Breast Cancer Diagnosis: IDC Aggressiveness Detector

June 14, 2018

Contributors
Ryan Fong, RRFONG@UCI.EDU
Kevin Maxion, KMAXION@UCI.EDU

Introduction and Problem Statement

Breast Cancer is the most common cancer for women. The most common *subtype* of breast cancer is Invasive Ductal Carcinoma (IDC). Lethal if left untreated, IDC penetrates the milk duct wall and attacks breast tissue. That's why accurately diagnosing breast cancer is such a critical clinical task. To combat this devastating disease, we created a machine learning model that takes a patient's breast histopathology images and determines the aggressiveness of their IDC. The Kaggle image dataset comes partially preprocessed in the form of 40x magnified subpatches of each patient's whole mount slide image. Our project leverages a convolutional neural network (CNN) who's architecture includes: 3x3 convolutional layers, 2x2 max pooling layers, relu activations, batch normalization, dropout regularization, and affine fully connected layers. After binary classifying each of the patient's subpatches as IDC positive or negative, we output the percentage of positive images out of the total as their aggressiveness score.

We were able to achieve a favorable test accuracy of 86% on IDC classification. Our observations conclude that there is no pattern in the ratio of false positives to false negatives.

Related Work

In 2012, Dan C. Ciresan used a deep max-pooling convolutional neural network (DNN) to detect mitosis in breast cancer histology images. His input is an RGB image and his goal was to classify whether or not mitosis was taking place in the image. His feed-forward DNN consisted of stacked pairs of convolutional and max-pooling layers, followed by a couple fully connected layers. Since our project also involves binary classification of breast cancer histology images, we decided to test out a similar approach to Ciersan's. In addition to other layers, we include in our model two stacked pairs of 3x3 convolutional and 2x2 max-pooling layers. We also include two affine fully connected layers.

In 2014, J. Dheeba experimented with a Particle Swarm Optimized Wavelet Neural Network (PSOWNN) to solve a very similar problem. This breast cancer detection algorithm is based off extracting Laws Texture Energy Measures from mammograms and identifies abnormal subsections by applying a pattern classifier. The wavelet transformation technique can analyze information from signals that are noisy or aperiodic in a feed-forward neural network. Researchers first applied a 1x5 1-D convolutional layer. That layer then generated the next 2-D convolutional layer inputs for texture discrimination. These kernels are *zero averaged* and are sensitive to edges, lines, and spots. We decided to take this information and base our architecture off it. As mentioned in the previous paragraph, we include two stacked pairs of convolutional and max-pooling layers. Similarly to how Dheeba zero averaged his kernels, we decided to perform batch normalization after our second convolutional layer to achieve a similar

goal of zero mean and unit variance. After adding the batch normalization layer, our model performed roughly 5-10% better on test data.

Cireşan, Dan C., et al. "Mitosis detection in breast cancer histology images with deep neural networks." International Conference on Medical Image Computing and Computer-assisted Intervention. Springer, Berlin, Heidelberg, 2013.

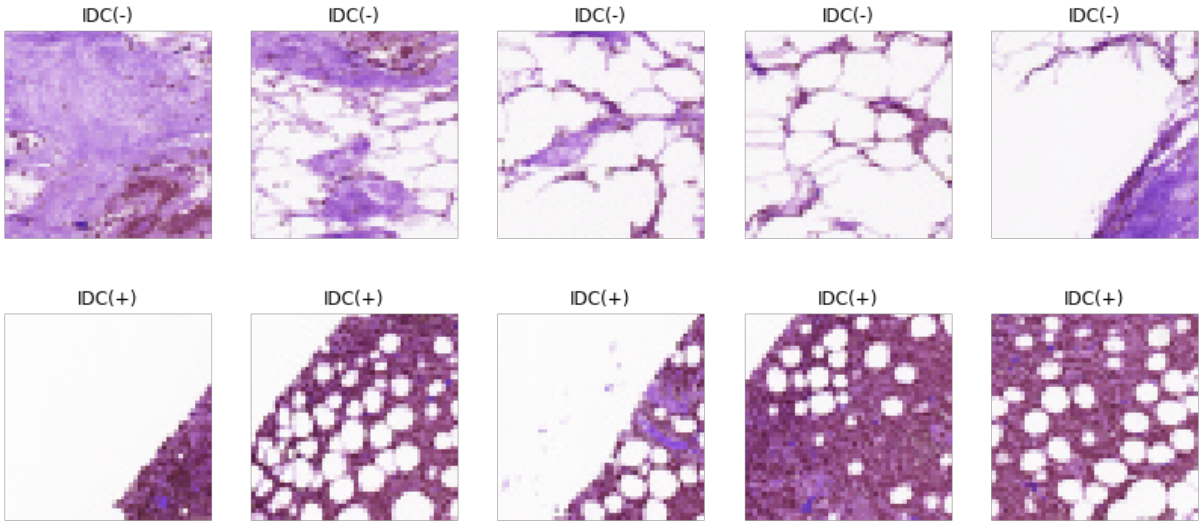
Dheeba, J., N. Albert Singh, and S. Tamil Selvi. "Computer-aided detection of breast cancer on mammograms: A swarm intelligence optimized wavelet neural network approach." Journal of biomedical informatics 49 (2014): 45-52.

