmoid"))
```

Fig. 8 - IDCCheck “less layer” model

The results are quite similar to what was previously noted using all of the default layers, but the “less layer” run has a slightly more noticeable variability in its values. If conserving processing resources is necessary, it might be feasible to utilize the simplified “less layer” implementation to arrive at comparable results. Figure 9 compares the similarities between these two models.

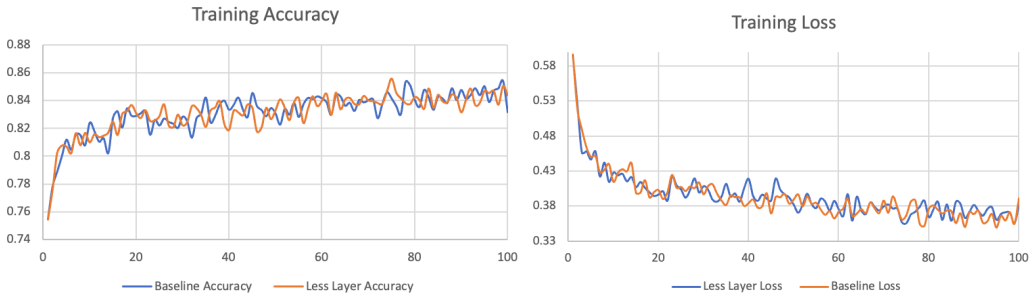


Fig. 9 - Comparison of baseline and “less layer” model

5 Conclusions and Future Work

Breast cancer is a very serious condition which affects many people; therefore pursuing computer assisted detection methods is warranted. The NIR IDC breast tissue patch data set is a very thorough data set and lends itself extremely well to these types of learning models. Even with many challenges, this IDCCheck project was consistently able to train a model to classify images as IDC positive or negative with greater than 80% accuracy. This accuracy could potentially be improved upon by increasing the amount of data used, as well as potentially adding in a different training dimension such as superimposing two different angles of breast tissue images.

If more time was available for this project, it would be prudent to better understand and harden the data preprocessing steps. This project is sufficient to replicate the groundwork that was laid with the original CancerNet implementation, but it does not notably improve upon its accuracy. Modifying the preprocessing step appropriately would potentially reduce the need for system RAM as well, allowing a standard Colab runtime with less than 18GB of RAM to train against the entire normalized, augmented data set.

6 References

[1] Jacques Ferlay, M Ervik, F Lam, M Colombet, L Mery, M Piñeros, A Znaor, I Soerjomataram, and F Bray. Global cancer observatory: cancer today. Lyon, France: international agency for research on cancer, pages 1–6, 2018.

[2] "Surgery for Breast Cancer: Breast Cancer Treatment." American Cancer Society, https://www.cancer.org/cancer/breast-cancer/treatment/surgery-for-breast-cancer.html.

[3] Cong, Wenxiang et al. "Optical tomographic imaging for breast cancer detection." Journal of biomedical optics vol. 22,9 (2017): 1-6. doi:10.1117/1.JBO.22.9.096011

[4] "Project in Python - Breast Cancer Classification with Deep Learning." DataFlair, 14 Mar. 2021, https://data-flair.training/blogs/project-in-python-breast-cancer-classification/.

[5] "Breast Cancer Screening: Thermogram No Substitute for Mammogram." U.S. Food and Drug Administration, FDA, 13 Jan. 2021, https://www.fda.gov/consumers/consumer-updates/breast-cancer-screening-thermogram-no-substitute-mammogram.

[6] Lenin G Falconí, María Pérez, and Wilbert G Aguilar. Transfer learning in breast mammogram abnormalities classification with mobilenet and nasnet. In 2019 International Conference on Systems, Signals and Image Processing (IWSSIP), pages 109–114. IEEE, 2019.

[7] Mooney, Paul. "Breast Histopathology Images." Kaggle, 19 Dec. 2017, https://www.kaggle.com/datasets/paultimothymooney/breast-histopathology-images.

[8] Ajagekar , Akash. "Adam." Cornell University Computational Optimization Open Textbook - Optimization Wiki, Cornell University, 2021, https://optimization.cbe.cornell.edu/index.php?title=Adam.

[9] Ioannis Sechopoulos, Jonas Teuwen, Ritse Mann. "Artificial intelligence for breast cancer detection in mammography and digital breast tomosynthesis." State of the art, Seminars in Cancer Biology, Volume 72, 2021, Pages 214-225, ISSN 1044-579X, https://doi.org/10.1016/j.semcancer.2020.06.002. (https://www.sciencedirect.com/science/article/pii/S1044579X20301358).

[10] The American Cancer Society medical and editorial content team. "Invasive Breast Cancer (IDC/ILC)." American Cancer Society, 19 Nov. 2021, https://www.cancer.org/cancer/breast-cancer/about/types-of-breast-cancer/invasive-breast-cancer.html.

[11] Van Engeland, Saskia, and Nico Karssemeijer. "Combining Two Mammographic Projections in a Computer Aided Mass Detection Method." Medical Physics, vol. 34, no. 3, 2007, pp. 898–905., https://doi.org/10.1118/1.2436974.

7 Contributions

David - CancerNet debugging, IDCCheck debugging, experiment data gathering, report/presentation writing, planning

Ramani - IDCCheck skeleton code, presentation writing, LaTeX assistance

Rupa - IDCCheck skeleton code, CancerNet testing, IDCCheck testing, planning, LaTeX assistance