Data Sets

Data URL: https://www.kaggle.com/paultimothymooney/breast-histopathology-images

Our entire Kaggle dataset consists of 162 whole mount slide images of breast cancer specimens broken up into 40x magnified subpatches. There are 277,524 subpatches, each of size 50 x 50. The distribution of positive to negative images is uneven, favoring negative (198,738 IDC negative and 78,786 IDC positive). We believe that this data imbalance may have resulted from the researchers not including enough people from the range severe to lethal IDC aggressiveness. Images are first separated by patient, then by classification. Since our dataset is too big to train on any free GPU (memory usage issues), we're downsizing our data to under 35k subpatches split among train and test.

Contrary to the hand pose and fruit datasets, IDC subpatch classification cannot be done with a naked eye (unless you're a doctor). Accurate human diagnosis comes from years of training and experience. However, this model can learn all the histopathology patterns in minutes.

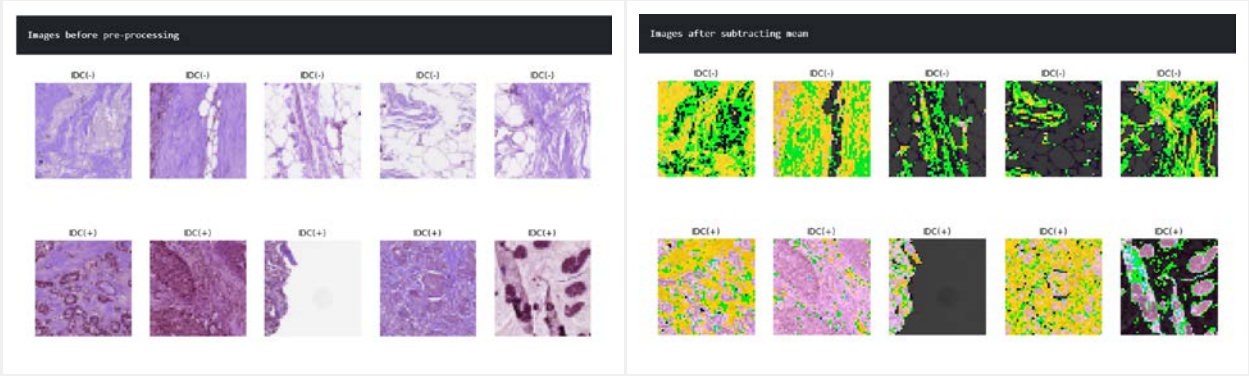


Technical Approach

<u>Preprocessing data</u>

Our first step was making sure that the data we had is ideal for training and testing on the neural network model. A significant portion of the pre-processing step was already done for

us by how the data is set up. That is, having equally magnified and split 50 x 50 subpatches of the whole histology slides, which gives us two benefits: firstly, we don't have to deal with large image sizes, but instead with localized patches of the whole slides. Secondly, it gave us consistent and relevant data. In some of the research papers, we read that that pre-processing step is one of the most significant, so we were fortunate that it's been done for us. Once we loaded the subpatches, we stored them in a vector. Then we normalized our data by subtracted the image mean from the entire dataset. We this this in order to normalize the color variation of the patches, in order to account for the fact that the colors of different histopathology slides can slightly vary depending on how the source of the histopathology slides processed them. Lastly, since the distribution of the Kaggle dataset favored negatively classified images and we had a huge dataset of 277,524 patches, we created a subset of that data where the number of positive and negative images are equal. We did this to prevent the model from being more inclined to classify patches as negative.



Different attempts

Building our best CNN model didn't happen first try. Trial and error helped us slowly gain a better understanding of how to use the different softwares. To first get baseline results, we attempted to copy and paste our assignment 4 TensorFlow.ipynb "complex_model" along with associated testing code into our notebook. Upon our first run with only 1000 total images for train and test, we achieved 100% test accuracy. We later concluded that our perfect test score was the result of not having enough images. After bumping our training set up to 50k images, our test accuracy dropped to 68%. That became the starting point for our CNN model.

We then started tweaking hyperparameters using GridSearchCV and nested for-loops (explained in #5) which brought our test accuracy up to 74%. Once we found the optimal parameters, we decided to start adding layers. However, TensorFlow's documentation was confusing and made adding layers extremely difficult. After hours of research and attempting to solve math equations by hand, we gave up on our TensorFlow model and translated the architecture to Keras code. Keras reduced the number of lines required to define our model by almost 2/3. It also made our layers easier to weave together and much more readable.

In attempt to go the extra mile, we experimented with ensembling our CNN with a Least Squares Support Vector Machine (LSSVM). In 2006, engineers in the Electrical and Electronics Engineering Department at Selcuk University utilized a LSSVM to diagnose breast cancer with

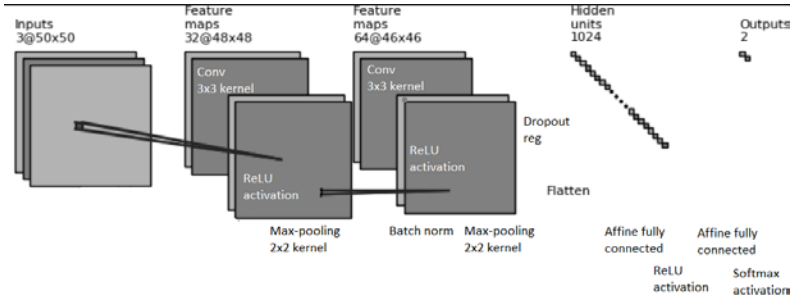
98.5% accuracy (Polat). However, when we attempted to integrate a similar approach into our ensemble, the LSSVM consistently scored lower than our CNN. As a result, we decided to ditch the LSSVM method entirely and just focus on our CNN.

As for data pre-processing, one of the algorithms we explored and experimented with, but decided not to keep was data augmentation. We had a large amount of data, so we already have a lot of variation in images for each negative and positive patches, so it didn't have any significant effect on our results.

Our final model

Our finalized CNN detects IDC in subpatches with ~**87%** accuracy. Inspiration for our final model architecture came from related scholarly work and the assignment 4 complex_model. Our model starts with a 3x3 convolutional layer with 32 filters, followed by ReLU activation, and a 2x2 max-pooling layer. The next 4 layers have almost the same structure, except the convolutional layer now has 64 filters and there is a batch normalization layer between the ReLU activation and max-pooling. The batch normalization transforms the data to be zero mean and variance, thus improving the processing at the next layer. After that, we included a dropout regularization layer which sets some weights to zero to make our model more robust to overfitting. We felt like this layer would be beneficial since our model is getting increasingly more complex. Following the dropout layer are two affine fully connected layers. The first fully connected layer has 1024 units and *ReLU* activation. That layer then feeds the second fully connected layer that has only two output units and *Softmax* activation. The two different activation functions yielded higher average test accuracy when compared to two of the same activation functions. The second connected layer's two output units then become our CNN model output. The zero-th and first indexes of the output are the probabilities of the image being negatively and positively classified, respectively. We classify the image as whichever probability is higher. But our project doesn't quite end there.

Once we have the vector of classifications for each subpatch, we then calculate the proportion of the patient's positive images to the total number of images. The resulting decimal value between 0-1 (inclusive) becomes the patient's IDC aggressiveness score.



Polat, Kemal. "Breast Cancer Diagnosis Using Least Square Support Vector Machine."
Electrical and Electronics Engineering Department, 27 Nov. 2006, pp. 694–701.

Experiments and Evaluation

Methods and metrics:

The dataset we used for this project doesn't come with any explicit test data, so throughout our whole project we used a split of 80% training data and 20% test data, with varying amounts of total data. We made sure to shuffle our data prior splitting it every time to make sure our model wasn't just working well on one particular subset of the data. Prior to starting, we were planning to use 5-fold cross validation, but ultimately decided on a training/test split because using cross validation with our big data set would've taken too long. We used the following metrics and methods:

Brute force hyperparameter tuning: In order to find our ideal hyperparameters, we used a nested for loop of testing learning rates, number of epochs, and batch size instead of tuning them one by one. Each iteration printed out the train and validation accuracy, as well as a graph showing the progression of the loss and accuracy by epoch. In addition to our loop printing the parameters with the best validation/testing score, we analyzed the printed information to ensure it makes sense.

Confusion matrix: Since our problem deals with classification of positives and negatives, we thought that one of the ways to measure the effectiveness of our model is through a confusion matrix, which displays the number of true positives, true negatives, false positives, and false negatives.

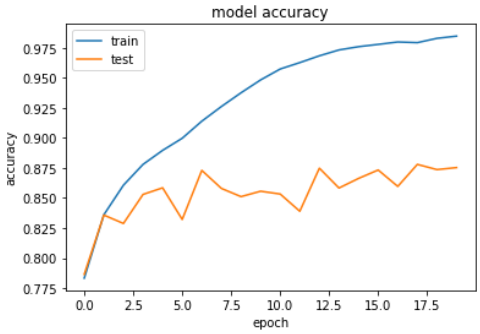
Results:

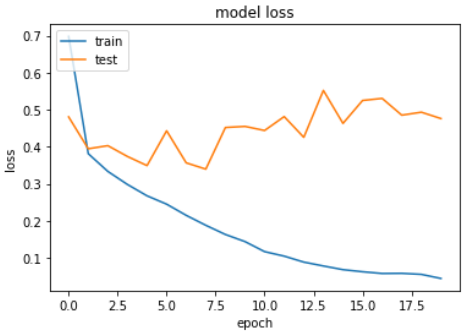
With our brute force hyper parameter tuning loop, we came to the following conclusions:

Our model had the best results with the parameters:

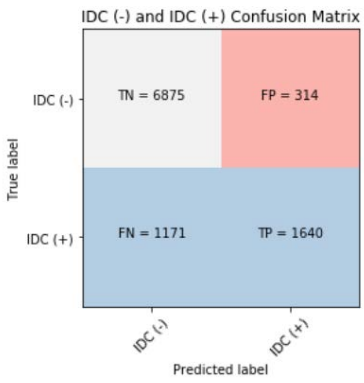
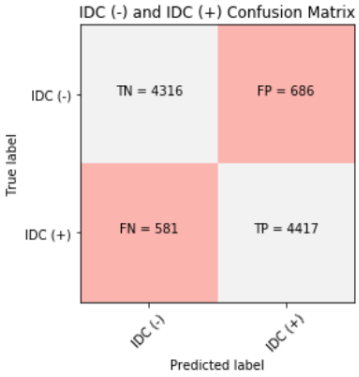
- Epochs: 20
- Batch-size: 64
- Learning rate: .0005

The graph on the right shows the model accuracy and model loss of these parameters. We found that out of these three parameters, the learning rate of our Adam optimizer had the biggest impact on the results.





Through our confusion matrix we found out that training our model on an imbalanced amount of negative and positive samples was causing it to be more biased to being false negative. We found that purposely training it on exactly half positive and half negative data, we came to significantly more even and consistent results.

Before	After
	

In the end, our Convolutional Neural Network Model had a test accuracy of 87%.

6. Discussion and Conclusion

After learning about neural networks for a quarter, we thought we had a firm grasp on what it takes to build a good model. Now in hindsight, our model never even reached a 90% test accuracy, despite constant additions and modifications. We knew that a convolutional network would do well in an image classification problem like this, but what surprised us was that once we reached a certain threshold of success, it took a lot of experimenting and effort to incrementally progress from that point. It definitely showed us that we have much more to learn on this topic.

We faced two big limitations in this project: limited technical resources and limited expert knowledge. In order to utilize a free GPU, we decided to train and test our model on Kaggle Kernel, which provided small limited memory usage with each run. Although the training of our data went fast, we couldn't use the entirety of our dataset to train our model. The other problem was that we had limited knowledge on what really determines the presence of IDC within a histopathology slide. We fully relied on our model and common pre-processing practices to understand and classify IDC for us. If we had more time or the help of a histopathology expert, we would be able to more specifically preprocess our data to emphasize the features that are most important in the detection.

Of course, we've put some thought into what our project could've been, and what it could become in the future. One thing that came to mind was that it should be able to handle whole slide images as input instead of vectors of subpatches. Another one was the detection of multiple subtypes of breast cancer than IDC. Those additions would make our project more useful in the real world industry, as data wouldn't have to passed pre-processed into the model, and it would be able to detect less common forms of breast cancer as well. We realize that our current model is not even close to ready for a real world application. For a task as sensitive as cancer detection, a 87% test accuracy is not even nearly good enough. Although we were

happy to see many improvements along the way of our project, we realize that lot more work has to be put into the model for it to be where we want it to be.